Package ‘ggasym’

May 15, 2021

Type Package

Title Asymmetric Matrix Plotting in 'ggplot2'

Version 0.1.6

URL https://github.com/jhrcook/ggasym
https://jhrcook.github.io/ggasym/

Description Plots a symmetric matrix with three different fill aesthetics for the top-left and bottom-right triangles and along the diagonal. It operates within the Grammar of Graphics paradigm implemented in 'ggplot2'.

License GPL-3

Encoding UTF-8

Depends R (>= 3.6.0)

Imports dplyr (>= 0.8.0), ggplot2 (>= 3.1.0), magrittr (>= 1.5.0), purrr (>= 0.3.0), rlang (>= 0.3.0), scales (>= 1.0.0), stringr (>= 1.4.0), tibble (>= 2.0.0), tidyr (>= 0.8.0)

RoxygenNote 7.1.1

Suggests knitr, rmarkdown, testthat, spelling, covr, broom (>= 0.5.0), corrr (>= 0.3.0)

VignetteBuilder knitr

Language en-US

BugReports https://github.com/jhrcook/ggasym/issues

Collate 'asymmetrise.R' 'asymmetrise_stats.R' 'geom_asymmat.R' 'ggasym.R' 'scale_continuous_asym.R' 'scale_fill_gradient.R' 'utils.R'

NeedsCompilation no

Author Joshua H Cook [aut, cre] (<https://orcid.org/0000-0001-9815-6879>)

Maintainer Joshua H Cook <joshuacook0023@gmail.com>

Repository CRAN

Date/Publication 2021-05-15 20:10:02 UTC

R topics documented:

`add_missing_combinations`

Add missing combinations of x and y to a data frame

Description

Add rows to `df` to complete all combinations of columns `.x` and `.y`. Importantly, this function observes and maintains any groups created by `dplyr::group_by()`.

Usage

```
add_missing_combinations(df, .x, .y)
```

Arguments

`df`	a data frame (or tibble) object
`.x, .y`	column names to make combinations of

Value

a data frame (or tibble) with additional columns

Examples

```
df <- data.frame(
  a = c("A", "B"),
  b = c("C", "D"),
  untouched = c(1, 2)
)
df

add_missing_combinations(df, a, b)
```

`asymmetrise` *Add all missing comparisons between two columns*

Description

This function prepares input data for `geom_asymmat()` by adding in any missing comparisons to be plotted. Note that this function observes groups created with the `dplyr::group_by()` function. For the 'ggasym' package, this is useful for when you want to facet the plot: before "asymmetrizing" the data table, use `dplyr::group_by()`, passing the column name you wish to later facet by. This functionality is demonstrated in the second example, below.

Usage

```
asymmetrise(df, .x, .y)

asymmetrize(df, .x, .y)
```

Arguments

`df`	a tidy `data.frame` or `tibble`
`.x, .y`	the data to add all comparisons between (ie. will be the x and y-axes for `geom_asymmat()`

Value

a data table with new rows for the added comparisons

Warning

This function does it's best when `x` or `y` are factors. If they have the same levels, then they are maintained. If the levels partially overlap, they are merged. Otherwise, the values are turned into characters and all levels dropped. If you are using factors, save yourself the headache and make both columns factors with the desired levels.

Examples

```
df <- data.frame(
  a = c("A", "B", "C"),
  b = c("C", "A", "B"),
  untouched = c(1, 2, 3),
  grouping_value = c("group1", "group1", "group2"),
  stringsAsFactors = FALSE
)
df

asymmetrise(df, a, b)

grouped_df <- dplyr::group_by(df, grouping_value)
asymmetrise(grouped_df, a, b)
```

---

asymmetrise_stats *Prepare an asymmetric data table from a statistical test*

Description

This function prepares the results of a statistical test for plotting using ‘geom_asymmat‘ from the ggasym package. For more information, see `vignette(ggasym-stats)`

Usage

```
asymmetrise_stats(df, contrast_sep = "-")

asymmetrize_stats(df, contrast_sep = "-")
```

Arguments

`df`	either the results of a statistical test or the tidy tibble from using the `broom::tidy()` function
`contrast_sep`	the separation used between the names being compared; it is usually a hyphen (set as default here); since it is passed as the `pattern` parameter to `stringr::str_split_fixed()`, this can be any regular expression that will reliably split the `"contrast"` (or `"comparison"` in 'broom' version <0.70) column returned by `broom::tidy()`.

Value

a tibble object that can be used as direct input for 'ggplot2' for use with the `geom_asymmat` geom

`bind_missing_combs` *Add the missing combinations of x and y*

Description

Adds rows to the input data table to include any combinations of `.x` and `.y` that are not already present. All other columns (if any) are set to `NA`

Usage

```
bind_missing_combs(df, .x, .y)
```

Arguments

`df`	input data table
`.x, .y`	names of the columns for which to add missing comparisons

Value

a data table with the new rows

Examples

```
df <- data.frame(
  a = c("A", "B"),
  b = c("C", "A"),
  untouched = c(1, 2),
  stringsAsFactors = FALSE
)
df

bind_missing_combs(df, a, b)
```

`continuous_scale_asym` *Continuous scale constructor for 'ggasym'*

Description

This is a this wrapper around `continuous_scale()` from the 'ggplot2' package. It is generally best to call this function implicitly using one of the wrappers that have the general naming scheme of `scale_*_tl/br_*()` (such as `scale_fill_tl_gradient()`).

Usage

```
continuous_scale_asym(aesthetics, scale_name, palette, na.value, guide, ...)
```

Arguments

`aesthetics` The names of the aesthetics that this scale works with

`scale_name` The name of the scale

`palette` A palette function that when called with a numeric vector with values between 0 and 1 returns the corresponding values in the range the scale maps to.

`na.value` Missing values will be replaced with this value.

`guide` A function used to create a guide or its name. See `guides()` for more info.

`...` other input is passed on to `ggplot2::continuous_scale()`; see `?ggplot2::continuous_scale` for complete documentation

Examples

```
library(tibble)
library(ggplot2)
tib <- tibble(
  g1 = c("A", "A", "B"),
  g2 = c("B", "C", "C"),
  val_1 = c(1, 2, 3),
  val_2 = c(-1, 0, 1)
)

tib

tib <- asymmetrise(tib, g1, g2)
ggplot(tib) +
  geom_asymmat(aes(x = g1, y = g2, fill_tl = val_1, fill_br = val_2)) +
  scale_fill_tl_gradient(low = "lightpink", high = "tomato") +
  scale_fill_br_gradient(low = "lightblue1", high = "dodgerblue") +
  labs(fill_tl = "top-left fill", fill_br = "bottom-right fill")
```

`factor_is_greater` *Determines if the level of a is greater than that of b*

Description

Determines if the level of a is greater than that of b

Usage

```
factor_is_greater(a, b)
```

Arguments

`a, b` Two same-length, same-leveled vectors of type `factor`

Value

a single boolean vector of type `logical`

Examples

```
first <- c("J", "O", "S", "H")
last <- c("C", "O", "O", "K")
first <- factor(first, LETTERS)
last <- factor(last, LETTERS)
factor_is_greater(first, last)
```

GeomAsymmat *GeomAsymmat*

Description

A 'ggproto' object for the 'ggasym' package and used by `geom_asymmat()`

Usage

```
GeomAsymmat
```

Format

An object of class `GeomAsymmat` (inherits from `GeomRect`, `Geom`, `ggproto`, `gg`) of length 7.

Warning

`GeomAsymmat` is subject to change in future versions. Use at your own risk. If dependent on `GeomAsymmat`, it is advisable to include tests with a cached version to test for equivalence.

geom_asymmat *Asymmetrically filled symmetric matrix (using 'ggplot2')*

Description

Generate an asymmetric matrix with different fill values for top-left and bottom-right triangles and along the diagonal as a `ggplot()` object

Usage

```
geom_asymmat(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

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.

`stat` The statistical transformation to use on the data for this layer, as a string.

`position` Position adjustment, either as a string, or the result of a call to a position adjustment function.

`...` Other arguments passed on to `layer()`. These are often aesthetics, used to set an aesthetic to a fixed value, like `colour = "red"` or `size = 3`. They may also be parameters to the paired geom/stat.

`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 should not inherit behaviour from the default plot specification, e.g. `borders()`.

Examples

```
library(tibble)
library(ggplot2)
suppressMessages(library(dplyr))
tib <- tibble(
  g1 = c("A", "A", "B"),
  g2 = c("B", "C", "C"),
  val_1 = c(1, 2, 3),
  val_2 = c(-1, 0, 1)
)

tib

tib <- asymmetrise(tib, g1, g2)
tib$val_3 <- NA
tib$val_3[tib$g1 == tib$g2] <- c(1, 2, 3)
ggplot(tib, aes(x = g1, y = g2)) +
  geom_asymmat(aes(fill_tl = val_1, fill_br = val_2, fill_diag = val_3)) +
  scale_fill_br_gradient(low = "lightblue1", high = "dodgerblue") +
  scale_fill_tl_gradient(low = "lightpink", high = "tomato") +
  scale_fill_diag_gradient(low = "aquamarine", high = "forestgreen") +
  labs(fill_tl = "top-left fill", fill_br = "bottom-right fill")
```

get_other_combs *Get all combinations of values between two vectors*

Description

Get all combinations of the values in vectors x and y that are not already there.

Usage

```
get_other_combs(x, y)
```

Arguments

x, y	two vectors

Value

data.frame of other possible combinations stored in Var1 and Var2 for x and y, respectively

Examples

```
get_other_combs(LETTERS[1:2], LETTERS[1:2])
```

ggasym *ggasym: Asymmetric Matrix Plotting in ggplot*

Description

This package plots a symmetric matrix with two different fill aesthetics for the top-left and bottom-right triangles and a third along the diagonal. It operates within the Grammar of Graphics paradigm implemented in 'ggplot2'.

is_grouped *Is a data table grouped?*

Description

Determines if the input data frame or tibble is grouped (using `dplyr::group_by()`

Usage

```
is_grouped(data)
```

Arguments

`data`	input `data.frame` or `tibble`

Value

boolean

Examples

```
df <- data.frame(x = c(1:5), g = c(1, 1, 2, 2, 2))
is_grouped(df)

is_grouped(dplyr::group_by(df, g))
```

make_fill_df *Make a data frame of all a single value*

Description

Makes a data frame with the same columns of `df` and `n_rows` number of rows and all values `fill_val`

Usage

```
make_fill_df(df, n_rows = 1, fill_val = NA)
```

Arguments

`df`	a data.frame (or tibble) object
`n_rows`	number of rows for the final data frame
`fill_val`	value to fill all cells of the data frame

Value

a data frame (or tibble) with the desired number of rows filled with `fill_val`

Examples

```
df <- data.frame(
  col_a = c("A", "B"),
  col_b = c("C", "D")
)
df

make_fill_df(df, 5)
```

---

organize_levels *Decides on the levels of factors x and y*

Description

Organizes the levels to use for the two inputs. This is useful for when one wants to merge two vectors that are factors. Ideally, they have the same levels, in which case those are returned. If they have overlapping levels, then the levels are merged and sorted (using `sort()`). Otherwise, the levels are dropped (returning `NULL`)

Usage

```
organize_levels(x, y, ...)
```

Arguments

`x, y`	Two factor vectors
`...`	passed to `sort`; see `?sort` for options

Value

vector of levels or `NULL` for no levels

Examples

```
set.seed(0)
a <- factor(sample(LETTERS, 5), levels = LETTERS)
b <- factor(sample(LETTERS, 5), levels = LETTERS)

a

b

organize_levels(a, b)
```

prepare_data *Prepares the input data into asymmetrise_stats*

Description

Tries to make the data ready for use in the `asymmetrise_stats()` function using `broom::tidy()`

Usage

```
prepare_data(df)
```

Arguments

`df`	input data of either a `tibble`, `data.frame`, or results from a statistical test

Value

a `tibble` data table

Warning

If you repeatedly get errors, try preparing the data before-hand using `broom::tidy(df)`

Examples

```
a <- rnorm(10, mean = 1, sd = 1)
b <- rnorm(10, mean = 1.5, sd = 1)
prepare_data(t.test(a, b))
```

scale_gradient *Gradient colour scales geom_asymmat*

Description

This dictates a gradient colour scheme for the top-left (`tl`), bottom_right (`br`), or diagonal (`diag`) of a `geom_asymmat()` geom. `scale_*_tl/br_gradient()` creates a two colour gradient (low-high), `scale_*_tl/br_gradient2()` creates a diverging colour gradient (low-mid-high), `scale_*_tl/br_gradientn()` creates a n-colour gradient.

Usage

```
scale_fill_tl_gradient(
  ...,
  low = "#132B43",
  high = "#56B1F7",
  space = "Lab",
  na.value = "grey50",
  guide = "colourbar",
  aesthetics = "fill_tl"
)

scale_fill_br_gradient(
  ...,
  low = "#132B43",
  high = "#56B1F7",
  space = "Lab",
  na.value = "grey50",
  guide = "colourbar",
  aesthetics = "fill_br"
)

scale_fill_diag_gradient(
  ...,
  low = "#132B43",
  high = "#56B1F7",
  space = "Lab",
  na.value = "grey50",
  guide = "colourbar",
  aesthetics = "fill_diag"
)

scale_fill_tl_gradient2(
  ...,
  low = scales::muted("red"),
  mid = "white",
  high = scales::muted("blue"),
  midpoint = 0,
  space = "Lab",
  na.value = "grey50",
  guide = "colourbar",
  aesthetics = "fill_tl"
)

scale_fill_br_gradient2(
  ...,
  low = scales::muted("red"),
  mid = "white",
  high = scales::muted("blue"),
  midpoint = 0,
  space = "Lab",
  na.value = "grey50",
  guide = "colourbar",
  aesthetics = "fill_br"
)

scale_fill_diag_gradient2(
  ...,
  low = scales::muted("red"),
  mid = "white",
  high = scales::muted("blue"),
  midpoint = 0,
  space = "Lab",
  na.value = "grey50",
  guide = "colourbar",
  aesthetics = "fill_diag"
)

scale_fill_tl_gradientn(
  ...,
  colours,
  values = NULL,
  space = "Lab",
  na.value = "grey50",
  guide = "colourbar",
  aesthetics = "fill_tl",
  colors
)

scale_fill_br_gradientn(
  ...,
  colours,
  values = NULL,
  space = "Lab",
  na.value = "grey50",
  guide = "colourbar",
  aesthetics = "fill_br",
  colors
)

scale_fill_diag_gradientn(
  ...,
  colours,
  values = NULL,
  space = "Lab",
  na.value = "grey50",
  guide = "colourbar",
  aesthetics = "fill_diag",
  colors
)

scale_fill_tl_distiller(
  ...,
  type = "seq",
  palette = 1,
  direction = -1,
  values = NULL,
  space = "Lab",
  na.value = "grey50",
  guide = "colourbar",
  aesthetics = "fill_tl"
)

scale_fill_br_distiller(
  ...,
  type = "seq",
  palette = 1,
  direction = -1,
  values = NULL,
  space = "Lab",
  na.value = "grey50",
  guide = "colourbar",
  aesthetics = "fill_br"
)

scale_fill_diag_distiller(
  ...,
  type = "seq",
  palette = 1,
  direction = -1,
  values = NULL,
  space = "Lab",
  na.value = "grey50",
  guide = "colourbar",
  aesthetics = "fill_diag"
)
```

Arguments

`...` arguments passed on to `continuous_scale_asym()`

`low, high` the colors to represent low and high values

`space` colour space in which to calculate gradient. Must be "Lab" - other values are deprecated.

`na.value` color of missing (`NA`) values `guide` Type of legend. Use "colourbar" for continuous colour bar, or "legend" for discrete colour legend.

`aesthetics` Character string or vector of character strings listing the name(s) of the aesthetic(s) that this scale works with. For now, leave the default alone, though I plan to reinstate the standard 'ggplot2' system here, eventually.

`mid` color for mid point (see `?scales::div_gradient_pal` for more documentation of how colors are calculated)

`midpoint` The midpoint (in data value) of the diverging scale. Defaults to 0.

`colours, colors`
Vector of colours to use for n-colour gradient.

`values` if colours should not be evenly positioned along the gradient this vector gives the position (between 0 and 1) for each colour in the colours vector. See `rescale()` for a convenience function to map an arbitrary range to between 0 and 1.

`type` One of `"seq"` (sequential), `"div"` (diverging) or `"qual"` (qualitative)

`palette` If a string, will use that named palette. If a number, will index into the list of palettes of appropriate type

`direction` Sets the order of colours in the scale. If 1, the default, colours are as output by `RColorBrewer::brewer.pal()`. If -1, the order of colours is reversed.

Examples

```
library(tibble)
library(ggplot2)
set.seed(0)

tib <- tibble(
  g1 = c("A", "A", "A", "B", "B", "C", "A", "B", "C", "D"),
  g2 = c("B", "C", "D", "C", "D", "D", "A", "B", "C", "D"),
  val_1 = c(1:10),
  val_2 = sample(-10:10, 10),
  val_3 = c(rep(NA, 6), 1, 2, 3, 4)
)
tib <- asymmetrise(tib, g1, g2)
g <- ggplot(tib, aes(x = g1, y = g2)) +
  geom_asymmat(aes(fill_tl = val_1, fill_br = val_2, fill_diag = val_3))

g + scale_fill_tl_gradient(low = "lightpink", high = "tomato") +
  scale_fill_br_gradient(low = "lightblue", high = "dodgerblue") +
  scale_fill_diag_gradient(low = "yellow", high = "orange3")

g + scale_fill_tl_gradient2(
  low = "dodgerblue",
  mid = "white", midpoint = 5,
  high = "tomato"
) +
  scale_fill_br_gradient2(
    low = "seagreen4",
    mid = "white", midpoint = 0,
    high = "orange"
  ) +
  scale_fill_diag_gradient2(
    low = "magenta",
    mid = "cornflowerblue", midpoint = 2.5,
    high = "chartreuse"
  )

g + scale_fill_tl_gradientn(colours = terrain.colors(200)) +
  scale_fill_br_gradientn(colours = heat.colors(200)) +
  scale_fill_diag_gradientn(colours = rainbow(200))

g + scale_fill_tl_distiller(type = "seq", palette = "Greens") +
  scale_fill_br_distiller(type = "div", palette = "PuOr") +
  scale_fill_diag_distiller(type = "seq", palette = "Blues")
```

swap_cols *Swap columns in a data frame*

Description

Swap columns .x and .y in df.

Usage

```
swap_cols(df, .x, .y)
```

Arguments

df	a data.frame (or tibble) object
.x, .y	column names to switch

Value

a data.frame (or tibble) object with .x and .y swapped

Examples

```
df <- data.frame(
  a = c("A", "B"),
  b = c("C", "D"),
  untouched = c(1, 2)
)
df

swap_cols(df, a, b)
```

which_level *Determine the level of a value in a vector of type* `factor`

Description

Determine the level of a value in a vector of type `factor`

Usage

```
which_level(x)
```

Arguments

`x`	vectors of type `factor`

Value

a vector holding the corresponding level of the input factors

Examples

```
first <- factor(c("J", "O", "S", "H"), LETTERS)
which_level(first)
```

Index