Package 'GGally'

March 7, 2021

Version 2.1.1

License GPL (>= 2.0)

Title Extension to 'ggplot2'

Type Package

LazyLoad yes

LazyData true

URL https://ggobi.github.io/ggally/, https://github.com/ggobi/ggally

BugReports https://github.com/ggobi/ggally/issues

Description The R package 'ggplot2' is a plotting system based on the grammar of graphics. 'GGally' extends 'ggplot2' by adding several functions to reduce the complexity of combining geometric objects with transformed data. Some of these functions include a pairwise plot matrix, a two group pairwise plot matrix, a parallel coordinates plot, a survival plot, and several functions to plot networks.

Depends R (>= 3.1), ggplot2 (>= 3.3.0)

Imports dplyr (>= 1.0.0), forcats, grDevices, grid, gtable (>= 0.2.0), lifecycle, plyr (>= 1.8.3), progress, RColorBrewer, reshape (>= 0.8.5), rlang, scales (>= 1.1.0), tidyr, utils

Suggests broom (>= 0.7.0), broom.helpers (>= 1.1.0), chemometrics, geosphere (>= 1.5-1), ggforce, Hmisc,

igraph (>= 1.0.1),
intergraph (>= 2.0-2),
labelled,
maps (>= 3.1.0),
mapproj,
nnet,
network (>= 1.12.0),
scagnostics,
sna (>= 2.3-2),
survival,
rmarkdown,
roxygen2,
testthat,
crosstalk,
knitr,
spelling,
emmeans

Roxygen list(markdown = TRUE)

RoxygenNote 7.1.1

SystemRequirements openssl

Encoding UTF-8

Language en-US

RdMacros lifecycle

R topics documented:

`+.gg` *Modify a* `ggmatrix` *object by adding an* **ggplot2** *object to all plots*

Description

This operator allows you to add **ggplot2** objects to a `ggmatrix` object.

Usage

```
## S3 method for class 'gg'
e1 + e2

add_to_ggmatrix(e1, e2, location = NULL, rows = NULL, cols = NULL)
```

Arguments

`e1`	An object of class `ggnostic` or `ggplot`
`e2`	A component to add to `e1`
`location`	`"all"`, `TRUE` All row and col combinations `"none"` No row and column combinations `"upper"` Locations where the column value is higher than the row value `"lower"` Locations where the row value is higher than the column value `"diag"` Locations where the column value is equal to the row value `matrix` **or** `data.frame` `matrix` values will be converted into `data.frames`.

- A `data.frame` with the exact column names `c("row","col")`
- A `data.frame` with the number of rows and columns matching the plot matrix object provided. Each cell will be tested for a "truthy" value to determine if the location should be kept.

`rows` numeric vector of the rows to be used. Will be used with `cols` if `location` is `NULL`

`cols` numeric vector of the cols to be used. Will be used with `rows` if `location` is `NULL`

Details

If the first object is an object of class `ggmatrix`, you can add the following types of objects, and it will return a modified **ggplot2** object.

- `theme`: update plot theme
- `scale`: replace current scale
- `coord`: override current coordinate system

The + operator completely replaces elements with elements from e2.

`add_to_ggmatrix` gives you more control to modify only some subplots. This function may be replaced and/or removed in the future. **[Experimental]**

See Also

ggplot2::+.gg and `ggplot2::theme()`

`ggmatrix_location`

Examples

```
# small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive
data(tips, package = "reshape")

pm <- ggpairs(tips[, 2:4], ggplot2::aes(color = sex))
## change to black and white theme
pm + ggplot2::theme_bw()
## change to linedraw theme
p_(pm + ggplot2::theme_linedraw())
## change to custom theme
p_(pm + ggplot2::theme(panel.background = ggplot2::element_rect(fill = "lightblue")))
## add a list of information
extra <- list(ggplot2::theme_bw(), ggplot2::labs(caption = "My caption!"))
p_(pm + extra)
## modify scale
p_(pm + scale_fill_brewer(type = "qual"))
## only first row
p_(add_to_ggmatrix(pm, scale_fill_brewer(type = "qual"), rows = 1:2))
## only second col
p_(add_to_ggmatrix(pm, scale_fill_brewer(type = "qual"), cols = 2:3))
## only to upper triangle of plot matrix
p_(add_to_ggmatrix(
  pm,
  scale_fill_brewer(type = "qual"),
  location = "upper"
))
```

add_ref_boxes *Add reference boxes around each cell of the glyphmap.*

Description

Add reference boxes around each cell of the glyphmap.

Usage

```
add_ref_boxes(
  data,
  var_fill = NULL,
  color = "white",
  size = 0.5,
  fill = NA,
  ...
)
```

Arguments

data	A glyphmap structure.
var_fill	Variable name to use to set the fill color
color	Set the color to draw in, default is "white"
size	Set the line size, default is 0.5
fill	fill value used if var_fill is NULL
...	other arguments passed onto ggplot2::geom_rect()

add_ref_lines *Add reference lines for each cell of the glyphmap.*

Description

Add reference lines for each cell of the glyphmap.

Usage

```
add_ref_lines(data, color = "white", size = 1.5, ...)
```

Arguments

data	A glyphmap structure.
color	Set the color to draw in, default is "white"
size	Set the line size, default is 1.5
...	other arguments passed onto ggplot2::geom_line()

`australia_PISA2012` *Programme for International Student Assessment (PISA) 2012 Data for Australia*

Description

About PISA

Usage

```
data(australia_PISA2012)
```

Format

A data frame with 8247 rows and 32 variables

Details

The Programme for International Student Assessment (PISA) is a triennial international survey which aims to evaluate education systems worldwide by testing the skills and knowledge of 15-year-old students. To date, students representing more than 70 economies have participated in the assessment.

While 65 economies took part in the 2012 study, this data set only contains information from the country of Australia.

- gender : Factor w/ 2 levels "female","male": 1 1 2 2 2 1 1 1 2 1 ...
- age : Factor w/ 4 levels "4","5","6","7": 2 2 2 4 3 1 2 2 2 2 ...
- homework : num 5 5 9 3 2 3 4 3 5 1 ...
- desk : num 1 0 1 1 1 1 1 1 1 1 ...
- room : num 1 1 1 1 1 1 1 1 1 1 ...
- study : num 1 1 1 1 1 1 1 1 1 1 ...
- computer : num 1 1 1 1 1 1 1 1 1 1 ...
- software : num 1 1 1 1 1 1 1 1 1 1 ...
- internet : num 1 1 1 1 1 1 1 1 1 1 ...
- literature : num 0 0 1 0 1 1 1 1 1 0 ...
- poetry : num 0 0 1 0 1 1 0 1 1 1 ...
- art : num 1 0 1 0 1 1 0 1 1 1 ...
- textbook : num 1 1 1 1 1 0 1 1 1 1 ...
- dictionary : num 1 1 1 1 1 1 1 1 1 1 ...
- dishwasher : num 1 1 1 1 0 1 1 1 1 1 ...
- PV1MATH : num 562 565 602 520 613 ...
- PV2MATH : num 569 557 594 507 567 ...
- PV3MATH : num 555 553 552 501 585 ...
- PV4MATH : num 579 538 526 521 596 ...
- PV5MATH : num 548 573 619 547 603 ...

- PV1READ : num 582 617 650 554 605 ...
- PV2READ : num 571 572 608 560 557 ...
- PV3READ : num 602 560 594 517 627 ...
- PV4READ : num 572 564 575 564 597 ...
- PV5READ : num 585 565 620 572 598 ...
- PV1SCIE : num 583 627 668 574 639 ...
- PV2SCIE : num 579 600 665 612 635 ...
- PV3SCIE : num 593 574 620 571 666 ...
- PV4SCIE : num 567 582 592 598 700 ...
- PV5SCIE : num 587 625 656 662 670 ...
- SENWGT_STU : num 0.133 0.133 0.141 0.141 0.141 ...
- possessions: num 10 8 12 9 11 11 10 12 12 11 ...

Source

http://www.oecd.org/pisa/pisaproducts/database-cbapisa2012.htm

---

`brew_colors` *RColorBrewer Set1 colors*

Description

RColorBrewer Set1 colors

Usage

```
brew_colors(col)
```

Arguments

`col`	standard color name used to retrieve hex color value

`broomify` *Broomify a model*

Description

broom::augment a model and add broom::glance and broom::tidy output as attributes. X and Y variables are also added.

Usage

```
broomify(model, lmStars = TRUE)
```

Arguments

`model`	model to be sent to `broom::augment()`, `broom::glance()`, and `broom::tidy()`
`lmStars`	boolean that determines if stars are added to labels

Value

broom::augmented data frame with the broom::glance data.frame and broom::tidy data.frame as 'broom_glance' and 'broom_tidy' attributes respectively. `var_x` and `var_y` variables are also added as attributes

Examples

```
data(mtcars)
model <- stats::lm(mpg ~ wt + qsec + am, data = mtcars)
broomified_model <- broomify(model)
str(broomified_model)
```

---

eval_data_col *Evaluate data column*

Description

Evaluate data column

Usage

```
eval_data_col(data, aes_col)
```

Arguments

`data`	data set to evaluate the data with
`aes_col`	Single value from an `ggplot2::aes(...)` object

Value

Aes mapping with the x and y values switched

Examples

```
mapping <- ggplot2::aes(Petal.Length)
eval_data_col(iris, mapping$x)
```

---

flea *Historical data used for classification examples.*

Description

This data contains physical measurements on three species of flea beetles.

Usage

```
data(flea)
```

Format

A data frame with 74 rows and 7 variables

Details

- species Ch. concinna, Ch. heptapotamica, Ch. heikertingeri
- tars1 width of the first joint of the first tarsus in microns
- tars2 width of the second joint of the first tarsus in microns
- head the maximal width of the head between the external edges of the eyes in 0.01 mm
- aede1 the maximal width of the aedeagus in the fore-part in microns
- aede2 the front angle of the aedeagus (1 unit = 7.5 degrees)
- aede3 the aedeagus width from the side in microns

References

Lubischew, A. A. (1962), On the Use of Discriminant Functions in Taxonomy, Biometrics 18:455-477.

`fn_switch` *Function switch*

Description

Function that allows you to call different functions based upon an aesthetic variable value.

Usage

```
fn_switch(types, mapping_val = "y")
```

Arguments

`types`	list of functions that follow the `ggmatrix` function standard: `function(data,mapping,...){ #make ggplot2 object }`. One key should be a 'default' key for a default switch case.
`mapping_val`	mapping value to switch on. Defaults to the 'y' variable of the aesthetics list.

Examples

```
ggnostic_continuous_fn <- fn_switch(list(
  default = ggally_points,
  .fitted = ggally_points,
  .se.fit = ggally_nostic_se_fit,
  .resid = ggally_nostic_resid,
  .hat = ggally_nostic_hat,
  .sigma = ggally_nostic_sigma,
  .cooksd = ggally_nostic_cooksd,
  .std.resid = ggally_nostic_std_resid
))

ggnostic_combo_fn <- fn_switch(list(
  default = ggally_box_no_facet,
  fitted = ggally_box_no_facet,
  .se.fit = ggally_nostic_se_fit,
  .resid = ggally_nostic_resid,
```

```
  .hat = ggally_nostic_hat,
  .sigma = ggally_nostic_sigma,
  .cooksd = ggally_nostic_cooksd,
  .std.resid = ggally_nostic_std_resid
))
```

`geom_stripped_rows` *Alternating Background Colour*

Description

Add alternating background color along the y-axis. The geom takes default aesthetics `odd` and `even` that receive color codes.

Usage

```
geom_stripped_rows(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  show.legend = NA,
  inherit.aes = TRUE,
  xfrom = -Inf,
  xto = Inf,
  width = 1,
  nudge_y = 0
)

geom_stripped_cols(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  show.legend = NA,
  inherit.aes = TRUE,
  yfrom = -Inf,
  yto = Inf,
  width = 1,
  nudge_x = 0
)
```

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	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.
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().
xfrom, xto	limitation of the strips along the x-axis
width	width of the strips
yfrom, yto	limitation of the strips along the y-axis
nudge_x, nudge_y	horizontal or vertical adjustment to nudge strips by

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

data(tips, package = "reshape")
p <- ggplot(tips) +
  aes(x = time, y = day) +
  geom_count() +
  theme_light()

p_(p)
p_(p + geom_stripped_rows())
p_(p + geom_stripped_cols())
p_(p + geom_stripped_rows() + geom_stripped_cols())

p <- ggplot(tips) +
  aes(x = total_bill, y = day) +
  geom_count() +
  theme_light()
p
p_(p + geom_stripped_rows())
p_(p + geom_stripped_rows() + scale_y_discrete(expand = expansion(0, 0.5)))
p_(p + geom_stripped_rows(xfrom = 10, xto = 35))
p_(p + geom_stripped_rows(odd = "blue", even = "yellow"))
```

```
p_(p + geom_stripped_rows(odd = "blue", even = "yellow", alpha = .1))
p_(p + geom_stripped_rows(odd = "#00FF0022", even = "#FF000022"))

p_(p + geom_stripped_cols())
p_(p + geom_stripped_cols(width = 10))
p_(p + geom_stripped_cols(width = 10, nudge_x = 5))
```

---

getPlot *Subset a* ggmatrix *object*

Description

Retrieves the ggplot object at the desired location.

Usage

```
getPlot(pm, i, j)

## S3 method for class 'ggmatrix'
pm[i, j, ...]
```

Arguments

pm	ggmatrix object to select from
i	row from the top
j	column from the left
...	ignored

Author(s)

Barret Schloerke

See Also

putPlot

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

data(tips, package = "reshape")
plotMatrix2 <- ggpairs(tips[, 3:2], upper = list(combo = "denstrip"))
p_(plotMatrix2[1, 2])
```

ggally_autopoint *Scatterplot for continuous and categorical variables*

Description

Make scatterplots compatible with both continuous and categorical variables using geom_autopoint from package **ggforce**.

Usage

```
ggally_autopoint(data, mapping, ...)

ggally_autopointDiag(data, mapping, ...)
```

Arguments

data	data set using
mapping	aesthetics being used
...	other arguments passed to geom_autopoint(...)

Author(s)

Joseph Larmarange

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

data(tips, package = "reshape")
p_(ggally_autopoint(tips, mapping = aes(x = tip, y = total_bill)))
p_(ggally_autopoint(tips, mapping = aes(x = tip, y = sex)))
p_(ggally_autopoint(tips, mapping = aes(x = smoker, y = sex)))
p_(ggally_autopoint(tips, mapping = aes(x = smoker, y = sex, color = day)))
p_(ggally_autopoint(tips, mapping = aes(x = smoker, y = sex), size = 8))
p_(ggally_autopoint(tips, mapping = aes(x = smoker, y = sex), alpha = .9))

p_(ggpairs(
  tips,
  mapping = aes(colour = sex),
  upper = list(discrete = "autopoint", combo = "autopoint", continuous = "autopoint"),
  diag = list(discrete = "autopointDiag", continuous = "autopointDiag")
))
```

ggally_barDiag *Bar plot*

Description

Displays a bar plot for the diagonal of a ggpairs plot matrix.

Usage

```
ggally_barDiag(data, mapping, ..., rescale = FALSE)
```

Arguments

data	data set using
mapping	aesthetics being used
...	other arguments are sent to geom_bar
rescale	boolean to decide whether or not to rescale the count output. Only applies to numeric data

Author(s)

Barret Schloerke

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

data(tips, package = "reshape")
p_(ggally_barDiag(tips, mapping = ggplot2::aes(x = day)))
p_(ggally_barDiag(tips, mapping = ggplot2::aes(x = tip), binwidth = 0.25))
```

ggally_blank *Blank plot*

Description

Draws nothing.

Usage

```
ggally_blank(...)

ggally_blankDiag(...)
```

Arguments

...	other arguments ignored

Details

Makes a "blank" ggplot object that will only draw white space

Author(s)

Barret Schloerke

See Also

ggplot2::element_blank()

---

ggally_box *Box plot*

Description

Make a box plot with a given data set. ggally_box_no_facet will be a single panel plot, while ggally_box will be a faceted plot

Usage

```
ggally_box(data, mapping, ...)

ggally_box_no_facet(data, mapping, ...)
```

Arguments

data	data set using
mapping	aesthetics being used
...	other arguments being supplied to geom_boxplot

Author(s)

Barret Schloerke

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

data(tips, package = "reshape")
p_(ggally_box(tips, mapping = ggplot2::aes(x = total_bill, y = sex)))
p_(ggally_box(tips, mapping = ggplot2::aes_string(x = "total_bill", y = "sex")))
p_(ggally_box(
  tips,
  mapping        = ggplot2::aes_string(y = "total_bill", x = "sex", color = "sex"),
  outlier.colour = "red",
  outlier.shape  = 13,
  outlier.size   = 8
))
```

ggally_colbar *Column and row bar plots*

Description

Plot column or row percentage using bar plots.

Usage

```
ggally_colbar(
  data,
  mapping,
  label_format = scales::label_percent(accuracy = 0.1),
  ...,
  remove_background = FALSE,
  remove_percentage_axis = FALSE,
  reverse_fill_levels = FALSE,
  geom_bar_args = NULL
)

ggally_rowbar(
  data,
  mapping,
  label_format = scales::label_percent(accuracy = 0.1),
  ...,
  remove_background = FALSE,
  remove_percentage_axis = FALSE,
  reverse_fill_levels = TRUE,
  geom_bar_args = NULL
)
```

Arguments

data	data set using
mapping	aesthetics being used
label_format	formatter function for displaying proportions, not taken into account if a label aesthetic is provided in mapping
...	other arguments passed to geom_text(...)
remove_background	should the panel.background be removed?
remove_percentage_axis	should percentage axis be removed? Removes the y-axis for ggally_colbar() and x-axis for ggally_rowbar()
reverse_fill_levels	should the levels of the fill variable be reversed?
geom_bar_args	other arguments passed to geom_bar(...)

Author(s)

Joseph Larmarange

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

data(tips, package = "reshape")
p_(ggally_colbar(tips, mapping = aes(x = smoker, y = sex)))
p_(ggally_rowbar(tips, mapping = aes(x = smoker, y = sex)))

# change labels' size
p_(ggally_colbar(tips, mapping = aes(x = smoker, y = sex), size = 8))

# change labels' colour and use bold
p_(ggally_colbar(tips, mapping = aes(x = smoker, y = sex),
              colour = "white", fontface = "bold"))

# display number of observations instead of proportions
p_(ggally_colbar(tips, mapping = aes(x = smoker, y = sex, label = after_stat(count))))

# custom bar width
p_(ggally_colbar(tips, mapping = aes(x = smoker, y = sex), geom_bar_args = list(width = .5)))

# change format of labels
p_(ggally_colbar(tips, mapping = aes(x = smoker, y = sex),
              label_format = scales::label_percent(accuracy = .01, decimal.mark = ",")))

p_(ggduo(
  data = as.data.frame(Titanic),
  mapping = aes(weight = Freq),
  columnsX = "Survived",
  columnsY = c("Sex", "Class", "Age"),
  types = list(discrete = "rowbar"),
  legend = 1
))
```

---

ggally_cor *Correlation value plot*

Description

Estimate correlation from the given data. If a color variable is supplied, the correlation will also be calculated per group.

Usage

```
ggally_cor(
  data,
  mapping,
  ...,
  stars = TRUE,
  method = "pearson",
  use = "complete.obs",
  display_grid = FALSE,
  digits = 3,
```

```
  title_args = list(...),
  group_args = list(...),
  justify_labels = "right",
  align_percent = 0.5,
  title = "Corr",
  alignPercent = warning("deprecated. Use `align_percent`"),
  displayGrid = warning("deprecated. Use `display_grid`")
)
```

Arguments

`data`	data set using
`mapping`	aesthetics being used
`...`	other arguments being supplied to `geom_text()` for the title and groups
`stars`	logical value which determines if the significance stars should be displayed. Given the `cor.test` p-values, display `"***"` if the p-value is < 0.001 `"**"` if the p-value is < 0.01 `"*"` if the p-value is < 0.05 `"."` if the p-value is < 0.10 `""` otherwise
`method`	`method` supplied to cor function
`use`	`use` supplied to `cor` function
`display_grid`	if `TRUE`, display aligned panel grid lines. If `FALSE` (default), display a thin panel border.
`digits`	number of digits to be displayed after the decimal point. See `formatC` for how numbers are calculated.
`title_args`	arguments being supplied to the title's `geom_text()`
`group_args`	arguments being supplied to the split-by-color group's `geom_text()`
`justify_labels`	`justify` argument supplied when `format`ting the labels
`align_percent`	relative align position of the text. When `justify_labels = 0.5`, this should not be needed to be set.
`title`	title text to be displayed
`alignPercent, displayGrid`	deprecated. Please use their snake-case counterparts.

Author(s)

Barret Schloerke

See Also

`ggally_statistic`, `ggally_cor_v1_5`

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

data(tips, package = "reshape")
p_(ggally_cor(tips, mapping = ggplot2::aes_string(x = "total_bill", y = "tip")))
# display with grid
p_(ggally_cor(
  tips,
  mapping = ggplot2::aes_string(x = "total_bill", y = "tip"),
  display_grid = TRUE
))
# change text attributes
p_(ggally_cor(
  tips,
  mapping = ggplot2::aes(x = total_bill, y = tip),
  size = 15,
  colour = I("red"),
  title = "Correlation"
))
# split by a variable
p_(ggally_cor(
  tips,
  mapping = ggplot2::aes_string(x = "total_bill", y = "tip", color = "sex"),
  size = 5
))
```

---

ggally_cor_v1_5 *Correlation value plot*

Description

(Deprecated. See ggally_cor.)

Usage

```
ggally_cor_v1_5(
  data,
  mapping,
  alignPercent = 0.6,
  method = "pearson",
  use = "complete.obs",
  corAlignPercent = NULL,
  corMethod = NULL,
  corUse = NULL,
  displayGrid = TRUE,
  ...
)
```

Arguments

data	data set using

mapping	aesthetics being used
alignPercent	right align position of numbers. Default is 60 percent across the horizontal
method	method supplied to cor function
use	use supplied to cor function
corAlignPercent	deprecated. Use parameter alignPercent
corMethod	deprecated. Use parameter method
corUse	deprecated. Use parameter use
displayGrid	if TRUE, display aligned panel gridlines
...	other arguments being supplied to geom_text

Details

Estimate correlation from the given data.

Author(s)

Barret Schloerke

See Also

ggally_cor

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

data(tips, package = "reshape")
p_(ggally_cor_v1_5(tips, mapping = ggplot2::aes_string(x = "total_bill", y = "tip")))

# display with no grid
p_(ggally_cor_v1_5(
  tips,
  mapping = ggplot2::aes_string(x = "total_bill", y = "tip"),
  displayGrid = FALSE
))

# change text attributes
p_(ggally_cor_v1_5(
  tips,
  mapping = ggplot2::aes(x = total_bill, y = tip),
  size = 15,
  colour = I("red")
))

# split by a variable
p_(ggally_cor_v1_5(
  tips,
  mapping = ggplot2::aes_string(x = "total_bill", y = "tip", color = "sex"),
  size = 5
))
```

`ggally_count` *Display counts of observations*

Description

Plot the number of observations by using rectangles with proportional areas.

Usage

```
ggally_count(data, mapping, ...)

ggally_countDiag(data, mapping, ...)
```

Arguments

`data`	data set using
`mapping`	aesthetics being used
`...`	other arguments passed to `geom_tile(...)`

Details

You can adjust the size of rectangles with the `x.width` argument.

Author(s)

Joseph Larmarange

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

data(tips, package = "reshape")
p_(ggally_count(tips, mapping = ggplot2::aes(x = smoker, y = sex)))
p_(ggally_count(tips, mapping = ggplot2::aes(x = smoker, y = sex, fill = day)))

p_(ggally_count(
  as.data.frame(Titanic),
  mapping = ggplot2::aes(x = Class, y = Survived, weight = Freq)
))
p_(ggally_count(
  as.data.frame(Titanic),
  mapping = ggplot2::aes(x = Class, y = Survived, weight = Freq),
  x.width = 0.5
))
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

p_(ggally_countDiag(tips, mapping = ggplot2::aes(x = smoker)))
p_(ggally_countDiag(tips, mapping = ggplot2::aes(x = smoker, fill = sex)))
```

ggally_cross *Plots the number of observations*

Description

Plot the number of observations by using square points with proportional areas. Could be filled according to chi-squared statistics computed by `stat_cross()`. Labels could also be added (see examples).

Usage

```
ggally_cross(data, mapping, ..., scale_max_size = 20, geom_text_args = NULL)
```

Arguments

`data`	data set using
`mapping`	aesthetics being used
`...`	other arguments passed to `ggplot2::geom_point()`
`scale_max_size`	`max_size` argument supplied to `ggplot2::scale_size_area()`
`geom_text_args`	other arguments passed to `ggplot2::geom_text()`

Author(s)

Joseph Larmarange

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

data(tips, package = "reshape")
p_(ggally_cross(tips, mapping = aes(x = smoker, y = sex)))
p_(ggally_cross(tips, mapping = aes(x = day, y = time)))

# Custom max size
p_(ggally_cross(tips, mapping = aes(x = smoker, y = sex)) +
  scale_size_area(max_size = 40))

# Custom fill
p_(ggally_cross(tips, mapping = aes(x = smoker, y = sex), fill = "red"))

# Custom shape
p_(ggally_cross(tips, mapping = aes(x = smoker, y = sex), shape = 21))

# Fill squares according to standardized residuals
d <- as.data.frame(Titanic)
p_(ggally_cross(
  d,
  mapping = aes(x = Class, y = Survived, weight = Freq, fill = after_stat(std.resid))
) +
  scale_fill_steps2(breaks = c(-3, -2, 2, 3), show.limits = TRUE))
```

```
# Add labels
p_(ggally_cross(
  tips,
  mapping = aes(
    x = smoker, y = sex, colour = smoker,
    label = scales::percent(after_stat(prop))
  )
))

# Customize labels' appearance and same size for all squares
p_(ggally_cross(
  tips,
  mapping = aes(
    x = smoker, y = sex,
    size = NULL, # do not map size to a variable
    label = scales::percent(after_stat(prop))
  ),
  size = 40, # fix value for points size
  fill = "darkblue",
  geom_text_args = list(colour = "white", fontface = "bold", size = 6)
))
```

`ggally_crosstable` *Display a cross-tabulated table*

Description

`ggally_crosstable` is a variation of `ggally_table` with few modifications: (i) table cells are drawn; (ii) x and y axis are not expanded (and therefore are not aligned with other `ggally_*` plots); (iii) content and fill of cells can be easily controlled with dedicated arguments.

Usage

```
ggally_crosstable(
  data,
  mapping,
  cells = c("observed", "prop", "row.prop", "col.prop", "expected", "resid",
    "std.resid"),
  fill = c("none", "std.resid", "resid"),
  ...,
  geom_tile_args = list(colour = "grey50")
)
```

Arguments

`data`	data set using
`mapping`	aesthetics being used
`cells`	Which statistic should be displayed in table cells?
`fill`	Which statistic should be used for filling table cells?
`...`	other arguments passed to `geom_text(...)`
`geom_tile_args`	other arguments passed to `geom_tile(...)`

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

data(tips, package = "reshape")

# differences with ggally_table()
p_(ggally_table(tips, mapping = aes(x = day, y = time)))
p_(ggally_crosstable(tips, mapping = aes(x = day, y = time)))

# display column proportions
p_(ggally_crosstable(tips, mapping = aes(x = day, y = sex), cells = "col.prop"))

# display row proportions
p_(ggally_crosstable(tips, mapping = aes(x = day, y = sex), cells = "row.prop"))

# change size of text
p_(ggally_crosstable(tips, mapping = aes(x = day, y = sex), size = 8))

# fill cells with standardized residuals
p_(ggally_crosstable(tips, mapping = aes(x = day, y = sex), fill = "std.resid"))

# change scale for fill
p_(ggally_crosstable(tips, mapping = aes(x = day, y = sex), fill = "std.resid") +
  scale_fill_steps2(breaks = c(-2, 0, 2), show.limits = TRUE))
```

---

ggally_density *Bivariate density plot*

Description

Make a 2D density plot from a given data.

Usage

```
ggally_density(data, mapping, ...)
```

Arguments

data	data set using
mapping	aesthetics being used
...	parameters sent to either stat_density2d or geom_density2d

Details

The aesthetic "fill" determines whether or not `stat_density2d` (filled) or `geom_density2d` (lines) is used.

Author(s)

Barret Schloerke

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

data(tips, package = "reshape")
p_(ggally_density(tips, mapping = ggplot2::aes(x = total_bill, y = tip)))
p_(ggally_density(tips, mapping = ggplot2::aes_string(x = "total_bill", y = "tip")))
p_(ggally_density(
  tips,
  mapping = ggplot2::aes_string(x = "total_bill", y = "tip", fill = "..level..")
))
p_(ggally_density(
  tips,
  mapping = ggplot2::aes_string(x = "total_bill", y = "tip", fill = "..level..")
) + ggplot2::scale_fill_gradient(breaks = c(0.05, 0.1, 0.15, 0.2)))
```

---

ggally_densityDiag *Univariate density plot*

Description

Displays a density plot for the diagonal of a ggpairs plot matrix.

Usage

```
ggally_densityDiag(data, mapping, ..., rescale = FALSE)
```

Arguments

data	data set using
mapping	aesthetics being used.
...	other arguments sent to stat_density
rescale	boolean to decide whether or not to rescale the count output

Author(s)

Barret Schloerke

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

data(tips, package = "reshape")
p_(ggally_densityDiag(tips, mapping = ggplot2::aes(x = total_bill)))
p_(ggally_densityDiag(tips, mapping = ggplot2::aes(x = total_bill, color = day)))
```

ggally_denstrip *Tile plot with facets*

Description

Displays a Tile Plot as densely as possible.

Usage

```
ggally_denstrip(data, mapping, ...)
```

Arguments

data	data set using
mapping	aesthetics being used
...	other arguments being sent to stat_bin

Author(s)

Barret Schloerke

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

data(tips, package = "reshape")
p_(ggally_denstrip(tips, mapping = ggplot2::aes(x = total_bill, y = sex)))
p_(ggally_denstrip(tips, mapping = ggplot2::aes_string(x = "total_bill", y = "sex")))
p_(ggally_denstrip(
  tips,
  mapping = ggplot2::aes_string(x = "sex", y = "tip", binwidth = "0.2")
) + ggplot2::scale_fill_gradient(low = "grey80", high = "black"))
```

ggally_diagAxis *Internal axis labels for ggpairs*

Description

This function is used when `axisLabels == "internal"`.

Usage

```
ggally_diagAxis(
  data,
  mapping,
  label = mapping$x,
  labelSize = 5,
  labelXPercent = 0.5,
  labelYPercent = 0.55,
```

```
  labelHJust = 0.5,
  labelVJust = 0.5,
  gridLabelSize = 4,
  ...
)
```

Arguments

data	dataset being plotted
mapping	aesthetics being used (x is the variable the plot will be made for)
label	title to be displayed in the middle. Defaults to mapping$x
labelSize	size of variable label
labelXPercent	percent of horizontal range
labelYPercent	percent of vertical range
labelHJust	hjust supplied to label
labelVJust	vjust supplied to label
gridLabelSize	size of grid labels
...	other arguments for geom_text

Author(s)

Jason Crowley and Barret Schloerke

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

data(tips, package = "reshape")
p_(ggally_diagAxis(tips, ggplot2::aes(x=tip)))
p_(ggally_diagAxis(tips, ggplot2::aes(x=sex)))
```

---

ggally_dot *Grouped dot plot*

Description

Add jittering with the box plot. ggally_dot_no_facet will be a single panel plot, while ggally_dot will be a faceted plot

Usage

```
ggally_dot(data, mapping, ...)

ggally_dot_no_facet(data, mapping, ...)
```

Arguments

data	data set using
mapping	aesthetics being used
...	other arguments being supplied to geom_jitter

Author(s)

Barret Schloerke

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

data(tips, package = "reshape")
p_(ggally_dot(tips, mapping = ggplot2::aes(x = total_bill, y = sex)))
p_(ggally_dot(tips, mapping = ggplot2::aes_string(x = "total_bill", y = "sex")))
p_(ggally_dot(
  tips,
  mapping = ggplot2::aes_string(y = "total_bill", x = "sex", color = "sex")
))
p_(ggally_dot(
  tips,
  mapping = ggplot2::aes_string(y = "total_bill", x = "sex", color = "sex", shape = "sex")
) + ggplot2::scale_shape(solid=FALSE))
```

---

ggally_facetbar *Faceted bar plot*

Description

X variables are plotted using geom_bar and are faceted by the Y variable.

Usage

```
ggally_facetbar(data, mapping, ...)
```

Arguments

data	data set using
mapping	aesthetics being used
...	other arguments are sent to geom_bar

Author(s)

Barret Schloerke

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

data(tips, package = "reshape")
p_(ggally_facetbar(tips, ggplot2::aes(x = sex, y = smoker, fill = time)))
p_(ggally_facetbar(tips, ggplot2::aes(x = smoker, y = sex, fill = time)))
```

ggally_facetdensity *Faceted density plot*

Description

Make density plots by displaying subsets of the data in different panels.

Usage

```
ggally_facetdensity(data, mapping, ...)
```

Arguments

data	data set using
mapping	aesthetics being used
...	other arguments being sent to stat_density

Author(s)

Barret Schloerke

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

data(tips, package = "reshape")
p_(ggally_facetdensity(tips, mapping = ggplot2::aes(x = total_bill, y = sex)))
p_(ggally_facetdensity(
  tips,
  mapping = ggplot2::aes_string(y = "total_bill", x = "sex", color = "sex")
))
```

ggally_facetdensitystrip *Density or tiles plot with facets*

Description

Make tile plot or density plot as compact as possible.

Usage

```
ggally_facetdensitystrip(data, mapping, ..., den_strip = FALSE)
```

Arguments

data	data set using
mapping	aesthetics being used
...	other arguments being sent to either geom_histogram or stat_density
den_strip	boolean to decide whether or not to plot a density strip(TRUE) or a facet density(FALSE) plot.

Author(s)

Barret Schloerke

Examples

```
example(ggally_facetdensity)
example(ggally_denstrip)
```

`ggally_facethist` *Faceted histogram*

Description

Display subsetted histograms of the data in different panels.

Usage

```
ggally_facethist(data, mapping, ...)
```

Arguments

`data`	data set using
`mapping`	aesthetics being used
`...`	parameters sent to stat_bin()

Author(s)

Barret Schloerke

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

data(tips, package = "reshape")
p_(ggally_facethist(tips, mapping = ggplot2::aes(x = tip, y = sex)))
p_(ggally_facethist(tips, mapping = ggplot2::aes_string(x = "tip", y = "sex"), binwidth = 0.1))
```

ggally_na *NA plot*

Description

Draws a large NA in the middle of the plotting area. This plot is useful when all X or Y data is NA

Usage

```
ggally_na(data = NULL, mapping = NULL, size = 10, color = "grey20", ...)

ggally_naDiag(...)
```

Arguments

data	ignored
mapping	ignored
size	size of the geom_text 'NA'
color	color of the geom_text 'NA'
...	other arguments sent to geom_text

Author(s)

Barret Schloerke

ggally_nostic_cooksd ggnostic *Cook's distance*

Description

A function to display stats::cooks.distance().

Usage

```
ggally_nostic_cooksd(
  data,
  mapping,
  ...,
 linePosition = pf(0.5, length(attr(data, "var_x")), nrow(data) - length(attr(data,
    "var_x"))),
  lineColor = brew_colors("grey"),
  lineType = 2
)
```

Arguments

data, mapping, ..., lineColor, lineType	parameters supplied to ggally_nostic_line
linePosition	4 / n is the general cutoff point for Cook's Distance

Details

A line is added at F_p, n - p(0.5) to display the general cutoff point for Cook's Distance.

Reference: Michael H. Kutner, Christopher J. Nachtsheim, John Neter, and William Li. Applied linear statistical models. The McGraw-Hill / Irwin series operations and decision sciences. McGraw-Hill Irwin, 2005, p. 403

Value

ggplot2 plot object

See Also

`stats::cooks.distance()`

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

dt <- broomify(stats::lm(mpg ~ wt + qsec + am, data = mtcars))
p_(ggally_nostic_cooksd(dt, ggplot2::aes(wt, .cooksd)))
```

`ggally_nostic_hat` `ggnostic` *leverage points*

Description

A function to display stats::influence's hat information against a given explanatory variable.

Usage

```
ggally_nostic_hat(
  data,
  mapping,
  ...,
  linePosition = 2 * sum(eval_data_col(data, mapping$y))/nrow(data),
  lineColor = brew_colors("grey"),
  lineSize = 0.5,
  lineAlpha = 1,
  lineType = 2,
  avgLinePosition = sum(eval_data_col(data, mapping$y))/nrow(data),
  avgLineColor = brew_colors("grey"),
  avgLineSize = lineSize,
  avgLineAlpha = lineAlpha,
  avgLineType = 1
)
```

Arguments

`data, mapping, ...`
: supplied directly to `ggally_nostic_line`

`linePosition, lineColor, lineSize, lineAlpha, lineType`
: parameters supplied to `ggplot2::geom_line()` for the cutoff line

`avgLinePosition, avgLineColor, avgLineSize, avgLineAlpha, avgLineType`
: parameters supplied to `ggplot2::geom_line()` for the average line

Details

As stated in `stats::influence()` documentation:

hat: a vector containing the diagonal of the 'hat' matrix.

The diagonal elements of the 'hat' matrix describe the influence each response value has on the fitted value for that same observation.

A suggested "cutoff" line is added to the plot at a height of 2 * p / n and an expected line at a height of p / n. If either `linePosition` or `avgLinePosition` is `NULL`, the respective line will not be drawn.

Value

ggplot2 plot object

See Also

`stats::influence()`

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

dt <- broomify(stats::lm(mpg ~ wt + qsec + am, data = mtcars))
p_(ggally_nostic_hat(dt, ggplot2::aes(wt, .hat)))
```

---

ggally_nostic_line *ggnostic background line with geom*

Description

If a non-null `linePosition` value is given, a line will be drawn before the given `continuous_geom` or `combo_geom` is added to the plot.

Usage

```
ggally_nostic_line(
  data,
  mapping,
  ...,
  linePosition = NULL,
  lineColor = "red",
  lineSize = 0.5,
```

```
  lineAlpha = 1,
  lineType = 1,
  continuous_geom = ggplot2::geom_point,
  combo_geom = ggplot2::geom_boxplot,
  mapColorToFill = TRUE
)
```

Arguments

`data, mapping`	supplied directly to `ggplot2::ggplot()`
`...`	parameters supplied to `continuous_geom` or `combo_geom`
`linePosition, lineColor, lineSize, lineAlpha, lineType`	parameters supplied to `ggplot2::geom_line()`
`continuous_geom`	**ggplot2** geom that is executed after the line is (possibly) added and if the x data is continuous
`combo_geom`	**ggplot2** geom that is executed after the line is (possibly) added and if the x data is discrete
`mapColorToFill`	boolean to determine if combo plots should cut the color mapping to the fill mapping

Details

Functions with a color in their name have different default color behavior.

Value

ggplot2 plot object

`ggally_nostic_resid` `ggnostic` *residuals*

Description

If non-null `pVal` and `sigma` values are given, confidence interval lines will be added to the plot at the specified `pVal` percentiles of a N(0, sigma) distribution.

Usage

```
ggally_nostic_resid(
  data,
  mapping,
  ...,
  linePosition = 0,
  lineColor = brew_colors("grey"),
  lineSize = 0.5,
  lineAlpha = 1,
  lineType = 1,
  lineConfColor = brew_colors("grey"),
  lineConfSize = lineSize,
```

```
  lineConfAlpha = lineAlpha,
  lineConfType = 2,
  pVal = c(0.025, 0.975),
  sigma = attr(data, "broom_glance")$sigma,
  se = TRUE,
  method = "auto",
  formula = y ~ x
)
```

Arguments

`data, mapping, ...`	parameters supplied to `ggally_nostic_line`
`linePosition, lineColor, lineSize, lineAlpha, lineType`	parameters supplied to `ggplot2::geom_line()`
`lineConfColor, lineConfSize, lineConfAlpha, lineConfType`	parameters supplied to the confidence interval lines
`pVal`	percentiles of a N(0, sigma) distribution to be drawn
`sigma`	sigma value for the `pVal` percentiles
`se`	boolean to determine if the confidence intervals should be displayed
`method, formula`	parameters supplied to `ggplot2::geom_smooth()`. Defaults to `"auto"` and `"y ~ x"`

Value

ggplot2 plot object

See Also

`stats::residuals`

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

dt <- broomify(stats::lm(mpg ~ wt + qsec + am, data = mtcars))
p_(ggally_nostic_resid(dt, ggplot2::aes(wt, .resid)))
```

---

ggally_nostic_se_fit ggnostic *fitted value's standard error*

Description

A function to display `stats::predict`'s standard errors

Usage

```
ggally_nostic_se_fit(
  data,
  mapping,
  ...,
  lineColor = brew_colors("grey"),
  linePosition = NULL
)
```

Arguments

`data, mapping, ..., lineColor`	parameters supplied to `ggally_nostic_line`
`linePosition`	base comparison for a perfect fit

Details

As stated in `stats::predict` documentation:

If the logical 'se.fit' is 'TRUE', standard errors of the predictions are calculated. If the numeric argument 'scale' is set (with optional "df"), it is used as the residual standard deviation in the computation of the standard errors, otherwise this is extracted from the model fit.

Since the se.fit is `TRUE` and scale is unset by default, the standard errors are extracted from the model fit.

A base line of 0 is added to give reference to a perfect fit.

Value

ggplot2 plot object

See Also

`stats::influence()`

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

dt <- broomify(stats::lm(mpg ~ wt + qsec + am, data = mtcars))
p_(ggally_nostic_se_fit(dt, ggplot2::aes(wt, .se.fit)))
```

---

ggally_nostic_sigma `ggnostic` *leave one out model sigma*

Description

A function to display `stats::influence()`'s sigma value.

Usage

```
ggally_nostic_sigma(
  data,
  mapping,
  ...,
  lineColor = brew_colors("grey"),
  linePosition = attr(data, "broom_glance")$sigma
)
```

Arguments

`data, mapping, ..., lineColor`	parameters supplied to `ggally_nostic_line`
`linePosition`	line that is drawn in the background of the plot. Defaults to the overall model's sigma value.

Details

As stated in `stats::influence()` documentation:

sigma: a vector whose i-th element contains the estimate of the residual standard deviation obtained when the i-th case is dropped from the regression. (The approximations needed for GLMs can result in this being 'NaN'.)

A line is added to display the overall model's sigma value. This gives a baseline for comparison

Value

ggplot2 plot object

See Also

`stats::influence()`

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

dt <- broomify(stats::lm(mpg ~ wt + qsec + am, data = mtcars))
p_(ggally_nostic_sigma(dt, ggplot2::aes(wt, .sigma)))
```

---

ggally_nostic_std_resid

`ggnostic` *standardized residuals*

Description

If non-null `pVal` and `sigma` values are given, confidence interval lines will be added to the plot at the specified `pVal` locations of a N(0, 1) distribution.

Usage

```
ggally_nostic_std_resid(data, mapping, ..., sigma = 1)
```

Arguments

data, mapping, ...	parameters supplied to ggally_nostic_resid
sigma	sigma value for the pVal percentiles. Set to 1 for standardized residuals

Value

ggplot2 plot object

See Also

stats::rstandard()

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

dt <- broomify(stats::lm(mpg ~ wt + qsec + am, data = mtcars))
p_(ggally_nostic_std_resid(dt, ggplot2::aes(wt, .std.resid)))
```

---

ggally_points *Scatter plot*

Description

Make a scatter plot with a given data set.

Usage

```
ggally_points(data, mapping, ...)
```

Arguments

data	data set using
mapping	aesthetics being used
...	other arguments are sent to geom_point

Author(s)

Barret Schloerke

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

data(mtcars)
p_(ggally_points(mtcars, mapping = ggplot2::aes(x = disp, y = hp)))
p_(ggally_points(mtcars, mapping = ggplot2::aes_string(x = "disp", y = "hp")))
p_(ggally_points(
  mtcars,
```

```
  mapping = ggplot2::aes_string(
    x     = "disp",
    y     = "hp",
    color = "as.factor(cyl)",
    size  = "gear"
  )
))
```

---

ggally_ratio *Mosaic plot*

Description

Plots the mosaic plot by using fluctuation.

Usage

```
ggally_ratio(
  data,
  mapping = do.call(ggplot2::aes_string, as.list(colnames(data)[1:2])),
  ...,
  floor = 0,
  ceiling = NULL
)
```

Arguments

data	data set using
mapping	aesthetics being used. Only x and y will used and both are required
...	passed to geom_tile(...)
floor	don't display cells smaller than this value
ceiling	max value to scale frequencies. If any frequency is larger than the ceiling, the fill color is displayed darker than other rectangles

Author(s)

Barret Schloerke

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

data(tips, package = "reshape")
p_(ggally_ratio(tips, ggplot2::aes(sex, day)))
p_(ggally_ratio(tips, ggplot2::aes(sex, day)) + ggplot2::coord_equal())
# only plot tiles greater or equal to 20 and scale to a max of 50
p_(ggally_ratio(
  tips, ggplot2::aes(sex, day),
  floor = 20, ceiling = 50
) + ggplot2::theme(aspect.ratio = 4/2))
```

ggally_smooth *Scatter plot with a smoothed line*

Description

Add a smoothed condition mean with a given scatter plot.

Usage

```
ggally_smooth(
  data,
  mapping,
  ...,
  method = "lm",
  formula = y ~ x,
  se = TRUE,
  shrink = TRUE
)

ggally_smooth_loess(data, mapping, ...)

ggally_smooth_lm(data, mapping, ...)
```

Arguments

data	data set using
mapping	aesthetics being used
method, se	parameters supplied to geom_smooth
formula, ...	other arguments to add to geom_smooth
shrink	boolean to determine if y range is reduced to range of points or points and error ribbon

Details

Y limits are reduced to match original Y range with the goal of keeping the Y axis the same across plots.

Author(s)

Barret Schloerke

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

data(tips, package = "reshape")
p_(ggally_smooth(tips, mapping = ggplot2::aes(x = total_bill, y = tip)))
p_(ggally_smooth(tips, mapping = ggplot2::aes_string(x = "total_bill", y = "tip")))
p_(ggally_smooth(tips, mapping = ggplot2::aes_string(x = "total_bill", y = "tip", color = "sex")))
```

`ggally_statistic` *Generalized text display*

Description

Generalized text display

Usage

```
ggally_statistic(
  data,
  mapping,
  text_fn,
  title,
  na.rm = NA,
  display_grid = FALSE,
  justify_labels = "right",
  justify_text = "left",
  sep = ": ",
  family = "mono",
  title_args = list(),
  group_args = list(),
  align_percent = 0.5,
  title_hjust = 0.5,
  group_hjust = 0.5
)
```

Arguments

`data`	data set using
`mapping`	aesthetics being used
`text_fn`	function that takes in `x` and `y` and returns a text string
`title`	title text to be displayed
`na.rm`	logical value which determines if `NA` values are removed. If `TRUE`, no warning message will be displayed.
`display_grid`	if `TRUE`, display aligned panel grid lines. If `FALSE` (default), display a thin panel border.
`justify_labels`	`justify` argument supplied when `format`ting the labels
`justify_text`	`justify` argument supplied when `format`ting the returned `text_fn(x,y)` values
`sep`	separation value to be placed between the labels and text
`family`	font family used when displaying all text. This value will be set in `title_args` or `group_args` if no `family` value exists. By using `"mono"`, groups will align with each other.
`title_args`	arguments being supplied to the title's `geom_text()`
`group_args`	arguments being supplied to the split-by-color group's `geom_text()`
`align_percent`	relative align position of the text. When `title_hjust = 0.5` and `group_hjust = 0.5`, this should not be needed to be set.

`title_hjust, group_hjust`
: `hjust` sent to `geom_text()` for the title and group values respectively. Any `hjust` value supplied in `title_args` or `group_args` will take precedence.

See Also

`ggally_cor`

`ggally_summarise_by` *Summarize a continuous variable by each value of a discrete variable*

Description

Display summary statistics of a continuous variable for each value of a discrete variable.

Usage

```
ggally_summarise_by(
  data,
  mapping,
  text_fn = weighted_median_iqr,
  text_fn_vertical = NULL,
  ...
)

weighted_median_iqr(x, weights = NULL)

weighted_mean_sd(x, weights = NULL)
```

Arguments

`data`	data set using
`mapping`	aesthetics being used
`text_fn`	function that takes an x and weights and returns a text string
`text_fn_vertical`	function that takes an x and weights and returns a text string, used when `x` is discrete and `y` is continuous. If not provided, will use `text_fn`, replacing spaces by carriage returns.
`...`	other arguments passed to `geom_text(...)`
`x`	a numeric vector
`weights`	an optional numeric vectors of weights. If `NULL`, equal weights of 1 will be taken into account.

Details

`weighted_median_iqr` computes weighted median and interquartile range.

`weighted_mean_sd` computes weighted mean and standard deviation.

Author(s)

Joseph Larmarange

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

if (require(Hmisc)) {
  data(tips, package = "reshape")
  p_(ggally_summarise_by(tips, mapping = aes(x = total_bill, y = day)))
  p_(ggally_summarise_by(tips, mapping = aes(x = day, y = total_bill)))

  # colour is kept only if equal to the discrete variable
  p_(ggally_summarise_by(tips, mapping = aes(x = total_bill, y = day, color = day)))
  p_(ggally_summarise_by(tips, mapping = aes(x = total_bill, y = day, color = sex)))
  p_(ggally_summarise_by(tips, mapping = aes(x = day, y = total_bill, color = day)))

  # custom text size
  p_(ggally_summarise_by(tips, mapping = aes(x = total_bill, y = day), size = 6))

  # change statistic to display
 p_(ggally_summarise_by(tips, mapping = aes(x = total_bill, y = day), text_fn = weighted_mean_sd))

  # custom stat function
  weighted_sum <- function(x, weights = NULL) {
    if (is.null(weights)) weights <- 1
    paste0("Total : ", round(sum(x * weights, na.rm = TRUE), digits = 1))
  }
 p_(ggally_summarise_by(tips, mapping = aes(x = total_bill, y = day), text_fn = weighted_sum))
}
```

---

ggally_table *Display a table of the number of observations*

Description

Plot the number of observations as a table. Other statistics computed by `stat_cross` could be used (see examples).

Usage

```
ggally_table(
  data,
  mapping,
  keep.zero.cells = FALSE,
  ...,
  geom_tile_args = NULL
)

ggally_tableDiag(
  data,
  mapping,
  keep.zero.cells = FALSE,
  ...,
  geom_tile_args = NULL
)
```

Arguments

data	data set using
mapping	aesthetics being used
keep.zero.cells	If TRUE, display cells with no observation.
...	other arguments passed to geom_text(...)
geom_tile_args	other arguments passed to geom_tile(...)

Note

The **colour** aesthetic is taken into account only if equal to **x** or **y**.

Author(s)

Joseph Larmarange

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

data(tips, package = "reshape")
p_(ggally_table(tips, mapping = aes(x = smoker, y = sex)))
p_(ggally_table(tips, mapping = aes(x = day, y = time)))
p_(ggally_table(tips, mapping = aes(x = smoker, y = sex, colour = smoker)))

# colour is kept only if equal to x or y
p_(ggally_table(tips, mapping = aes(x = smoker, y = sex, colour = day)))

# diagonal version
p_(ggally_tableDiag(tips, mapping = aes(x = smoker)))

# custom label size and color
p_(ggally_table(tips, mapping = aes(x = smoker, y = sex), size = 16, color = "red"))

# display column proportions
p_(ggally_table(
  tips,
  mapping = aes(x = day, y = sex, label = scales::percent(after_stat(col.prop)))
))

# draw table cells
p_(ggally_table(
  tips,
  mapping = aes(x = smoker, y = sex),
  geom_tile_args = list(colour = "black", fill = "white")
))

# Use standardized residuals to fill table cells
p_(ggally_table(
  as.data.frame(Titanic),
  mapping = aes(
    x = Class, y = Survived, weight = Freq,
    fill = after_stat(std.resid),
    label = scales::percent(after_stat(col.prop), accuracy = .1)
```

```
  ),
  geom_tile_args = list(colour = "black")
) +
scale_fill_steps2(breaks = c(-3, -2, 2, 3), show.limits = TRUE))
```

---

ggally_text *Text plot*

Description

Plot text for a plot.

Usage

```
ggally_text(
  label,
  mapping = ggplot2::aes(color = "black"),
  xP = 0.5,
  yP = 0.5,
  xrange = c(0, 1),
  yrange = c(0, 1),
  ...
)
```

Arguments

label	text that you want to appear
mapping	aesthetics that don't relate to position (such as color)
xP	horizontal position percentage
yP	vertical position percentage
xrange	range of the data around it. Only nice to have if plotting in a matrix
yrange	range of the data around it. Only nice to have if plotting in a matrix
...	other arguments for geom_text

Author(s)

Barret Schloerke

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

p_(ggally_text("Example 1"))
p_(ggally_text("Example\nTwo", mapping = ggplot2::aes(size = 15), color = I("red")))
```

`ggally_trends` *Trends line plot*

Description

Plot trends using line plots. For continuous y variables, plot the evolution of the mean. For binary y variables, plot the evolution of the proportion.

Usage

```
ggally_trends(data, mapping, ..., include_zero = FALSE)
```

Arguments

`data`	data set using
`mapping`	aesthetics being used
`...`	other arguments passed to `ggplot2::geom_line()`
`include_zero`	Should 0 be included on the y-axis?

Author(s)

Joseph Larmarange

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

data(tips, package = "reshape")
tips_f <- tips
tips_f$day <- factor(tips$day, c("Thur", "Fri", "Sat", "Sun"))

# Numeric variable
p_(ggally_trends(tips_f, mapping = aes(x = day, y = total_bill)))
p_(ggally_trends(tips_f, mapping = aes(x = day, y = total_bill, colour = time)))

# Binary variable
p_(ggally_trends(tips_f, mapping = aes(x = day, y = smoker)))
p_(ggally_trends(tips_f, mapping = aes(x = day, y = smoker, colour = sex)))

# Discrete variable with 3 or more categories
p_(ggally_trends(tips_f, mapping = aes(x = smoker, y = day)))
p_(ggally_trends(tips_f, mapping = aes(x = smoker, y = day, color = sex)))

# Include zero on Y axis
p_(ggally_trends(tips_f, mapping = aes(x = day, y = total_bill), include_zero = TRUE))
p_(ggally_trends(tips_f, mapping = aes(x = day, y = smoker), include_zero = TRUE))

# Change line size
p_(ggally_trends(tips_f, mapping = aes(x = day, y = smoker, colour = sex), size = 3))

# Define weights with the appropriate aesthetic
d <- as.data.frame(Titanic)
```

```
p_(ggally_trends(
  d,
  mapping = aes(x = Class, y = Survived, weight = Freq, color = Sex),
  include_zero = TRUE
))
```

`ggbivariate` *Display an outcome using several potential explanatory variables*

Description

`ggbivariate` is a variant of `ggduo` for plotting an outcome variable with several potential explanatory variables.

Usage

```
ggbivariate(
  data,
  outcome,
  explanatory = NULL,
  mapping = NULL,
  types = NULL,
  ...,
  rowbar_args = NULL
)
```

Arguments

`data`	dataset to be used, can have both categorical and numerical variables
`outcome`	name or position of the outcome variable (one variable only)
`explanatory`	names or positions of the explanatory variables (if `NULL`, will take all variables other than `outcome`)
`mapping`	additional aesthetic to be used, for example to indicate weights (see examples)
`types`	custom types of plots to use, see `ggduo`
`...`	additional arguments passed to `ggduo` (see examples)
`rowbar_args`	additional arguments passed to `ggally_rowbar` (see examples)

Author(s)

Joseph Larmarange

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

data(tips, package = "reshape")
p_(ggbivariate(tips, "smoker", c("day", "time", "sex", "tip")))

# Personalize plot title and legend title
p_(ggbivariate(
```

```
  tips, "smoker", c("day", "time", "sex", "tip"),
  title = "Custom title"
) +
  labs(fill = "Smoker ?"))

# Customize fill colour scale
p_(ggbivariate(tips, "smoker", c("day", "time", "sex", "tip")) +
  scale_fill_brewer(type = "qual"))

# Customize labels
p_(ggbivariate(
  tips, "smoker", c("day", "time", "sex", "tip"),
  rowbar_args = list(
    colour = "white",
    size = 4,
    fontface = "bold",
    label_format = scales::label_percent(accurary = 1)
  )
))

# Choose the sub-plot from which get legend
p_(ggbivariate(tips, "smoker"))
p_(ggbivariate(tips, "smoker", legend = 3))

# Use mapping to indicate weights
d <- as.data.frame(Titanic)
p_(ggbivariate(d, "Survived", mapping = aes(weight = Freq)))

# outcome can be numerical
p_(ggbivariate(tips, outcome = "tip", title = "tip"))
```

---

ggcoef *Model coefficients with* **broom** *and* **ggplot2**

Description

Plot the coefficients of a model with **broom** and **ggplot2**. For an updated and improved version, see ggcoef_model().

Usage

```
ggcoef(
  x,
  mapping = aes_string(y = "term", x = "estimate"),
  conf.int = TRUE,
  conf.level = 0.95,
  exponentiate = FALSE,
  exclude_intercept = FALSE,
  vline = TRUE,
  vline_intercept = "auto",
  vline_color = "gray50",
  vline_linetype = "dotted",
  vline_size = 1,
```

```
  errorbar_color = "gray25",
  errorbar_height = 0,
  errorbar_linetype = "solid",
  errorbar_size = 0.5,
  sort = c("none", "ascending", "descending"),
  ...
)
```

Arguments

x	a model object to be tidied with broom::tidy() or a data frame (see Details)
mapping	default aesthetic mapping
conf.int	display confidence intervals as error bars?
conf.level	level of confidence intervals (passed to broom::tidy() if x is not a data frame)
exponentiate	if TRUE, x-axis will be logarithmic (also passed to broom::tidy() if x is not a data frame)
exclude_intercept	should the intercept be excluded from the plot?
vline	print a vertical line?
vline_intercept	xintercept for the vertical line. "auto" for x = 0 (or x = 1 if exponentiate is TRUE)
vline_color	color of the vertical line
vline_linetype	line type of the vertical line
vline_size	size of the vertical line
errorbar_color	color of the error bars
errorbar_height	height of the error bars
errorbar_linetype	line type of the error bars
errorbar_size	size of the error bars
sort	"none" (default) do not sort, "ascending" sort by increasing coefficient value, or "descending" sort by decreasing coefficient value
...	additional arguments sent to ggplot2::geom_point()

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

library(broom)
reg <- lm(Sepal.Length ~ Sepal.Width + Petal.Length + Petal.Width, data = iris)
p_(ggcoef(reg))
d <- as.data.frame(Titanic)
reg2 <- glm(Survived ~ Sex + Age + Class, family = binomial, data = d, weights = d$Freq)
ggcoef(reg2, exponentiate = TRUE)
ggcoef(
  reg2, exponentiate = TRUE, exclude_intercept = TRUE,
  errorbar_height = .2, color = "blue", sort = "ascending"
)
```

ggcoef_model *Plot model coefficients*

Description

Plot model coefficients

Usage

```
ggcoef_model(
  model,
  tidy_fun = broom::tidy,
  conf.int = TRUE,
  conf.level = 0.95,
  exponentiate = FALSE,
  variable_labels = NULL,
  term_labels = NULL,
  interaction_sep = " * ",
  categorical_terms_pattern = "{level}",
  add_reference_rows = TRUE,
  no_reference_row = NULL,
  intercept = FALSE,
  include = dplyr::everything(),
  significance = 1 - conf.level,
  significance_labels = NULL,
  show_p_values = TRUE,
  signif_stars = TRUE,
  return_data = FALSE,
  ...
)

ggcoef_compare(
  models,
  type = c("dodged", "faceted"),
  tidy_fun = broom::tidy,
  conf.int = TRUE,
  conf.level = 0.95,
  exponentiate = FALSE,
  variable_labels = NULL,
  term_labels = NULL,
  interaction_sep = " * ",
  categorical_terms_pattern = "{level}",
  add_reference_rows = TRUE,
  no_reference_row = NULL,
  intercept = FALSE,
  include = dplyr::everything(),
  significance = 1 - conf.level,
  significance_labels = NULL,
  return_data = FALSE,
  ...
)
```

```
ggcoef_multinom(
  model,
  type = c("dodged", "faceted"),
  y.level_label = NULL,
  tidy_fun = broom::tidy,
  conf.int = TRUE,
  conf.level = 0.95,
  exponentiate = FALSE,
  variable_labels = NULL,
  term_labels = NULL,
  interaction_sep = " * ",
  categorical_terms_pattern = "{level}",
  add_reference_rows = TRUE,
  no_reference_row = NULL,
  intercept = FALSE,
  include = dplyr::everything(),
  significance = 1 - conf.level,
  significance_labels = NULL,
  show_p_values = TRUE,
  signif_stars = TRUE,
  return_data = FALSE,
  ...
)

ggcoef_plot(
  data,
  x = "estimate",
  y = "label",
  exponentiate = FALSE,
  point_size = 2,
  point_stroke = 2,
  point_fill = "white",
  colour = NULL,
  colour_guide = TRUE,
  colour_lab = "",
  colour_labels = ggplot2::waiver(),
  shape = "significance",
  shape_values = c(16, 21),
  shape_guide = TRUE,
  shape_lab = "",
  errorbar = TRUE,
  errorbar_height = 0.1,
  errorbar_coloured = FALSE,
  stripped_rows = TRUE,
  strips_odd = "#11111111",
  strips_even = "#00000000",
  vline = TRUE,
  vline_colour = "grey50",
  dodged = FALSE,
  dodged_width = 0.8,
  facet_row = "var_label",
```

```
  facet_col = NULL,
  facet_labeller = "label_value"
)
```

Arguments

model	a regression model object
tidy_fun	option to specify a custom tidier function
conf.int	should confidence intervals be computed? (see broom::tidy())
conf.level	the confidence level to use for the confidence interval if conf.int = TRUE; must be strictly greater than 0 and less than 1; defaults to 0.95, which corresponds to a 95 percent confidence interval
exponentiate	if TRUE a logarithmic scale will be used for x-axis
variable_labels	a named list or a named vector of custom variable labels
term_labels	a named list or a named vector of custom term labels
interaction_sep	separator for interaction terms
categorical_terms_pattern	a glue pattern for labels of categorical terms with treatment or sum contrasts (see model_list_terms_levels())
add_reference_rows	should reference rows be added?
no_reference_row	variables (accepts tidyselect notation) for those no reference row should be added, when add_reference_rows = TRUE
intercept	should the intercept(s) be included?
include	variables to include. Accepts tidyselect syntax. Use - to remove a variable. Default is everything(). See also all_continuous(), all_categorical(), all_dichotomous() and all_interaction()
significance	level (between 0 and 1) below which a coefficient is consider to be significantly different from 0 (or 1 if exponentiate = TRUE), NULL for not highlighting such coefficients
significance_labels	optional vector with custom labels for significance variable
show_p_values	if TRUE, add p-value to labels
signif_stars	if TRUE, add significant stars to labels
return_data	if TRUE, will return the data.frame used for plotting instead of the plot
...	parameters passed to ggcoef_plot()
models	named list of models
type	a dodged plot or a faceted plot?
y.level_label	an optional named vector for labeling y.level (see examples)
data	a data frame containing data to be plotted, typically the output of ggcoef_model(), ggcoef_compare() or ggcoef_multinom() with the option return_data = TRUE
x, y	variables mapped to x and y axis
point_size	size of the points

point_stroke	thickness of the points
point_fill	fill colour for the points
colour	optional variable name to be mapped to colour aesthetic
colour_guide	should colour guide be displayed in the legend?
colour_lab	label of the colour aesthetic in the legend
colour_labels	labels argument passed to ggplot2::scale_colour_discrete() and ggplot2::discrete_scale(
shape	optional variable name to be mapped to the shape aesthetic
shape_values	values of the different shapes to use in ggplot2::scale_shape_manual()
shape_guide	should shape guide be displayed in the legend?
shape_lab	label of the shape aesthetic in the legend
errorbar	should error bars be plotted?
errorbar_height	height of error bars
errorbar_coloured	should error bars be colored as the points?
stripped_rows	should stripped rows be displayed in the background?
strips_odd	color of the odd rows
strips_even	color of the even rows
vline	should a vertical line be drawn at 0 (or 1 if exponentiate = TRUE)?
vline_colour	colour of vertical line
dodged	should points be dodged (according to the colour aesthetic)?
dodged_width	width value for ggplot2::position_dodge()
facet_row	variable name to be used for row facets
facet_col	optional variable name to be used for column facets
facet_labeller	labeller function to be used for labeling facets; if labels are too long, you can use ggplot2::label_wrap_gen() (see examples), more information in the documentation of ggplot2::facet_grid()

Details

ggcoef_model(), ggcoef_multinom() and ggcoef_compare() use broom.helpers::tidy_plus_plus() to obtain a tibble of the model coefficients, apply additional data transformation and then pass the produced tibble to ggcoef_plot() to generate the plot.

For more control, you can use the argument return_data = TRUE to get the produced tibble, apply any transformation of your own and then pass your customized tibble to ggcoef_plot().

Functions

- ggcoef_model: Redesign of ggcoef() based on broom.helpers::tidy_plus_plus().
- ggcoef_compare: Designed for displaying several models on the same plot.
- ggcoef_multinom: A variation of ggcoef_model() adapted to multinomial logistic regressions performed with nnet::multinom().
- ggcoef_plot: SOME DESCRIPTION HERE

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

if (require(broom.helpers)) {
  data(tips, package = "reshape")
  mod_simple <- lm(tip ~ day + time + total_bill, data = tips)
  p_(ggcoef_model(mod_simple))

  # custom variable labels
  # you can use the labelled package to define variable labels before computing model
  if (require(labelled)) {
    tips_labelled <- tips %>%
      labelled::set_variable_labels(
        day = "Day of the week",
        time = "Lunch or Dinner",
        total_bill = "Bill's total"
      )
    mod_labelled <- lm(tip ~ day + time + total_bill, data = tips_labelled)
    p_(ggcoef_model(mod_labelled))
  }
  # you can provide custom variable labels with 'variable_labels'
  p_(ggcoef_model(
    mod_simple,
    variable_labels = c(
      day = "Week day",
      time = "Time (lunch or dinner ?)",
      total_bill = "Total of the bill"
    )
  ))
  # if labels are too long, you can use 'facet_labeller' to wrap them
  p_(ggcoef_model(
    mod_simple,
    variable_labels = c(
      day = "Week day",
      time = "Time (lunch or dinner ?)",
      total_bill = "Total of the bill"
    ),
    facet_labeller = label_wrap_gen(10)
  ))

  # do not display variable facets but add colour guide
  p_(ggcoef_model(mod_simple, facet_row = NULL, colour_guide = TRUE))

  # a logistic regression example
  d_titanic <- as.data.frame(Titanic)
  d_titanic$Survived <- factor(d_titanic$Survived, c("No", "Yes"))
  mod_titanic <- glm(
    Survived ~ Sex * Age + Class,
    weights = Freq,
    data = d_titanic,
    family = binomial
  )

  # use 'exponentiate = TRUE' to get the Odds Ratio
  p_(ggcoef_model(mod_titanic, exponentiate = TRUE))
```

```
# display intercepts
p_(ggcoef_model(mod_titanic, exponentiate = TRUE, intercept = TRUE))

# customize terms labels
p_(
  ggcoef_model(
    mod_titanic,
    exponentiate = TRUE,
    show_p_values = FALSE,
    signif_stars = FALSE,
    add_reference_rows = FALSE,
    categorical_terms_pattern = "{level} (ref: {reference_level})",
    interaction_sep = " x "
  ) +
  scale_y_discrete(labels = scales::label_wrap(15))
)

# display only a subset of terms
p_(ggcoef_model(mod_titanic, exponentiate = TRUE, include = c("Age", "Class")))

# do not change points' shape based on significance
p_(ggcoef_model(mod_titanic, exponentiate = TRUE, significance = NULL))

# a black and white version
p_(ggcoef_model(
  mod_titanic, exponentiate = TRUE,
  colour = NULL, stripped_rows = FALSE
))

# show dichotomous terms on one row
p_(ggcoef_model(
  mod_titanic,
  exponentiate = TRUE,
  no_reference_row = broom.helpers::all_dichotomous(),
  categorical_terms_pattern =
    "{ifelse(dichotomous, paste0(level, ' / ', reference_level), level)}",
  show_p_values = FALSE
))

# works also with with polynomial terms
mod_poly <- lm(
  tip ~ poly(total_bill, 3) + day,
  data = tips,
)
p_(ggcoef_model(mod_poly))

# or with different type of contrasts
# for sum contrasts, the value of the reference term is computed
if (require(emmeans)) {
  mod2 <- lm(
    tip ~ day + time + sex,
    data = tips,
    contrasts = list(time = contr.sum, day = contr.treatment(4, base = 3))
  )
  p_(ggcoef_model(mod2))
}
```

```
}

if (require(broom.helpers)) {
  # Use ggcoef_compare() for comparing several models on the same plot
  mod1 <- lm(Fertility ~ ., data = swiss)
  mod2 <- step(mod1, trace = 0)
  mod3 <- lm(Fertility ~ Agriculture + Education * Catholic, data = swiss)
  models <- list("Full model" = mod1, "Simplified model" = mod2, "With interaction" = mod3)

  p_(ggcoef_compare(models))
  p_(ggcoef_compare(models, type = "faceted"))

  # you can reverse the vertical position of the point by using a negative value
  # for dodged_width (but it will produce some warnings)
## Not run:
  p_(ggcoef_compare(models, dodged_width = -.9))

## End(Not run)
}

# specific function for nnet::multinom models
if (require(broom.helpers) && require(nnet)) {
  data(happy)
  mod <- multinom(happy ~ age + degree + sex, data = happy)
  p_(ggcoef_multinom(mod, exponentiate = TRUE))
  p_(ggcoef_multinom(mod, type = "faceted"))
  p_(ggcoef_multinom(
    mod, type = "faceted",
    y.level = c(
      "pretty happy" = "pretty happy\n(ref: very happy)",
      "very happy" = "very happy"
    )
  ))
}
```

---

ggcorr *Correlation matrix*

Description

Function for making a correlation matrix plot, using **ggplot2**. The function is directly inspired by Tian Zheng and Yu-Sung Su's `corrplot` function in the 'arm' package. Please visit https://github.com/briatte/ggcorr for the latest version of `ggcorr`, and see the vignette at https://briatte.github.io/ggcorr/ for many examples of how to use it.

Usage

```
ggcorr(
  data,
  method = c("pairwise", "pearson"),
  cor_matrix = NULL,
  nbreaks = NULL,
  digits = 2,
  name = "",
```

```
  low = "#3B9AB2",
  mid = "#EEEEEE",
  high = "#F21A00",
  midpoint = 0,
  palette = NULL,
  geom = "tile",
  min_size = 2,
  max_size = 6,
  label = FALSE,
  label_alpha = FALSE,
  label_color = "black",
  label_round = 1,
  label_size = 4,
  limits = c(-1, 1),
  drop = is.null(limits) || identical(limits, FALSE),
  layout.exp = 0,
  legend.position = "right",
  legend.size = 9,
  ...
)
```

Arguments

data	a data frame or matrix containing numeric (continuous) data. If any of the columns contain non-numeric data, they will be dropped with a warning.
method	a vector of two character strings. The first value gives the method for computing covariances in the presence of missing values, and must be (an abbreviation of) one of "everything", "all.obs", "complete.obs", "na.or.complete" or "pairwise.complete.obs". The second value gives the type of correlation coefficient to compute, and must be one of "pearson", "kendall" or "spearman". See cor for details. Defaults to c("pairwise","pearson").
cor_matrix	the named correlation matrix to use for calculations. Defaults to the correlation matrix of data when data is supplied.
nbreaks	the number of breaks to apply to the correlation coefficients, which results in a categorical color scale. See 'Note'. Defaults to NULL (no breaks, continuous scaling).
digits	the number of digits to show in the breaks of the correlation coefficients: see cut for details. Defaults to 2.
name	a character string for the legend that shows the colors of the correlation coefficients. Defaults to "" (no legend name).
low	the lower color of the gradient for continuous scaling of the correlation coefficients. Defaults to "#3B9AB2" (blue).
mid	the midpoint color of the gradient for continuous scaling of the correlation coefficients. Defaults to "#EEEEEE" (very light grey).
high	the upper color of the gradient for continuous scaling of the correlation coefficients. Defaults to "#F21A00" (red).
midpoint	the midpoint value for continuous scaling of the correlation coefficients. Defaults to 0.
palette	if nbreaks is used, a ColorBrewer palette to use instead of the colors specified by low, mid and high. Defaults to NULL.

geom	the geom object to use. Accepts either "tile", "circle", "text" or "blank".
min_size	when geom has been set to "circle", the minimum size of the circles. Defaults to 2.
max_size	when geom has been set to "circle", the maximum size of the circles. Defaults to 6.
label	whether to add correlation coefficients to the plot. Defaults to FALSE.
label_alpha	whether to make the correlation coefficients increasingly transparent as they come close to 0. Also accepts any numeric value between 0 and 1, in which case the level of transparency is set to that fixed value. Defaults to FALSE (no transparency).
label_color	the color of the correlation coefficients. Defaults to "grey75".
label_round	the decimal rounding of the correlation coefficients. Defaults to 1.
label_size	the size of the correlation coefficients. Defaults to 4.
limits	bounding of color scaling for correlations, set limits = NULL or FALSE to remove
drop	if using nbreaks, whether to drop unused breaks from the color scale. Defaults to FALSE (recommended).
layout.exp	a multiplier to expand the horizontal axis to the left if variable names get clipped. Defaults to 0 (no expansion).
legend.position	where to put the legend of the correlation coefficients: see theme for details. Defaults to "bottom".
legend.size	the size of the legend title and labels, in points: see theme for details. Defaults to 9.
...	other arguments supplied to geom_text for the diagonal labels.

Note

Recommended values for the nbreaks argument are 3 to 11, as values above 11 are visually difficult to separate and are not supported by diverging ColorBrewer palettes.

Author(s)

Francois Briatte, with contributions from Amos B. Elberg and Barret Schloerke

See Also

cor and corrplot in the arm package.

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

# Basketball statistics provided by Nathan Yau at Flowing Data.
dt <- read.csv("http://datasets.flowingdata.com/ppg2008.csv")

# Default output.
p_(ggcorr(dt[, -1]))
```

```
# Labeled output, with coefficient transparency.
p_(ggcorr(dt[, -1],
       label = TRUE,
       label_alpha = TRUE))

# Custom options.
p_(ggcorr(
  dt[, -1],
  name = expression(rho),
  geom = "circle",
  max_size = 10,
  min_size = 2,
  size = 3,
  hjust = 0.75,
  nbreaks = 6,
  angle = -45,
  palette = "PuOr" # colorblind safe, photocopy-able
))

# Supply your own correlation matrix
p_(ggcorr(
  data = NULL,
  cor_matrix = cor(dt[, -1], use = "pairwise")
))
```

---

ggduo **ggplot2** *generalized pairs plot for two columns sets of data*

Description

Make a matrix of plots with a given data set with two different column sets

Usage

```
ggduo(
  data,
  mapping = NULL,
  columnsX = 1:ncol(data),
  columnsY = 1:ncol(data),
  title = NULL,
  types = list(continuous = "smooth_loess", comboVertical = "box_no_facet",
    comboHorizontal = "facethist", discrete = "count"),
  axisLabels = c("show", "none"),
  columnLabelsX = colnames(data[columnsX]),
  columnLabelsY = colnames(data[columnsY]),
  labeller = "label_value",
  switch = NULL,
  xlab = NULL,
  ylab = NULL,
  showStrips = NULL,
  legend = NULL,
  cardinality_threshold = 15,
  progress = NULL,
```

```
  xProportions = NULL,
  yProportions = NULL,
  legends = stop("deprecated")
)
```

Arguments

`data`	data set using. Can have both numerical and categorical data.
`mapping`	aesthetic mapping (besides `x` and `y`). See `aes()`. If `mapping` is numeric, `columns` will be set to the `mapping` value and `mapping` will be set to `NULL`.
`columnsX, columnsY`	which columns are used to make plots. Defaults to all columns.
`title, xlab, ylab`	title, x label, and y label for the graph
`types`	see Details
`axisLabels`	either "show" to display axisLabels or "none" for no axis labels
`columnLabelsX, columnLabelsY`	label names to be displayed. Defaults to names of columns being used.
`labeller`	labeller for facets. See `labellers`. Common values are `"label_value"` (default) and `"label_parsed"`.
`switch`	switch parameter for facet_grid. See `ggplot2::facet_grid`. By default, the labels are displayed on the top and right of the plot. If `"x"`, the top labels will be displayed to the bottom. If `"y"`, the right-hand side labels will be displayed to the left. Can also be set to `"both"`
`showStrips`	boolean to determine if each plot's strips should be displayed. `NULL` will default to the top and right side plots only. `TRUE` or `FALSE` will turn all strips on or off respectively.
`legend`	May be the two objects described below or the default `NULL` value. The legend position can be moved by using ggplot2's theme element `pm + theme(legend.position = "bottom")`
	a numeric vector of length 2 provides the location of the plot to use the legend for the plot matrix's legend. Such as `legend = c(3,5)` which will use the legend from the plot in the third row and fifth column
	a single numeric value provides the location of a plot according to the display order. Such as `legend = 3` in a plot matrix with 2 rows and 5 columns displayed by column will return the plot in position `c(1,2)`
	a object from `grab_legend()` a predetermined plot legend that will be displayed directly
`cardinality_threshold`	maximum number of levels allowed in a character / factor column. Set this value to NULL to not check factor columns. Defaults to 15
`progress`	`NULL` (default) for a progress bar in interactive sessions with more than 15 plots, `TRUE` for a progress bar, `FALSE` for no progress bar, or a function that accepts at least a plot matrix and returns a new `progress::progress_bar`. See `ggmatrix_progress`.
`xProportions, yProportions`	Value to change how much area is given for each plot. Either `NULL` (default), numeric value matching respective length, `grid::unit` object with matching respective length or `"auto"` for automatic relative proportions based on the number of levels for categorical variables.
`legends`	deprecated

Details

types is a list that may contain the variables 'continuous', 'combo', 'discrete', and 'na'. Each element of the list may be a function or a string. If a string is supplied, If a string is supplied, it must be a character string representing the tail end of a ggally_NAME function. The list of current valid ggally_NAME functions is visible in a dedicated vignette.

continuous This option is used for continuous X and Y data.

comboHorizontal This option is used for either continuous X and categorical Y data or categorical X and continuous Y data.

comboVertical This option is used for either continuous X and categorical Y data or categorical X and continuous Y data.

discrete This option is used for categorical X and Y data.

na This option is used when all X data is NA, all Y data is NA, or either all X or Y data is NA.

If 'blank' is ever chosen as an option, then ggduo will produce an empty plot.

If a function is supplied as an option, it should implement the function api of function(data,mapping,...){#make ggplot2 plot}. If a specific function needs its parameters set, wrap(fn,param1 = val1,param2 = val2) the function with its parameters.

Examples

```
# small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

data(baseball, package = "plyr")

# Keep players from 1990-1995 with at least one at bat
# Add how many singles a player hit
# (must do in two steps as X1b is used in calculations)
dt <- transform(
  subset(baseball, year >= 1990 & year <= 1995 & ab > 0),
  X1b = h - X2b - X3b - hr
)
# Add
#  the player's batting average,
#  the player's slugging percentage,
#  and the player's on base percentage
# Make factor a year, as each season is discrete
dt <- transform(
  dt,
  batting_avg = h / ab,
  slug = (X1b + 2*X2b + 3*X3b + 4*hr) / ab,
  on_base = (h + bb + hbp) / (ab + bb + hbp),
  year = as.factor(year)
)


pm <- ggduo(
  dt,
  c("year", "g", "ab", "lg"),
  c("batting_avg", "slug", "on_base"),
  mapping = ggplot2::aes(color = lg)
)
# Prints, but
```

```
#   there is severe over plotting in the continuous plots
#   the labels could be better
#   want to add more hitting information
p_(pm)

# address overplotting issues and add a title
pm <- ggduo(
  dt,
  c("year", "g", "ab", "lg"),
  c("batting_avg", "slug", "on_base"),
  columnLabelsX = c("year", "player game count", "player at bat count", "league"),
  columnLabelsY = c("batting avg", "slug %", "on base %"),
  title = "Baseball Hitting Stats from 1990-1995",
  mapping = ggplot2::aes(color = lg),
  types = list(
    # change the shape and add some transparency to the points
    continuous = wrap("smooth_loess", alpha = 0.50, shape = "+")
  ),
  showStrips = FALSE
)

p_(pm)

# Use "auto" to adapt width of the sub-plots
pm <- ggduo(
  dt,
  c("year", "g", "ab", "lg"),
  c("batting_avg", "slug", "on_base"),
  mapping = ggplot2::aes(color = lg),
  xProportions = "auto"
)

p_(pm)

# Custom widths & heights of the sub-plots
pm <- ggduo(
  dt,
  c("year", "g", "ab", "lg"),
  c("batting_avg", "slug", "on_base"),
  mapping = ggplot2::aes(color = lg),
  xProportions = c(6, 4, 3, 2),
  yProportions = c(1, 2, 1)
)

p_(pm)

# Example derived from:
## R Data Analysis Examples | Canonical Correlation Analysis.  UCLA: Institute for Digital
##   Research and Education.
##   from http://www.stats.idre.ucla.edu/r/dae/canonical-correlation-analysis
##   (accessed May 22, 2017).
# "Example 1. A researcher has collected data on three psychological variables, four
#  academic variables (standardized test scores) and gender for 600 college freshman.
#  She is interested in how the set of psychological variables relates to the academic
#  variables and gender. In particular, the researcher is interested in how many
#  dimensions (canonical variables) are necessary to understand the association between
#  the two sets of variables."
```

```
data(psychademic)
summary(psychademic)

(psych_variables <- attr(psychademic, "psychology"))
(academic_variables <- attr(psychademic, "academic"))

## Within correlation
p_(ggpairs(psychademic, columns = psych_variables))
p_(ggpairs(psychademic, columns = academic_variables))

## Between correlation
loess_with_cor <- function(data, mapping, ..., method = "pearson") {
  x <- eval_data_col(data, mapping$x)
  y <- eval_data_col(data, mapping$y)
  cor <- cor(x, y, method = method)
  ggally_smooth_loess(data, mapping, ...) +
    ggplot2::geom_label(
      data = data.frame(
        x = min(x, na.rm = TRUE),
        y = max(y, na.rm = TRUE),
        lab = round(cor, digits = 3)
      ),
      mapping = ggplot2::aes(x = x, y = y, label = lab),
      hjust = 0, vjust = 1,
      size = 5, fontface = "bold",
      inherit.aes = FALSE # do not inherit anything from the ...
    )
}
pm <- ggduo(
  psychademic,
  rev(psych_variables), academic_variables,
  types = list(continuous = loess_with_cor),
  showStrips = FALSE
)
suppressWarnings(p_(pm)) # ignore warnings from loess

# add color according to sex
pm <- ggduo(
  psychademic,
  mapping = ggplot2::aes(color = sex),
  rev(psych_variables), academic_variables,
  types = list(continuous = loess_with_cor),
  showStrips = FALSE,
  legend = c(5,2)
)
suppressWarnings(p_(pm))


# add color according to sex
pm <- ggduo(
  psychademic,
  mapping = ggplot2::aes(color = motivation),
  rev(psych_variables), academic_variables,
  types = list(continuous = loess_with_cor),
  showStrips = FALSE,
  legend = c(5,2)
) +
```

```
  ggplot2::theme(legend.position = "bottom")
suppressWarnings(p_(pm))
```

---

ggfacet *Single* **ggplot2** *plot matrix with* facet_grid

Description

Single **ggplot2** plot matrix with facet_grid

Usage

```
ggfacet(
  data,
  mapping = NULL,
  columnsX = 1:ncol(data),
  columnsY = 1:ncol(data),
  fn = ggally_points,
  ...,
  columnLabelsX = names(data[columnsX]),
  columnLabelsY = names(data[columnsY]),
  xlab = NULL,
  ylab = NULL,
  title = NULL,
  scales = "free"
)
```

Arguments

data	data.frame that contains all columns to be displayed. This data will be melted before being passed into the function fn
mapping	aesthetic mapping (besides x and y). See aes()
columnsX	columns to be displayed in the plot matrix
columnsY	rows to be displayed in the plot matrix
fn	function to be executed. Similar to ggpairs and ggduo, the function may either be a string identifier or a real function that wrap understands.
...	extra arguments passed directly to fn
columnLabelsX, columnLabelsY	column and row labels to display in the plot matrix
xlab, ylab, title	plot matrix labels
scales	parameter supplied to ggplot2::facet_grid. Default behavior is "free"

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive
if (requireNamespace("chemometrics", quietly = TRUE)) {
  data(NIR, package = "chemometrics")
  NIR_sub <- data.frame(NIR$yGlcEtOH, NIR$xNIR[,1:3])
  str(NIR_sub)
  x_cols <- c("X1115.0", "X1120.0", "X1125.0")
  y_cols <- c("Glucose", "Ethanol")

  # using ggduo directly
  p <- ggduo(NIR_sub, x_cols, y_cols, types = list(continuous = "points"))
  p_(p)

  # using ggfacet
  p <- ggfacet(NIR_sub, x_cols, y_cols)
  p_(p)

  # add a smoother
  p <- ggfacet(NIR_sub, x_cols, y_cols, fn = 'smooth_loess')
  p_(p)
  # same output
  p <- ggfacet(NIR_sub, x_cols, y_cols, fn = ggally_smooth_loess)
  p_(p)

  # Change scales to be the same in for every row and for every column
  p <- ggfacet(NIR_sub, x_cols, y_cols, scales = "fixed")
  p_(p)
}
```

`gglegend` *Plot only legend of plot function*

Description

Plot only legend of plot function

Usage

```
gglegend(fn)
```

Arguments

`fn`	this value is passed directly to an empty `wrap` call. Please see ?`wrap` for more details.

Value

a function that when called with arguments will produce the legend of the plotting function supplied.

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

# display regular plot
p_(ggally_points(iris, ggplot2::aes(Sepal.Length, Sepal.Width, color = Species)))

# Make a function that will only print the legend
points_legend <- gglegend(ggally_points)
p_(points_legend(iris, ggplot2::aes(Sepal.Length, Sepal.Width, color = Species)))

# produce the sample legend plot, but supply a string that 'wrap' understands
same_points_legend <- gglegend("points")
identical(
  attr(attr(points_legend, "fn"), "original_fn"),
  attr(attr(same_points_legend, "fn"), "original_fn")
)

# Complicated examples
custom_legend <- wrap(gglegend("points"), size = 6)
p_(custom_legend(iris, ggplot2::aes(Sepal.Length, Sepal.Width, color = Species)))

# Use within ggpairs
pm <- ggpairs(
  iris, 1:2,
  mapping = ggplot2::aes(color = Species),
  upper = list(continuous = gglegend("points"))
)
p_(pm)

# Place a legend in a specific location
pm <- ggpairs(iris, 1:2, mapping = ggplot2::aes(color = Species))
# Make the legend
pm[1,2] <- points_legend(iris, ggplot2::aes(Sepal.Width, Sepal.Length, color = Species))
p_(pm)
```

---

ggmatrix **ggplot2** *plot matrix*

Description

Make a generic matrix of **ggplot2** plots.

Usage

```
ggmatrix(
  plots,
  nrow,
  ncol,
  xAxisLabels = NULL,
  yAxisLabels = NULL,
  title = NULL,
  xlab = NULL,
```

```
  ylab = NULL,
  byrow = TRUE,
  showStrips = NULL,
  showAxisPlotLabels = TRUE,
  showXAxisPlotLabels = TRUE,
  showYAxisPlotLabels = TRUE,
  labeller = NULL,
  switch = NULL,
  xProportions = NULL,
  yProportions = NULL,
  progress = NULL,
  data = NULL,
  gg = NULL,
  legend = NULL
)
```

Arguments

`plots`	list of plots to be put into matrix
`nrow, ncol`	number of rows and columns
`xAxisLabels, yAxisLabels`	strip titles for the x and y axis respectively. Set to `NULL` to not be displayed
`title, xlab, ylab`	title, x label, and y label for the graph. Set to `NULL` to not be displayed
`byrow`	boolean that determines whether the plots should be ordered by row or by column
`showStrips`	boolean to determine if each plot's strips should be displayed. `NULL` will default to the top and right side plots only. `TRUE` or `FALSE` will turn all strips on or off respectively.
`showAxisPlotLabels, showXAxisPlotLabels, showYAxisPlotLabels`	booleans that determine if the plots axis labels are printed on the X (bottom) or Y (left) part of the plot matrix. If `showAxisPlotLabels` is set, both `showXAxisPlotLabels` and `showYAxisPlotLabels` will be set to the given value.
`labeller`	labeller for facets. See `labellers`. Common values are `"label_value"` (default) and `"label_parsed"`.
`switch`	switch parameter for facet_grid. See `ggplot2::facet_grid`. By default, the labels are displayed on the top and right of the plot. If `"x"`, the top labels will be displayed to the bottom. If `"y"`, the right-hand side labels will be displayed to the left. Can also be set to `"both"`
`xProportions, yProportions`	Value to change how much area is given for each plot. Either `NULL` (default), numeric value matching respective length, or `grid::unit` object with matching respective length
`progress`	`NULL` (default) for a progress bar in interactive sessions with more than 15 plots, `TRUE` for a progress bar, `FALSE` for no progress bar, or a function that accepts at least a plot matrix and returns a new `progress::progress_bar`. See `ggmatrix_progress`.
`data`	data set using. This is the data to be used in place of 'ggally_data' if the plot is a string to be evaluated at print time
`gg`	**ggplot2** theme objects to be applied to every plot

legend — May be the two objects described below or the default NULL value. The legend position can be moved by using ggplot2's theme element pm + theme(legend.position = "bottom")

a numeric vector of length 2 provides the location of the plot to use the legend for the plot matrix's legend. Such as legend = c(3,5) which will use the legend from the plot in the third row and fifth column

a single numeric value provides the location of a plot according to the display order. Such as legend = 3 in a plot matrix with 2 rows and 5 columns displayed by column will return the plot in position c(1,2)

a object from grab_legend() a predetermined plot legend that will be displayed directly

Memory usage

Now that the print.ggmatrix method uses a large **gtable** object, rather than print each plot independently, memory usage may be of concern. From small tests, memory usage flutters around object.size(data) * 0.3 * length(plots). So, for a 80Mb random noise dataset with 100 plots, about 2.4 Gb of memory needed to print. For the 3.46 Mb diamonds dataset with 100 plots, about 100 Mb of memory was needed to print. The benefits of using the **ggplot2** format greatly outweigh the price of about 20% increase in memory usage from the prior ad-hoc print method.

Author(s)

Barret Schloerke

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

plotList <- list()
for (i in 1:6) {
  plotList[[i]] <- ggally_text(paste("Plot #", i, sep = ""))
}
pm <- ggmatrix(
  plotList,
  2, 3,
  c("A", "B", "C"),
  c("D", "E"),
  byrow = TRUE
)
p_(pm)

pm <- ggmatrix(
  plotList,
  2, 3,
  xAxisLabels = c("A", "B", "C"),
  yAxisLabels = NULL,
  byrow = FALSE,
  showXAxisPlotLabels = FALSE
)
p_(pm)
```

ggmatrix_gtable ggmatrix gtable *object*

Description

Specialized method to print the ggmatrix object.

Usage

```
ggmatrix_gtable(
  pm,
  ...,
  progress = NULL,
  progress_format = formals(ggmatrix_progress)$format
)
```

Arguments

pm	ggmatrix object to be plotted
...	ignored
progress, progress_format	Please use the 'progress' parameter in your ggmatrix-like function. See ggmatrix_progress for a few examples. These parameters will soon be deprecated.

Author(s)

Barret Schloerke

Examples

```
data(tips, package = "reshape")
pm <- ggpairs(tips, c(1,3,2), mapping = ggplot2::aes_string(color = "sex"))
ggmatrix_gtable(pm)
```

ggmatrix_location ggmatrix *plot locations*

Description

[Experimental]

Usage

```
ggmatrix_location(pm, location = NULL, rows = NULL, cols = NULL)
```

Arguments

pm	ggmatrix plot object
location	"all", TRUE All row and col combinations "none" No row and column combinations "upper" Locations where the column value is higher than the row value "lower" Locations where the row value is higher than the column value "diag" Locations where the column value is equal to the row value matrix **or** data.frame matrix values will be converted into data.frames. • A data.frame with the exact column names c("row","col") • A data.frame with the number of rows and columns matching the plot matrix object provided. Each cell will be tested for a "truthy" value to determine if the location should be kept.
rows	numeric vector of the rows to be used. Will be used with cols if location is NULL
cols	numeric vector of the cols to be used. Will be used with rows if location is NULL

Details

Convert many types of location values to a consistent data.frame of row and col values.

Value

Data frame with columns c("row","col") containing locations for the plot matrix

Examples

```
pm <- ggpairs(reshape::tips, 1:3)

# All locations
ggmatrix_location(pm, location = "all")
ggmatrix_location(pm, location = TRUE)

# No locations
ggmatrix_location(pm, location = "none")

# "upper" triangle locations
ggmatrix_location(pm, location = "upper")

# "lower" triangle locations
ggmatrix_location(pm, location = "lower")

# "diag" locations
ggmatrix_location(pm, location = "diag")

# specific rows
ggmatrix_location(pm, rows = 2)

# specific columns
ggmatrix_location(pm, cols = 2)

# row and column combinations
ggmatrix_location(pm, rows = c(1,2), cols = c(1,3))
```

```
# matrix locations
mat <- matrix(TRUE, ncol = 3, nrow = 3)
mat[1,1] <- FALSE
locs <- ggmatrix_location(pm, location = mat)
## does not contain the 1,1 cell
locs

# Use the output of a prior ggmatrix_location
ggmatrix_location(pm, location = locs)
```

---

ggmatrix_progress *ggmatrix default progress bar*

Description

ggmatrix default progress bar

Usage

```
ggmatrix_progress(
  format = " plot: [:plot_i,:plot_j] [:bar]:percent est::eta ",
  clear = TRUE,
  show_after = 0,
  ...
)
```

Arguments

format, clear, show_after, ...
: parameters supplied directly to progress::progress_bar$new()

Value

function that accepts a plot matrix as the first argument and ... for future expansion. Internally, the plot matrix is used to determine the total number of plots for the progress bar.

Examples

```
p_ <- GGally::print_if_interactive

pm <- ggpairs(iris, 1:2, progress = ggmatrix_progress())
p_(pm)

# does not clear after finishing
pm <- ggpairs(iris, 1:2, progress = ggmatrix_progress(clear = FALSE))
p_(pm)
```

ggnet *Network plot*

Description

Function for plotting network objects using **ggplot2**, now replaced by the ggnet2 function, which provides additional control over plotting parameters. Please visit https://github.com/briatte/ggnet for the latest version of ggnet2, and https://briatte.github.io/ggnet/ for a vignette that contains many examples and explanations.

Usage

```
ggnet(
  net,
  mode = "fruchtermanreingold",
  layout.par = NULL,
  layout.exp = 0,
  size = 9,
  alpha = 1,
  weight = "none",
  weight.legend = NA,
  weight.method = weight,
  weight.min = NA,
  weight.max = NA,
  weight.cut = FALSE,
  group = NULL,
  group.legend = NA,
  node.group = group,
  node.color = NULL,
  node.alpha = alpha,
  segment.alpha = alpha,
  segment.color = "grey50",
  segment.label = NULL,
  segment.size = 0.25,
  arrow.size = 0,
  arrow.gap = 0,
  arrow.type = "closed",
  label = FALSE,
  label.nodes = label,
  label.size = size/2,
  label.trim = FALSE,
  legend.size = 9,
  legend.position = "right",
  names = c("", ""),
  quantize.weights = FALSE,
  subset.threshold = 0,
  top8.nodes = FALSE,
  trim.labels = FALSE,
  ...
)
```

Arguments

net	an object of class network, or any object that can be coerced to this class, such as an adjacency or incidence matrix, or an edge list: see edgeset.constructors and network for details. If the object is of class igraph and the intergraph package is installed, it will be used to convert the object: see asNetwork for details.
mode	a placement method from those provided in the sna package: see gplot.layout for details. Also accepts the names of two numeric vertex attributes of net, or a matrix of numeric coordinates, in which case the first two columns of the matrix are used. Defaults to the Fruchterman-Reingold force-directed algorithm.
layout.par	options to be passed to the placement method, as listed in gplot.layout. Defaults to NULL.
layout.exp	a multiplier to expand the horizontal axis if node labels get clipped: see expand_range for details. Defaults to 0 (no expansion).
size	size of the network nodes. If the nodes are weighted, their area is proportionally scaled up to the size set by size. Defaults to 9.
alpha	a level of transparency for nodes, vertices and arrows. Defaults to 1.
weight	the weighting method for the nodes, which might be a vertex attribute or a vector of size values. Also accepts "indegree", "outdegree", "degree" or "freeman" to size the nodes by their unweighted degree centrality ("degree" and "freeman" are equivalent): see degree for details. All node weights must be positive. Defaults to "none" (no weighting).
weight.legend	the name to assign to the legend created by weight. Defaults to NA (no name).
weight.method	see weight
weight.min	whether to subset the network to nodes with a minimum size, based on the values of weight. Defaults to NA (preserves all nodes).
weight.max	whether to subset the network to nodes with a maximum size, based on the values of weight. Defaults to NA (preserves all nodes).
weight.cut	whether to cut the size of the nodes into a certain number of quantiles. Accepts TRUE, which tries to cut the sizes into quartiles, or any positive numeric value, which tries to cut the sizes into that many quantiles. If the size of the nodes do not contain the specified number of distinct quantiles, the largest possible number is used. See quantile and cut for details. Defaults to FALSE (does nothing).
group	the groups of the nodes, either as a vector of values or as a vertex attribute. If set to mode on a bipartite network, the nodes will be grouped as "actor" if they belong to the primary mode and "event" if they belong to the secondary mode.
group.legend	the name to assign to the legend created by group.
node.group	see group
node.color	a vector of character strings to color the nodes with, holding as many colors as there are levels in node.group. Defaults to NULL, which will assign grayscale colors to each group.
node.alpha	transparency of the nodes. Inherits from alpha.
segment.alpha	the level of transparency of the edges. Defaults to alpha, which defaults to 1.
segment.color	the color of the edges, as a color value, a vector of color values, or as an edge attribute containing color values. Defaults to "grey50".
segment.label	the labels to plot at the middle of the edges, as a single value, a vector of values, or as an edge attribute. Defaults to NULL (no edge labels).

segment.size — the size of the edges, in points, as a single numeric value, a vector of values, or as an edge attribute. Defaults to 0.25.

arrow.size — the size of the arrows for directed network edges, in points. See arrow for details. Defaults to 0 (no arrows).

arrow.gap — a setting aimed at improving the display of edge arrows by plotting slightly shorter edges. Accepts any value between 0 and 1, where a value of 0.05 will generally achieve good results when the size of the nodes is reasonably small. Defaults to 0 (no shortening).

arrow.type — the type of the arrows for directed network edges. See arrow for details. Defaults to "closed".

label — whether to label the nodes. If set to TRUE, nodes are labeled with their vertex names. If set to a vector that contains as many elements as there are nodes in net, nodes are labeled with these. If set to any other vector of values, the nodes are labeled only when their vertex name matches one of these values. Defaults to FALSE (no labels).

label.nodes — see label

label.size — the size of the node labels, in points, as a numeric value, a vector of numeric values, or as a vertex attribute containing numeric values. Defaults to size / 2 (half the maximum node size), which defaults to 6.

label.trim — whether to apply some trimming to the node labels. Accepts any function that can process a character vector, or a strictly positive numeric value, in which case the labels are trimmed to a fixed-length substring of that length: see substr for details. Defaults to FALSE (does nothing).

legend.size — the size of the legend symbols and text, in points. Defaults to 9.

legend.position — the location of the plot legend(s). Accepts all legend.position values supported by theme. Defaults to "right".

names — deprecated: see group.legend and size.legend

quantize.weights — deprecated: see weight.cut

subset.threshold — deprecated: see weight.min

top8.nodes — deprecated: this functionality was experimental and has been removed entirely from ggnet

trim.labels — deprecated: see label.trim

... — other arguments passed to the geom_text object that sets the node labels: see geom_text for details.

Details

The degree centrality measures that can be produced through the weight argument will take the directedness of the network into account, but will be unweighted. To compute weighted network measures, see the tnet package by Tore Opsahl (help("tnet",package = "tnet")).

Author(s)

Moritz Marbach and Francois Briatte, with help from Heike Hofmann, Pedro Jordano and Ming-Yu Liu

See Also

ggnet2 in this package, gplot in the sna package, and plot.network in the network package

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

library(network)

# random adjacency matrix
x           <- 10
ndyads      <- x * (x - 1)
density     <- x / ndyads
m           <- matrix(0, nrow = x, ncol = x)
dimnames(m) <- list(letters[ 1:x ], letters[ 1:x ])
m[ row(m) != col(m) ] <- runif(ndyads) < density
m

# random undirected network
n <- network::network(m, directed = FALSE)
n

ggnet(n, label = TRUE, alpha = 1, color = "white", segment.color = "black")

# random groups
g <- sample(letters[ 1:3 ], 10, replace = TRUE)
g

# color palette
p <- c("a" = "steelblue", "b" = "forestgreen", "c" = "tomato")

p_(ggnet(n, node.group = g, node.color = p, label = TRUE, color = "white"))

# edge arrows on a directed network
p_(ggnet(network(m, directed = TRUE), arrow.gap = 0.05, arrow.size = 10))
```

---

ggnet2 *Network plot*

Description

Function for plotting network objects using **ggplot2**, with additional control over graphical parameters that are not supported by the ggnet function. Please visit https://github.com/briatte/ggnet for the latest version of ggnet2, and https://briatte.github.io/ggnet/ for a vignette that contains many examples and explanations.

Usage

```
ggnet2(
  net,
  mode = "fruchtermanreingold",
  layout.par = NULL,
```

```
  layout.exp = 0,
  alpha = 1,
  color = "grey75",
  shape = 19,
  size = 9,
  max_size = 9,
  na.rm = NA,
  palette = NULL,
  alpha.palette = NULL,
  alpha.legend = NA,
  color.palette = palette,
  color.legend = NA,
  shape.palette = NULL,
  shape.legend = NA,
  size.palette = NULL,
  size.legend = NA,
  size.zero = FALSE,
  size.cut = FALSE,
  size.min = NA,
  size.max = NA,
  label = FALSE,
  label.alpha = 1,
  label.color = "black",
  label.size = max_size/2,
  label.trim = FALSE,
  node.alpha = alpha,
  node.color = color,
  node.label = label,
  node.shape = shape,
  node.size = size,
  edge.alpha = 1,
  edge.color = "grey50",
  edge.lty = "solid",
  edge.size = 0.25,
  edge.label = NULL,
  edge.label.alpha = 1,
  edge.label.color = label.color,
  edge.label.fill = "white",
  edge.label.size = max_size/2,
  arrow.size = 0,
  arrow.gap = 0,
  arrow.type = "closed",
  legend.size = 9,
  legend.position = "right",
  ...
)
```

Arguments

net	an object of class network, or any object that can be coerced to this class, such as an adjacency or incidence matrix, or an edge list: see edgeset.constructors and network for details. If the object is of class igraph and the intergraph package is installed, it will be used to convert the object: see asNetwork for details.

mode	a placement method from those provided in the sna package: see gplot.layout for details. Also accepts the names of two numeric vertex attributes of net, or a matrix of numeric coordinates, in which case the first two columns of the matrix are used. Defaults to the Fruchterman-Reingold force-directed algorithm.
layout.par	options to be passed to the placement method, as listed in gplot.layout. Defaults to NULL.
layout.exp	a multiplier to expand the horizontal axis if node labels get clipped: see expand_range for details. Defaults to 0 (no expansion).
alpha	the level of transparency of the edges and nodes, which might be a single value, a vertex attribute, or a vector of values. Also accepts "mode" on bipartite networks (see 'Details'). Defaults to 1 (no transparency).
color	the color of the nodes, which might be a single value, a vertex attribute, or a vector of values. Also accepts "mode" on bipartite networks (see 'Details'). Defaults to grey75.
shape	the shape of the nodes, which might be a single value, a vertex attribute, or a vector of values. Also accepts "mode" on bipartite networks (see 'Details'). Defaults to 19 (solid circle).
size	the size of the nodes, in points, which might be a single value, a vertex attribute, or a vector of values. Also accepts "indegree", "outdegree", "degree" or "freeman" to size the nodes by their unweighted degree centrality ("degree" and "freeman" are equivalent): see degree for details. All node sizes must be strictly positive. Also accepts "mode" on bipartite networks (see 'Details'). Defaults to 9.
max_size	the *maximum* size of the node when size produces nodes of different sizes, in points. Defaults to 9.
na.rm	whether to subset the network to nodes that are *not* missing a given vertex attribute. If set to any vertex attribute of net, the nodes for which this attribute is NA will be removed. Defaults to NA (does nothing).
palette	the palette to color the nodes, when color is not a color value or a vector of color values. Accepts named vectors of color values, or if RColorBrewer is installed, any ColorBrewer palette name: see RColorBrewer::brewer.pal() and https://colorbrewer2.org/ for details. Defaults to NULL, which will create an array of grayscale color values if color is not a color value or a vector of color values.
alpha.palette	the palette to control the transparency levels of the nodes set by alpha when the levels are not numeric values. Defaults to NULL, which will create an array of alpha transparency values if alpha is not a numeric value or a vector of numeric values.
alpha.legend	the name to assign to the legend created by alpha when its levels are not numeric values. Defaults to NA (no name).
color.palette	see palette
color.legend	the name to assign to the legend created by palette. Defaults to NA (no name).
shape.palette	the palette to control the shapes of the nodes set by shape when the shapes are not numeric values. Defaults to NULL, which will create an array of shape values if shape is not a numeric value or a vector of numeric values.
shape.legend	the name to assign to the legend created by shape when its levels are not numeric values. Defaults to NA (no name).

size.palette — the palette to control the sizes of the nodes set by size when the sizes are not numeric values.

size.legend — the name to assign to the legend created by size. Defaults to NA (no name).

size.zero — whether to accept zero-sized nodes based on the value(s) of size. Defaults to FALSE, which ensures that zero-sized nodes are still shown in the plot and its size legend.

size.cut — whether to cut the size of the nodes into a certain number of quantiles. Accepts TRUE, which tries to cut the sizes into quartiles, or any positive numeric value, which tries to cut the sizes into that many quantiles. If the size of the nodes do not contain the specified number of distinct quantiles, the largest possible number is used. See quantile and cut for details. Defaults to FALSE (does nothing).

size.min — whether to subset the network to nodes with a minimum size, based on the values of size. Defaults to NA (preserves all nodes).

size.max — whether to subset the network to nodes with a maximum size, based on the values of size. Defaults to NA (preserves all nodes).

label — whether to label the nodes. If set to TRUE, nodes are labeled with their vertex names. If set to a vector that contains as many elements as there are nodes in net, nodes are labeled with these. If set to any other vector of values, the nodes are labeled only when their vertex name matches one of these values. Defaults to FALSE (no labels).

label.alpha — the level of transparency of the node labels, as a numeric value, a vector of numeric values, or as a vertex attribute containing numeric values. Defaults to 1 (no transparency).

label.color — the color of the node labels, as a color value, a vector of color values, or as a vertex attribute containing color values. Defaults to "black".

label.size — the size of the node labels, in points, as a numeric value, a vector of numeric values, or as a vertex attribute containing numeric values. Defaults to max_size / 2 (half the maximum node size), which defaults to 4.5.

label.trim — whether to apply some trimming to the node labels. Accepts any function that can process a character vector, or a strictly positive numeric value, in which case the labels are trimmed to a fixed-length substring of that length: see substr for details. Defaults to FALSE (does nothing).

node.alpha — see alpha

node.color — see color

node.label — see label

node.shape — see shape

node.size — see size

edge.alpha — the level of transparency of the edges. Defaults to the value of alpha, which defaults to 1.

edge.color — the color of the edges, as a color value, a vector of color values, or as an edge attribute containing color values. Defaults to "grey50".

edge.lty — the linetype of the edges, as a linetype value, a vector of linetype values, or as an edge attribute containing linetype values. Defaults to "solid".

edge.size — the size of the edges, in points, as a numeric value, a vector of numeric values, or as an edge attribute containing numeric values. All edge sizes must be strictly positive. Defaults to 0.25.

edge.label — the labels to plot at the middle of the edges, as a single value, a vector of values, or as an edge attribute. Defaults to NULL (no edge labels).

edge.label.alpha — the level of transparency of the edge labels, as a numeric value, a vector of numeric values, or as an edge attribute containing numeric values. Defaults to 1 (no transparency).

edge.label.color — the color of the edge labels, as a color value, a vector of color values, or as an edge attribute containing color values. Defaults to label.color, which defaults to "black".

edge.label.fill — the background color of the edge labels. Defaults to "white".

edge.label.size — the size of the edge labels, in points, as a numeric value, a vector of numeric values, or as an edge attribute containing numeric values. All edge label sizes must be strictly positive. Defaults to max_size / 2 (half the maximum node size), which defaults to 4.5.

arrow.size — the size of the arrows for directed network edges, in points. See arrow for details. Defaults to 0 (no arrows).

arrow.gap — a setting aimed at improving the display of edge arrows by plotting slightly shorter edges. Accepts any value between 0 and 1, where a value of 0.05 will generally achieve good results when the size of the nodes is reasonably small. Defaults to 0 (no shortening).

arrow.type — the type of the arrows for directed network edges. See arrow for details. Defaults to "closed".

legend.size — the size of the legend symbols and text, in points. Defaults to 9.

legend.position — the location of the plot legend(s). Accepts all legend.position values supported by theme. Defaults to "right".

... — other arguments passed to the geom_text object that sets the node labels: see geom_text for details.

Details

The degree centrality measures that can be produced through the size argument will take the directedness of the network into account, but will be unweighted. To compute weighted network measures, see the tnet package by Tore Opsahl (help("tnet",package = "tnet")).

The nodes of bipartite networks can be mapped to their mode by passing the "mode" argument to any of alpha, color, shape and size, in which case the nodes of the primary mode will be mapped as "actor", and the nodes of the secondary mode will be mapped as "event".

Author(s)

Moritz Marbach and Francois Briatte, with help from Heike Hofmann, Pedro Jordano and Ming-Yu Liu

See Also

ggnet in this package, gplot in the sna package, and plot.network in the network package

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

library(network)

# random adjacency matrix
x           <- 10
ndyads      <- x * (x - 1)
density     <- x / ndyads
m           <- matrix(0, nrow = x, ncol = x)
dimnames(m) <- list(letters[ 1:x ], letters[ 1:x ])
m[ row(m) != col(m) ] <- runif(ndyads) < density
m

# random undirected network
n <- network::network(m, directed = FALSE)
n

p_(ggnet2(n, label = TRUE))
p_(ggnet2(n, label = TRUE, shape = 15))
p_(ggnet2(n, label = TRUE, shape = 15, color = "black", label.color = "white"))

# add vertex attribute
x = network.vertex.names(n)
x = ifelse(x %in% c("a", "e", "i"), "vowel", "consonant")
n %v% "phono" = x

p_(ggnet2(n, color = "phono"))
p_(ggnet2(n, color = "phono", palette = c("vowel" = "gold", "consonant" = "grey")))
p_(ggnet2(n, shape = "phono", color = "phono"))

if (require(RColorBrewer)) {

  # random groups
  n %v% "group" <- sample(LETTERS[1:3], 10, replace = TRUE)

  p_(ggnet2(n, color = "group", palette = "Set2"))

}

# random weights
n %e% "weight" <- sample(1:3, network.edgecount(n), replace = TRUE)
p_(ggnet2(n, edge.size = "weight", edge.label = "weight"))

# edge arrows on a directed network
p_(ggnet2(network(m, directed = TRUE), arrow.gap = 0.05, arrow.size = 10))

# Padgett's Florentine wedding data
data(flo, package = "network")
flo

p_(ggnet2(flo, label = TRUE))
p_(ggnet2(flo, label = TRUE, label.trim = 4, vjust = -1, size = 3, color = 1))
p_(ggnet2(flo, label = TRUE, size = 12, color = "white"))
```

ggnetworkmap *Network plot map overlay*

Description

Plots a network with **ggplot2** suitable for overlay on a **ggmap** plot or **ggplot2**

Usage

```
ggnetworkmap(
  gg,
  net,
  size = 3,
  alpha = 0.75,
  weight,
  node.group,
  node.color = NULL,
  node.alpha = NULL,
  ring.group,
  segment.alpha = NULL,
  segment.color = "grey",
  great.circles = FALSE,
  segment.size = 0.25,
  arrow.size = 0,
  label.nodes = FALSE,
  label.size = size/2,
  ...
)
```

Arguments

gg	an object of class ggplot.
net	an object of class network, or any object that can be coerced to this class, such as an adjacency or incidence matrix, or an edge list: see edgeset.constructors and network for details. If the object is of class igraph and the intergraph package is installed, it will be used to convert the object: see asNetwork for details.
size	size of the network nodes. Defaults to 3. If the nodes are weighted, their area is proportionally scaled up to the size set by size.
alpha	a level of transparency for nodes, vertices and arrows. Defaults to 0.75.
weight	if present, the unquoted name of a vertex attribute in data. Otherwise nodes are unweighted.
node.group	NULL, the default, or the unquoted name of a vertex attribute that will be used to determine the color of each node.
node.color	If node.group is null, a character string specifying a color.
node.alpha	transparency of the nodes. Inherits from alpha.
ring.group	if not NULL, the default, the unquoted name of a vertex attribute that will be used to determine the color of each node border.
segment.alpha	transparency of the vertex links. Inherits from alpha

segment.color	color of the vertex links. Defaults to "grey".
great.circles	whether to draw edges as great circles using the geosphere package. Defaults to FALSE
segment.size	size of the vertex links, as a vector of values or as a single value. Defaults to 0.25.
arrow.size	size of the vertex arrows for directed network plotting, in centimeters. Defaults to 0.
label.nodes	label nodes with their vertex names attribute. If set to TRUE, all nodes are labelled. Also accepts a vector of character strings to match with vertex names.
label.size	size of the labels. Defaults to size / 2.
...	other arguments supplied to geom_text for the node labels. Arguments pertaining to the title or other items can be achieved through **ggplot2** methods.

Details

This is a descendant of the original ggnet function. ggnet added the innovation of plotting the network geographically. However, ggnet needed to be the first object in the ggplot chain. ggnetworkmap does not. If passed a ggplot object as its first argument, such as output from ggmap, ggnetworkmap will plot on top of that chart, looking for vertex attributes lon and lat as coordinates. Otherwise, ggnetworkmap will generate coordinates using the Fruchterman-Reingold algorithm.

This is a function for plotting graphs generated by network or igraph in a more flexible and elegant manner than permitted by ggnet. The function does not need to be the first plot in the ggplot chain, so the graph can be plotted on top of a map or other chart. Segments can be straight lines, or plotted as great circles. Note that the great circles feature can produce odd results with arrows and with vertices beyond the plot edges; this is a **ggplot2** limitation and cannot yet be fixed. Nodes can have two color schemes, which are then plotted as the center and ring around the node. The color schemes are selected by adding scale_fill_ or scale_color_ just like any other **ggplot2** plot. If there are no rings, scale_color sets the color of the nodes. If there are rings, scale_color sets the color of the rings, and scale_fill sets the color of the centers. Note that additional arguments in the ... are passed to geom_text for plotting labels.

Author(s)

Amos Elberg. Original by Moritz Marbach, Francois Briatte

Examples

```
# small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

invisible(lapply(c("ggplot2", "maps", "network", "sna"), base::library, character.only = TRUE))

## Example showing great circles on a simple map of the USA
## http://flowingdata.com/2011/05/11/how-to-map-connections-with-great-circles/

airports <- read.csv("http://datasets.flowingdata.com/tuts/maparcs/airports.csv", header = TRUE)
rownames(airports) <- airports$iata

# select some random flights
set.seed(1234)
flights <- data.frame(
  origin = sample(airports[200:400, ]$iata, 200, replace = TRUE),
```

```
  destination = sample(airports[200:400, ]$iata, 200, replace = TRUE)
)

# convert to network
flights <- network(flights, directed = TRUE)

# add geographic coordinates
flights %v% "lat" <- airports[ network.vertex.names(flights), "lat" ]
flights %v% "lon" <- airports[ network.vertex.names(flights), "long" ]

# drop isolated airports
delete.vertices(flights, which(degree(flights) < 2))

# compute degree centrality
flights %v% "degree" <- degree(flights, gmode = "digraph")

# add random groups
flights %v% "mygroup" <- sample(letters[1:4], network.size(flights), replace = TRUE)

# create a map of the USA
usa <- ggplot(map_data("usa"), aes(x = long, y = lat)) +
  geom_polygon(aes(group = group), color = "grey65",
               fill = "#f9f9f9", size = 0.2)

# overlay network data to map
p <- ggnetworkmap(
  usa, flights, size = 4, great.circles = TRUE,
  node.group = mygroup, segment.color = "steelblue",
  ring.group = degree, weight = degree
)
p_(p)

## Exploring a community of spambots found on Twitter
## Data by Amos Elberg: see ?twitter_spambots for details

data(twitter_spambots)

# create a world map
world <- fortify(map("world", plot = FALSE, fill = TRUE))
world <- ggplot(world, aes(x = long, y = lat)) +
  geom_polygon(aes(group = group), color = "grey65",
               fill = "#f9f9f9", size = 0.2)

# view global structure
p <- ggnetworkmap(world, twitter_spambots)
p_(p)

# domestic distribution
p <- ggnetworkmap(net = twitter_spambots)
p_(p)

# topology
p <- ggnetworkmap(net = twitter_spambots, arrow.size = 0.5)
p_(p)

# compute indegree and outdegree centrality
twitter_spambots %v% "indegree" <- degree(twitter_spambots, cmode = "indegree")
```

```
twitter_spambots %v% "outdegree" <- degree(twitter_spambots, cmode = "outdegree")

p <- ggnetworkmap(
  net = twitter_spambots,
  arrow.size = 0.5,
  node.group = indegree,
  ring.group = outdegree, size = 4
) +
  scale_fill_continuous("Indegree", high = "red", low = "yellow") +
  labs(color = "Outdegree")
p_(p)

# show some vertex attributes associated with each account
p <- ggnetworkmap(
  net = twitter_spambots,
  arrow.size = 0.5,
  node.group = followers,
  ring.group = friends,
  size = 4,
  weight = indegree,
  label.nodes = TRUE, vjust = -1.5
) +
  scale_fill_continuous("Followers", high = "red", low = "yellow") +
  labs(color = "Friends") +
  scale_color_continuous(low = "lightgreen", high = "darkgreen")
p_(p)
```

---

ggnostic *Plot matrix of statistical model diagnostics*

Description

Plot matrix of statistical model diagnostics

Usage

```
ggnostic(
  model,
  ...,
  columnsX = attr(data, "var_x"),
  columnsY = c(".resid", ".sigma", ".hat", ".cooksd"),
  columnLabelsX = attr(data, "var_x_label"),
  columnLabelsY = gsub("\\.", " ", gsub("^\\.", "", columnsY)),
  xlab = "explanatory variables",
  ylab = "diagnostics",
  title = paste(deparse(model$call, width.cutoff = 500L), collapse = "\n"),
  continuous = list(default = ggally_points, .fitted = ggally_points, .se.fit =
   ggally_nostic_se_fit, .resid = ggally_nostic_resid, .hat = ggally_nostic_hat, .sigma
    = ggally_nostic_sigma, .cooksd = ggally_nostic_cooksd, .std.resid =
    ggally_nostic_std_resid),
  combo = list(default = ggally_box_no_facet, fitted = ggally_box_no_facet, .se.fit =
```

```
    ggally_nostic_se_fit, .resid = ggally_nostic_resid, .hat = ggally_nostic_hat, .sigma
      = ggally_nostic_sigma, .cooksd = ggally_nostic_cooksd, .std.resid =
      ggally_nostic_std_resid),
    discrete = list(default = ggally_ratio, .fitted = ggally_ratio, .se.fit =
    ggally_ratio, .resid = ggally_ratio, .hat = ggally_ratio, .sigma = ggally_ratio,
      .cooksd = ggally_ratio, .std.resid = ggally_ratio),
    progress = NULL,
    data = broomify(model)
  )
```

Arguments

`model`	statistical model object such as output from `stats::lm` or `stats::glm`
`...`	arguments passed directly to ggduo
`columnsX`	columns to be displayed in the plot matrix. Defaults to the predictor columns of the `model`
`columnsY`	rows to be displayed in the plot matrix. Defaults to residuals, leave one out sigma value, diagonal of the hat matrix, and Cook's Distance. The possible values are the response variables in the model and the added columns provided by `broom::augment()`. See details for more information.
`columnLabelsX, columnLabelsY`	column and row labels to display in the plot matrix
`xlab, ylab, title`	plot matrix labels passed directly to ggmatrix
`continuous, combo, discrete`	list of functions for each y variable. See details for more information.
`progress`	`NULL` (default) for a progress bar in interactive sessions with more than 15 plots, `TRUE` for a progress bar, `FALSE` for no progress bar, or a function that accepts at least a plot matrix and returns a new `progress::progress_bar`. See ggmatrix_progress.
`data`	data defaults to a 'broomify'ed model object. This object will contain information about the X variables, Y variables, and multiple broom outputs. See `broomify(model)` for more information

`columnsY`

`broom::augment()` collects data from the supplied model and returns a data.frame with the following columns (taken directly from broom documentation). These columns are the only allowed values in the `columnsY` parameter to ggnostic.

.resid Residuals

.hat Diagonal of the hat matrix

.sigma Estimate of residual standard deviation when corresponding observation is dropped from model

.cooksd Cooks distance, `stats::cooks.distance()`

.fitted Fitted values of model

.se.fit Standard errors of fitted values

.std.resid Standardized residuals

response variable name The response variable in the model may be added. Such as `"mpg"` in the model `lm(mpg ~ ., data = mtcars)`

`continuous`, `combo`, `discrete` **types**

Similar to ggduo and ggpairs, functions may be supplied to display the different column types. However, since the Y rows are fixed, each row has it's own corresponding function in each of the plot types: continuous, combo, and discrete. Each plot type list can have keys that correspond to the `broom::augment()` output: `".fitted"`, `".resid"`, `".std.resid"`, `".sigma"`, `".se.fit"`, `".hat"`, `".cooksd"`. An extra key, `"default"`, is used to plot the response variables of the model if they are included. Having a function for each diagnostic allows for very fine control over the diagnostics plot matrix. The functions for each type list are wrapped into a switch function that calls the function corresponding to the y variable being plotted. These switch functions are then passed directly to the `types` parameter in ggduo.

Examples

```
# small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive
data(mtcars)

# use mtcars dataset and alter the 'am' column to display actual name values
mtc <- mtcars
mtc$am <- c("0" = "automatic", "1" = "manual")[as.character(mtc$am)]

# step the complete model down to a smaller model
mod <- stats::step(stats::lm(mpg ~ ., data = mtc), trace = FALSE)

# display using defaults
pm <- ggnostic(mod)
p_(pm)

# color by am value
pm <- ggnostic(mod, mapping = ggplot2::aes(color = am))
p_(pm)

# turn resid smooth error ribbon off
pm <- ggnostic(mod, continuous = list(.resid = wrap("nostic_resid", se = FALSE)))
p_(pm)


## plot residuals vs fitted in a ggpairs plot matrix
dt <- broomify(mod)
pm <- ggpairs(
  dt, c(".fitted", ".resid"),
  columnLabels = c("fitted", "residuals"),
  lower = list(continuous = ggally_nostic_resid)
)
p_(pm)
```

---

ggpairs *ggplot2 generalized pairs plot*

Description

Make a matrix of plots with a given data set

Usage

```
ggpairs(
  data,
  mapping = NULL,
  columns = 1:ncol(data),
  title = NULL,
  upper = list(continuous = "cor", combo = "box_no_facet", discrete = "count", na =
    "na"),
  lower = list(continuous = "points", combo = "facethist", discrete = "facetbar", na =
    "na"),
  diag = list(continuous = "densityDiag", discrete = "barDiag", na = "naDiag"),
  params = NULL,
  ...,
  xlab = NULL,
  ylab = NULL,
  axisLabels = c("show", "internal", "none"),
  columnLabels = colnames(data[columns]),
  labeller = "label_value",
  switch = NULL,
  showStrips = NULL,
  legend = NULL,
  cardinality_threshold = 15,
  progress = NULL,
  proportions = NULL,
  legends = stop("deprecated")
)
```

Arguments

data	data set using. Can have both numerical and categorical data.
mapping	aesthetic mapping (besides x and y). See aes(). If mapping is numeric, columns will be set to the mapping value and mapping will be set to NULL.
columns	which columns are used to make plots. Defaults to all columns.
title, xlab, ylab	title, x label, and y label for the graph
upper	see Details
lower	see Details
diag	see Details
params	deprecated. Please see wrap_fn_with_param_arg
...	deprecated. Please use mapping
axisLabels	either "show" to display axisLabels, "internal" for labels in the diagonal plots, or "none" for no axis labels
columnLabels	label names to be displayed. Defaults to names of columns being used.
labeller	labeller for facets. See labellers. Common values are "label_value" (default) and "label_parsed".
switch	switch parameter for facet_grid. See ggplot2::facet_grid. By default, the labels are displayed on the top and right of the plot. If "x", the top labels will be displayed to the bottom. If "y", the right-hand side labels will be displayed to the left. Can also be set to "both"

showStrips — boolean to determine if each plot's strips should be displayed. NULL will default to the top and right side plots only. TRUE or FALSE will turn all strips on or off respectively.

legend — May be the two objects described below or the default NULL value. The legend position can be moved by using ggplot2's theme element pm + theme(legend.position = "bottom")

a numeric vector of length 2 provides the location of the plot to use the legend for the plot matrix's legend. Such as legend = c(3,5) which will use the legend from the plot in the third row and fifth column

a single numeric value provides the location of a plot according to the display order. Such as legend = 3 in a plot matrix with 2 rows and 5 columns displayed by column will return the plot in position c(1,2)

a object from grab_legend() a predetermined plot legend that will be displayed directly

cardinality_threshold — maximum number of levels allowed in a character / factor column. Set this value to NULL to not check factor columns. Defaults to 15

progress — NULL (default) for a progress bar in interactive sessions with more than 15 plots, TRUE for a progress bar, FALSE for no progress bar, or a function that accepts at least a plot matrix and returns a new progress::progress_bar. See ggmatrix_progress.

proportions — Value to change how much area is given for each plot. Either NULL (default), numeric value matching respective length, grid::unit object with matching respective length or "auto" for automatic relative proportions based on the number of levels for categorical variables.

legends — deprecated

Details

upper and lower are lists that may contain the variables 'continuous', 'combo', 'discrete', and 'na'. Each element of the list may be a function or a string. If a string is supplied, it must be a character string representing the tail end of a ggally_NAME function. The list of current valid ggally_NAME functions is visible in a dedicated vignette.

continuous This option is used for continuous X and Y data.

combo This option is used for either continuous X and categorical Y data or categorical X and continuous Y data.

discrete This option is used for categorical X and Y data.

na This option is used when all X data is NA, all Y data is NA, or either all X or Y data is NA.

diag is a list that may only contain the variables 'continuous', 'discrete', and 'na'. Each element of the diag list is a string implementing the following options:

continuous exactly one of ('densityDiag', 'barDiag', 'blankDiag'). This option is used for continuous X data.

discrete exactly one of ('barDiag', 'blankDiag'). This option is used for categorical X and Y data.

na exactly one of ('naDiag', 'blankDiag'). This option is used when all X data is NA.

If 'blank' is ever chosen as an option, then ggpairs will produce an empty plot.

If a function is supplied as an option to upper, lower, or diag, it should implement the function api of function(data,mapping,...){#make ggplot2 plot}. If a specific function needs its parameters set, wrap(fn,param1 = val1,param2 = val2) the function with its parameters.

Value

ggmatrix object that if called, will print

Author(s)

Barret Schloerke, Jason Crowley, Di Cook, Heike Hofmann, Hadley Wickham

References

John W Emerson, Walton A Green, Barret Schloerke, Jason Crowley, Dianne Cook, Heike Hofmann, Hadley Wickham. The Generalized Pairs Plot. Journal of Computational and Graphical Statistics, vol. 22, no. 1, pp. 79-91, 2012.

See Also

wrap v1_ggmatrix_theme

Examples

```
# small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive


## Quick example, with and without colour
data(flea)
ggpairs(flea, columns = 2:4)
pm <- ggpairs(flea, columns = 2:4, ggplot2::aes(colour=species))
p_(pm)
# Note: colour should be categorical, else you will need to reset
# the upper triangle to use points instead of trying to compute corr

data(tips, package = "reshape")
pm <- ggpairs(tips[, 1:3])
p_(pm)
pm <- ggpairs(tips, 1:3, columnLabels = c("Total Bill", "Tip", "Sex"))
p_(pm)
pm <- ggpairs(tips, upper = "blank")
p_(pm)

## Plot Types
# Change default plot behavior
pm <- ggpairs(
  tips[, c(1, 3, 4, 2)],
  upper = list(continuous = "density", combo = "box_no_facet"),
  lower = list(continuous = "points", combo = "dot_no_facet")
)
p_(pm)
# Supply Raw Functions (may be user defined functions!)
pm <- ggpairs(
  tips[, c(1, 3, 4, 2)],
  upper = list(continuous = ggally_density, combo = ggally_box_no_facet),
  lower = list(continuous = ggally_points, combo = ggally_dot_no_facet)
)
p_(pm)

# Use sample of the diamonds data
```

```
data(diamonds, package="ggplot2")
diamonds.samp <- diamonds[sample(1:dim(diamonds)[1], 1000), ]

# Different aesthetics for different plot sections and plot types
pm <- ggpairs(
 diamonds.samp[, 1:5],
 mapping = ggplot2::aes(color = cut),
 upper = list(continuous = wrap("density", alpha = 0.5), combo = "box_no_facet"),
 lower = list(continuous = wrap("points", alpha = 0.3), combo = wrap("dot_no_facet", alpha = 0.4)),
 title = "Diamonds"
)
p_(pm)

## Axis Label Variations
# Only Variable Labels on the diagonal (no axis labels)
pm <- ggpairs(tips[, 1:3], axisLabels="internal")
p_(pm)
# Only Variable Labels on the outside (no axis labels)
pm <- ggpairs(tips[, 1:3], axisLabels="none")
p_(pm)

## Facet Label Variations
#  Default:
df_x <- rnorm(100)
df_y <- df_x + rnorm(100, 0, 0.1)
df <- data.frame(x = df_x, y = df_y, c = sqrt(df_x^2 + df_y^2))
pm <- ggpairs(
  df,
  columnLabels = c("alpha[foo]", "alpha[bar]", "sqrt(alpha[foo]^2 + alpha[bar]^2)")
)
p_(pm)
#  Parsed labels:
pm <- ggpairs(
  df,
  columnLabels = c("alpha[foo]", "alpha[bar]", "sqrt(alpha[foo]^2 + alpha[bar]^2)"),
  labeller = "label_parsed"
)
p_(pm)

## Plot Insertion Example
custom_car <- ggpairs(mtcars[, c("mpg", "wt", "cyl")], upper = "blank", title = "Custom Example")
# ggplot example taken from example(geom_text)
  plot <- ggplot2::ggplot(mtcars, ggplot2::aes(x=wt, y=mpg, label=rownames(mtcars)))
  plot <- plot +
    ggplot2::geom_text(ggplot2::aes(colour=factor(cyl)), size = 3) +
    ggplot2::scale_colour_discrete(l=40)
custom_car[1, 2] <- plot
personal_plot <- ggally_text(
  "ggpairs allows you\nto put in your\nown plot.\nLike that one.\n <---"
)
custom_car[1, 3] <- personal_plot
p_(custom_car)

## Remove binwidth warning from ggplot2
# displays warning about picking a better binwidth
pm <- ggpairs(tips, 2:3)
p_(pm)
```

```
# no warning displayed
pm <- ggpairs(tips, 2:3, lower = list(combo = wrap("facethist", binwidth = 0.5)))
p_(pm)
# no warning displayed with user supplied function
pm <- ggpairs(tips, 2:3, lower = list(combo = wrap(ggally_facethist, binwidth = 0.5)))
p_(pm)

## Remove panel grid lines from correlation plots
pm <- ggpairs(
  flea, columns = 2:4,
  upper = list(continuous = wrap(ggally_cor, displayGrid = FALSE))
)
p_(pm)

## Custom with/height of subplots
pm <- ggpairs(tips, columns = c(2, 3, 5))
p_(pm)

pm <- ggpairs(tips, columns = c(2, 3, 5), proportions = "auto")
p_(pm)

pm <- ggpairs(tips, columns = c(2, 3, 5), proportions = c(1, 3, 2))
p_(pm)
```

---

ggparcoord *Parallel coordinate plot*

Description

A function for plotting static parallel coordinate plots, utilizing the ggplot2 graphics package.

Usage

```
ggparcoord(
  data,
  columns = 1:ncol(data),
  groupColumn = NULL,
  scale = "std",
  scaleSummary = "mean",
  centerObsID = 1,
  missing = "exclude",
  order = columns,
  showPoints = FALSE,
  splineFactor = FALSE,
  alphaLines = 1,
  boxplot = FALSE,
  shadeBox = NULL,
  mapping = NULL,
  title = ""
)
```

Arguments

data	the dataset to plot
columns	a vector of variables (either names or indices) to be axes in the plot
groupColumn	a single variable to group (color) by
scale	method used to scale the variables (see Details)
scaleSummary	if scale=="center", summary statistic to univariately center each variable by
centerObsID	if scale=="centerObs", row number of case plot should univariately be centered on
missing	method used to handle missing values (see Details)
order	method used to order the axes (see Details)
showPoints	logical operator indicating whether points should be plotted or not
splineFactor	logical or numeric operator indicating whether spline interpolation should be used. Numeric values will multiplied by the number of columns, TRUE will default to cubic interpolation, AsIs to set the knot count directly and 0, FALSE, or non-numeric values will not use spline interpolation.
alphaLines	value of alpha scaler for the lines of the parcoord plot or a column name of the data
boxplot	logical operator indicating whether or not boxplots should underlay the distribution of each variable
shadeBox	color of underlying box which extends from the min to the max for each variable (no box is plotted if shadeBox == NULL)
mapping	aes string to pass to ggplot object
title	character string denoting the title of the plot

Details

scale is a character string that denotes how to scale the variables in the parallel coordinate plot. Options:

- std: univariately, subtract mean and divide by standard deviation
- robust: univariately, subtract median and divide by median absolute deviation
- uniminmax: univariately, scale so the minimum of the variable is zero, and the maximum is one
- globalminmax: no scaling is done; the range of the graphs is defined by the global minimum and the global maximum
- center: use uniminmax to standardize vertical height, then center each variable at a value specified by the scaleSummary param
- centerObs: use uniminmax to standardize vertical height, then center each variable at the value of the observation specified by the centerObsID param

missing is a character string that denotes how to handle missing missing values. Options:

- exclude: remove all cases with missing values
- mean: set missing values to the mean of the variable
- median: set missing values to the median of the variable
- min10: set missing values to 10% below the minimum of the variable

- random: set missing values to value of randomly chosen observation on that variable

order is either a vector of indices or a character string that denotes how to order the axes (variables) of the parallel coordinate plot. Options:

- (default): order by the vector denoted by columns
- (given vector): order by the vector specified
- anyClass: order variables by their separation between any one class and the rest (as opposed to their overall variation between classes). This is accomplished by calculating the F-statistic for each class vs. the rest, for each axis variable. The axis variables are then ordered (decreasing) by their maximum of k F-statistics, where k is the number of classes.
- allClass: order variables by their overall F statistic (decreasing) from an ANOVA with groupColumn as the explanatory variable (note: it is required to specify a groupColumn with this ordering method). Basically, this method orders the variables by their variation between classes (most to least).
- skewness: order variables by their sample skewness (most skewed to least skewed)
- Outlying: order by the scagnostic measure, Outlying, as calculated by the package scagnostics. Other scagnostic measures available to order by are Skewed, Clumpy, Sparse, Striated, Convex, Skinny, Stringy, and Monotonic. Note: To use these methods of ordering, you must have the scagnostics package loaded.

Value

ggplot object that if called, will print

Author(s)

Jason Crowley, Barret Schloerke, Di Cook, Heike Hofmann, Hadley Wickham

Examples

```
# small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

# use sample of the diamonds data for illustrative purposes
data(diamonds, package="ggplot2")
diamonds.samp <- diamonds[sample(1:dim(diamonds)[1], 100), ]

# basic parallel coordinate plot, using default settings
p <- ggparcoord(data = diamonds.samp, columns = c(1, 5:10))
p_(p)

# this time, color by diamond cut
p <- ggparcoord(data = diamonds.samp, columns = c(1, 5:10), groupColumn = 2)
p_(p)

# underlay univariate boxplots, add title, use uniminmax scaling
p <- ggparcoord(data = diamonds.samp, columns = c(1, 5:10), groupColumn = 2,
  scale = "uniminmax", boxplot = TRUE, title = "Parallel Coord. Plot of Diamonds Data")
p_(p)

# utilize ggplot2 aes to switch to thicker lines
p <- ggparcoord(data = diamonds.samp, columns = c(1, 5:10), groupColumn = 2,
  title ="Parallel Coord. Plot of Diamonds Data", mapping = ggplot2::aes(size = 1)) +
```

```
  ggplot2::scale_size_identity()
p_(p)

# basic parallel coord plot of the msleep data, using 'random' imputation and
# coloring by diet (can also use variable names in the columns and groupColumn
# arguments)
data(msleep, package="ggplot2")
p <- ggparcoord(data = msleep, columns = 6:11, groupColumn = "vore", missing =
  "random", scale = "uniminmax")
p_(p)

# center each variable by its median, using the default missing value handler,
# 'exclude'
p <- ggparcoord(data = msleep, columns = 6:11, groupColumn = "vore", scale =
  "center", scaleSummary = "median")
p_(p)

# with the iris data, order the axes by overall class (Species) separation using
# the anyClass option
p <- ggparcoord(data = iris, columns = 1:4, groupColumn = 5, order = "anyClass")
p_(p)

# add points to the plot, add a title, and use an alpha scalar to make the lines
# transparent
p <- ggparcoord(data = iris, columns = 1:4, groupColumn = 5, order = "anyClass",
  showPoints = TRUE, title = "Parallel Coordinate Plot for the Iris Data",
  alphaLines = 0.3)
p_(p)

# color according to a column
iris2 <- iris
iris2$alphaLevel <- c("setosa" = 0.2, "versicolor" = 0.3, "virginica" = 0)[iris2$Species]
p <- ggparcoord(data = iris2, columns = 1:4, groupColumn = 5, order = "anyClass",
  showPoints = TRUE, title = "Parallel Coordinate Plot for the Iris Data",
  alphaLines = "alphaLevel")
p_(p)

## Use splines on values, rather than lines (all produce the same result)
columns <- c(1, 5:10)
p <- ggparcoord(diamonds.samp, columns, groupColumn = 2, splineFactor = TRUE)
p_(p)
p <- ggparcoord(diamonds.samp, columns, groupColumn = 2, splineFactor = 3)
p_(p)
splineFactor <- length(columns) * 3
p <- ggparcoord(diamonds.samp, columns, groupColumn = 2, splineFactor = I(splineFactor))
p_(p)
```

`ggscatmat` *Traditional scatterplot matrix for purely quantitative variables*

Description

This function makes a scatterplot matrix for quantitative variables with density plots on the diagonal and correlation printed in the upper triangle.

Usage

```
ggscatmat(
  data,
  columns = 1:ncol(data),
  color = NULL,
  alpha = 1,
  corMethod = "pearson"
)
```

Arguments

data	a data matrix. Should contain numerical (continuous) data.
columns	an option to choose the column to be used in the raw dataset. Defaults to 1:ncol(data).
color	an option to group the dataset by the factor variable and color them by different colors. Defaults to NULL, i.e. no coloring. If supplied, it will be converted to a factor.
alpha	an option to set the transparency in scatterplots for large data. Defaults to 1.
corMethod	method argument supplied to cor

Author(s)

Mengjia Ni, Di Cook

Examples

```
# small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

data(flea)

p_(ggscatmat(flea, columns = 2:4))
p_(ggscatmat(flea, columns = 2:4, color = "species"))
```

---

ggsurv *Survival curves*

Description

This function produces Kaplan-Meier plots using **ggplot2**. As a first argument it needs a survfit object, created by the survival package. Default settings differ for single stratum and multiple strata objects.

Usage

```
ggsurv(
  s,
  CI = "def",
  plot.cens = TRUE,
  surv.col = "gg.def",
```

```
  cens.col = "gg.def",
  lty.est = 1,
  lty.ci = 2,
  size.est = 0.5,
  size.ci = size.est,
  cens.size = 2,
  cens.shape = 3,
  back.white = FALSE,
  xlab = "Time",
  ylab = "Survival",
  main = "",
  order.legend = TRUE
)
```

Arguments

s	an object of class survfit
CI	should a confidence interval be plotted? Defaults to TRUE for single stratum objects and FALSE for multiple strata objects.
plot.cens	mark the censored observations?
surv.col	colour of the survival estimate. Defaults to black for one stratum, and to the default **ggplot2** colours for multiple strata. Length of vector with colour names should be either 1 or equal to the number of strata.
cens.col	colour of the points that mark censored observations.
lty.est	linetype of the survival curve(s). Vector length should be either 1 or equal to the number of strata.
lty.ci	linetype of the bounds that mark the 95% CI.
size.est	line width of the survival curve
size.ci	line width of the 95% CI
cens.size	point size of the censoring points
cens.shape	shape of the points that mark censored observations.
back.white	if TRUE the background will not be the default grey of ggplot2 but will be white with borders around the plot.
xlab	the label of the x-axis.
ylab	the label of the y-axis.
main	the plot label.
order.legend	boolean to determine if the legend display should be ordered by final survival time

Value

An object of class ggplot

Author(s)

Edwin Thoen

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

if (require(survival) && require(scales)) {
  data(lung, package = "survival")
  sf.lung <- survival::survfit(Surv(time, status) ~ 1, data = lung)
  p_(ggsurv(sf.lung))

  # Multiple strata examples
  sf.sex <- survival::survfit(Surv(time, status) ~ sex, data = lung)
  pl.sex <- ggsurv(sf.sex)
  p_(pl.sex)

  # Adjusting the legend of the ggsurv fit
  p_(pl.sex +
    ggplot2::guides(linetype = FALSE) +
    ggplot2::scale_colour_discrete(
      name   = 'Sex',
      breaks = c(1,2),
      labels = c('Male', 'Female')
    ))

  # Multiple factors
  lung2 <- plyr::mutate(lung, older = as.factor(age > 60))
  sf.sex2 <- survival::survfit(Surv(time, status) ~ sex + older, data = lung2)
  pl.sex2 <- ggsurv(sf.sex2)
  p_(pl.sex2)

  # Change legend title
  p_(pl.sex2 + labs(color = "New Title", linetype = "New Title"))

  # We can still adjust the plot after fitting
  data(kidney, package = "survival")
  sf.kid <- survival::survfit(Surv(time, status) ~ disease, data = kidney)
  pl.kid <- ggsurv(sf.kid, plot.cens = FALSE)
  p_(pl.kid)

  # Zoom in to first 80 days
  p_(pl.kid + ggplot2::coord_cartesian(xlim = c(0, 80), ylim = c(0.45, 1)))

  # Add the diseases names to the plot and remove legend
  p_(pl.kid +
    ggplot2::annotate(
      "text",
      label  = c("PKD", "Other", "GN", "AN"),
      x      = c(90, 125, 5, 60),
      y      = c(0.8, 0.65, 0.55, 0.30),
      size   = 5,
      colour = scales::hue_pal(
        h         = c(0, 360) + 15,
        c         = 100,
        l         = 65,
        h.start   = 0,
        direction = 1
      )(4)
```

```
    ) +
    ggplot2::guides(color = FALSE, linetype = FALSE))
}
```

---

ggtable *Cross-tabulated tables of discrete variables*

Description

ggtable is a variant of ggduo for quick cross-tabulated tables of discrete variables.

Usage

```
ggtable(
  data,
  columnsX = 1:ncol(data),
  columnsY = 1:ncol(data),
  cells = c("observed", "prop", "row.prop", "col.prop", "expected", "resid",
    "std.resid"),
  fill = c("none", "std.resid", "resid"),
  mapping = NULL,
  ...
)
```

Arguments

data	dataset to be used, can have both categorical and numerical variables
columnsX, columnsY	names or positions of which columns are used to make plots. Defaults to all columns.
cells	Which statistic should be displayed in table cells?
fill	Which statistic should be used for filling table cells?
mapping	additional aesthetic to be used, for example to indicate weights (see examples)
...	additional arguments passed to ggduo (see examples)

Author(s)

Joseph Larmarange

Examples

```
# small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

if (require(reshape)) {
  data(tips, package = "reshape")
  p_(ggtable(tips, "smoker", c("day", "time", "sex")))

  # displaying row proportions
  p_(ggtable(tips, "smoker", c("day", "time", "sex"), cells = "row.prop"))
```

```
  # filling cells with standardized residuals
  p_(ggtable(tips, "smoker", c("day", "time", "sex"), fill = "std.resid", legend = 1))

  # if continuous variables are provided, just displaying some summary statistics
  p_(ggtable(tips, c("smoker", "total_bill"), c("day", "time", "sex", "tip")))
}

# specifying weights
d <- as.data.frame(Titanic)
p_(ggtable(
  d,
  "Survived",
  c("Class", "Sex", "Age"),
  mapping = aes(weight = Freq),
  cells = "row.prop",
  fill = "std.resid"
))
```

`ggts` *Multiple time series*

Description

GGally implementation of ts.plot. Wraps around the ggduo function and removes the column strips

Usage

```
ggts(..., columnLabelsX = NULL, xlab = "time")
```

Arguments

`...`	supplied directly to ggduo
`columnLabelsX`	remove top strips for the X axis by default
`xlab`	defaults to "time"

Value

ggmatrix object

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

p_(ggts(pigs, "time", c("gilts", "profit", "s_per_herdsz", "production", "herdsz")))
```

glyphplot *Glyph plot class*

Description

Glyph plot class

Usage

```
glyphplot(data, width, height, polar, x_major, y_major)

is.glyphplot(x)

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

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

Arguments

`data`	A data frame containing variables named in `x_major`, `x_minor`, `y_major` and `y_minor`.
`height, width`	The height and width of each glyph. Defaults to 95% of the `resolution` of the data. Specify the width absolutely by supplying a numeric vector of length 1, or relative to the
`polar`	A logical of length 1, specifying whether the glyphs should be drawn in polar coordinates. Defaults to `FALSE`.
`x_major, y_major`	The name of the variable (as a string) for the major x and y axes. Together, the
`x`	glyphplot to be printed
`...`	ignored

Author(s)

Di Cook, Heike Hofmann, Hadley Wickham

glyphs *Create `glyphplot` data*

Description

Create the data needed to generate a glyph plot.

Usage

```
glyphs(
  data,
  x_major,
  x_minor,
  y_major,
  y_minor,
  polar = FALSE,
  height = ggplot2::rel(0.95),
  width = ggplot2::rel(0.95),
  y_scale = identity,
  x_scale = identity
)
```

Arguments

`data`	A data frame containing variables named in `x_major`, `x_minor`, `y_major` and `y_minor`.
`x_major, x_minor, y_major, y_minor`	The name of the variable (as a string) for the major and minor x and y axes. Together, each unique
`polar`	A logical of length 1, specifying whether the glyphs should be drawn in polar coordinates. Defaults to `FALSE`.
`height, width`	The height and width of each glyph. Defaults to 95% of the `resolution` of the data. Specify the width absolutely by supplying a numeric vector of length 1, or relative to the
`y_scale, x_scale`	The scaling function to be applied to each set of minor values within a grid cell. Defaults to `identity` so that no scaling is performed.

Author(s)

Di Cook, Heike Hofmann, Hadley Wickham

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

data(nasa)
nasaLate <- nasa[
  nasa$date >= as.POSIXct("1998-01-01") &
  nasa$lat >= 20 &
  nasa$lat <= 40 &
  nasa$long >= -80 &
  nasa$long <= -60
, ]
temp.gly <- glyphs(nasaLate, "long", "day", "lat", "surftemp", height=2.5)
p_(ggplot2::ggplot(temp.gly, ggplot2::aes(gx, gy, group = gid)) +
  add_ref_lines(temp.gly, color = "grey90") +
  add_ref_boxes(temp.gly, color = "grey90") +
  ggplot2::geom_path() +
  ggplot2::theme_bw() +
  ggplot2::labs(x = "", y = ""))
```

grab_legend *Grab the legend and print it as a plot*

Description

Grab the legend and print it as a plot

Usage

```
grab_legend(p)

## S3 method for class 'legend_guide_box'
print(x, ..., plotNew = FALSE)
```

Arguments

p	ggplot2 plot object
x	legend object that has been grabbed from a ggplot2 object
...	ignored
plotNew	boolean to determine if the `grid.newpage()` command and a new blank rectangle should be printed

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

library(ggplot2)
histPlot <- qplot(
  x = Sepal.Length,
  data = iris,
  fill = Species,
  geom = "histogram",
  binwidth = 1/4
)
(right <- histPlot)
(bottom <- histPlot + theme(legend.position = "bottom"))
(top <- histPlot + theme(legend.position = "top"))
(left <- histPlot + theme(legend.position = "left"))

p_(grab_legend(right))
p_(grab_legend(bottom))
p_(grab_legend(top))
p_(grab_legend(left))
```

happy *Data related to happiness from the General Social Survey, 1972-2006.*

Description

This data extract is taken from Hadley Wickham's `productplots` package. The original description follows, with minor edits.

Usage

```
data(happy)
```

Format

A data frame with 51020 rows and 10 variables

Details

The data is a small sample of variables related to happiness from the General Social Survey (GSS). The GSS is a yearly cross-sectional survey of Americans, run from 1972. We combine data for 25 years to yield 51,020 observations, and of the over 5,000 variables, we select nine related to happiness:

- age. age in years: 18–89.
- degree. highest education: lt high school, high school, junior college, bachelor, graduate.
- finrela. relative financial status: far above, above average, average, below average, far below.
- happy. happiness: very happy, pretty happy, not too happy.
- health. health: excellent, good, fair, poor.
- marital. marital status: married, never married, divorced, widowed, separated.
- sex. sex: female, male.
- wtsall. probability weight. 0.43–6.43.

References

Smith, Tom W., Peter V. Marsden, Michael Hout, Jibum Kim. *General Social Surveys, 1972-2006*. [machine-readable data file]. Principal Investigator, Tom W. Smith; Co-Principal Investigators, Peter V. Marsden and Michael Hout, NORC ed. Chicago: National Opinion Research Center, producer, 2005; Storrs, CT: The Roper Center for Public Opinion Research, University of Connecticut, distributor. 1 data file (57,061 logical records) and 1 codebook (3,422 pp).

is_horizontal *Check if plot is horizontal*

Description

Check if plot is horizontal

Usage

```
is_horizontal(data, mapping, val = "y")

is_character_column(data, mapping, val = "y")
```

Arguments

data	data used in ggplot2 plot
mapping	ggplot2 aes() mapping
val	key to retrieve from mapping

Value

Boolean determining if the data is a character-like data

Examples

```
is_horizontal(iris, ggplot2::aes(Sepal.Length, Species)) # TRUE
is_horizontal(iris, ggplot2::aes(Sepal.Length, Species), "x") # FALSE
is_horizontal(iris, ggplot2::aes(Sepal.Length, Sepal.Width)) # FALSE
```

lowertriangle *lowertriangle - rearrange dataset as the preparation of* ggscatmat *function*

Description

function for making the melted dataset used to plot the lowertriangle scatterplots.

Usage

```
lowertriangle(data, columns = 1:ncol(data), color = NULL)
```

Arguments

data	a data matrix. Should contain numerical (continuous) data.
columns	an option to choose the column to be used in the raw dataset. Defaults to 1:ncol(data)
color	an option to choose a factor variable to be grouped with. Defaults to (NULL)

Author(s)

Mengjia Ni, Di Cook

Examples

```
data(flea)
head(lowertriangle(flea, columns= 2:4))
head(lowertriangle(flea))
head(lowertriangle(flea, color="species"))
```

`mapping_color_to_fill` *Aesthetic mapping color fill*

Description

Replace the fill with the color and make color NULL.

Usage

```
mapping_color_to_fill(current)
```

Arguments

`current`	the current aesthetics

`mapping_string` *Aes name*

Description

Aes name

Usage

```
mapping_string(aes_col)
```

Arguments

`aes_col`	Single value from `ggplot2::aes(...)`

Value

character string

Examples

```
mapping <- ggplot2::aes(Petal.Length)
mapping_string(mapping$x)
```

mapping_swap_x_y *Swap x and y mapping*

Description

Swap x and y mapping

Usage

```
mapping_swap_x_y(mapping)
```

Arguments

mapping	output of ggplot2::aes(...)

Value

Aes mapping with the x and y values switched

Examples

```
mapping <- ggplot2::aes(Petal.Length, Sepal.Width)
mapping
mapping_swap_x_y(mapping)
```

model_response_variables *Model term names*

Description

Retrieve either the response variable names, the beta variable names, or beta variable names. If the model is an object of class 'lm', by default, the beta variable names will include anova significance stars.

Usage

```
model_response_variables(model, data = broom::augment(model))

model_beta_variables(model, data = broom::augment(model))

model_beta_label(model, data = broom::augment(model), lmStars = TRUE)
```

Arguments

model	model in question
data	equivalent to broom::augment(model)
lmStars	boolean that determines if stars are added to labels

Value

character vector of names

nasa *Data from the Data Expo JSM 2006.*

Description

This data was provided by NASA for the competition.

Usage

```
data(nasa)
```

Format

A data frame with 41472 rows and 17 variables

Details

The data shows 6 years of monthly measurements of a 24x24 spatial grid from Central America:

- time integer specifying temporal order of measurements
- x, y, lat, long spatial location of measurements.
- cloudhigh, cloudlow, cloudmid, ozone, pressure, surftemp, temperature are the various satellite measurements.
- date, day, month, year specifying the time of measurements.
- id unique ide for each spatial position.

References

Murrell, P. (2010) The 2006 Data Expo of the American Statistical Association. Computational Statistics, 25:551-554.

pigs *United Kingdom Pig Production*

Description

This data contains about the United Kingdom Pig Production from the book 'Data' by Andrews and Herzberg. The original data can be on Statlib: http://lib.stat.cmu.edu/datasets/Andrews/T62.1

Usage

```
data(pigs)
```

Format

A data frame with 48 rows and 8 variables

Details

The time variable has been added from a combination of year and quarter

- time year + (quarter - 1) / 4
- year year of production
- quarter quarter of the year of production
- gilts number of sows giving birth for the first time
- profit ratio of price to an index of feed price
- s_per_herdsz ratio of the number of breeding pigs slaughtered to the total breeding herd size
- production number of pigs slaughtered that were reared for meat
- herdsz breeding herd size

References

Andrews, David F., and Agnes M. Herzberg. Data: a collection of problems from many fields for the student and research worker. Springer Science & Business Media, 2012.

`print.ggmatrix` *Print* `ggmatrix` *object*

Description

Print method taken from `ggplot2:::print.ggplot` and altered for a `ggmatrix` object

Usage

```
## S3 method for class 'ggmatrix'
print(x, newpage = is.null(vp), vp = NULL, ...)
```

Arguments

`x`	plot to display
`newpage`	draw new (empty) page first?
`vp`	viewport to draw plot in
`...`	arguments passed onto `ggmatrix_gtable`

Author(s)

Barret Schloerke

Examples

```
data(tips, package = "reshape")
pMat <- ggpairs(tips, c(1,3,2), mapping = ggplot2::aes_string(color = "sex"))
pMat # calls print(pMat), which calls print.ggmatrix(pMat)
```

print_if_interactive *Print if not CRAN*

Description

Small function to print a plot if the R session is interactive or in a CI build

Usage

```
print_if_interactive(p)
```

Arguments

p	plot to be displayed

psychademic *UCLA canonical correlation analysis data*

Description

This data contains 600 observations on eight variables

Usage

```
data(psychademic)
```

Format

A data frame with 600 rows and 8 variables

Details

- locus_of_control - psychological
- self_concept - psychological
- motivation - psychological. Converted to four character groups
- read - academic
- write - academic
- math - academic
- science - academic
- female - academic. Dropped from original source
- sex - academic. Added as a character version of female column

References

R Data Analysis Examples | Canonical Correlation Analysis. UCLA: Institute for Digital Research and Education. from http://www.stats.idre.ucla.edu/r/dae/canonical-correlation-analysis (accessed May 22, 2017).

putPlot *Insert a plot into a* ggmatrix *object*

Description

Function to place your own plot in the layout.

Usage

```
putPlot(pm, value, i, j)

## S3 replacement method for class 'ggmatrix'
pm[i, j, ...] <- value
```

Arguments

pm	ggally object to be altered
value	ggplot object to be placed
i	row from the top
j	column from the left
...	ignored

Author(s)

Barret Schloerke

See Also

getPlot

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

custom_car <- ggpairs(mtcars[, c("mpg", "wt", "cyl")], upper = "blank", title = "Custom Example")
# ggplot example taken from example(geom_text)
  plot <- ggplot2::ggplot(mtcars, ggplot2::aes(x=wt, y=mpg, label=rownames(mtcars)))
  plot <- plot +
    ggplot2::geom_text(ggplot2::aes(colour=factor(cyl)), size = 3) +
    ggplot2::scale_colour_discrete(l=40)
custom_car[1, 2] <- plot
personal_plot <- ggally_text(
  "ggpairs allows you\nto put in your\nown plot.\nLike that one.\n <---"
)
custom_car[1, 3] <- personal_plot
# custom_car

# remove plots after creating a plot matrix
custom_car[2,1] <- NULL
custom_car[3,1] <- "blank" # the same as storing null
custom_car[3,2] <- NULL
p_(custom_car)
```

remove_color_unless_equal *Remove colour mapping unless found in select mapping keys*

Description

Remove colour mapping unless found in select mapping keys

Usage

```
remove_color_unless_equal(mapping, to = c("x", "y"))
```

Arguments

mapping	output of ggplot2::aes(...)
to	set of mapping keys to check

Value

Aes mapping with colour mapping kept only if found in selected mapping keys.

Examples

```
mapping <- aes(x = sex, y = age, colour = sex)

mapping <- aes(x = sex, y = age, colour = region)
remove_color_unless_equal(mapping)
```

rescale01 *Rescaling functions*

Description

Rescaling functions

Usage

```
range01(x)

max1(x)

mean0(x)

min0(x)

rescale01(x, xlim = NULL)

rescale11(x, xlim = NULL)
```

Arguments

x	numeric vector
xlim	value used in range

---

scag_order *Find order of variables*

Description

Find order of variables based on a specified scagnostic measure by maximizing the index values of that measure along the path.

Usage

```
scag_order(scag, vars, measure)
```

Arguments

scag	scagnostics object
vars	character vector of the variables to be ordered
measure	scagnostics measure to order according to

Value

character vector of variable ordered according to the given scagnostic measure

Author(s)

Barret Schloerke

---

scatmat *Plots the lowertriangle and density plots of the scatter plot matrix.*

Description

Function for making scatterplots in the lower triangle and diagonal density plots.

Usage

```
scatmat(data, columns = 1:ncol(data), color = NULL, alpha = 1)
```

Arguments

data	a data matrix. Should contain numerical (continuous) data.
columns	an option to choose the column to be used in the raw dataset. Defaults to 1:ncol(data)
color	an option to group the dataset by the factor variable and color them by different colors. Defaults to NULL
alpha	an option to set the transparency in scatterplots for large data. Defaults to 1.

Author(s)

Mengjia Ni, Di Cook

Examples

```
# small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

data(flea)

p_(scatmat(flea, columns=2:4))
p_(scatmat(flea, columns= 2:4, color="species"))
```

---

signif_stars *Significance Stars*

Description

Calculate significance stars

Usage

```
signif_stars(x, three = 0.001, two = 0.01, one = 0.05, point = 0.1)
```

Arguments

x	numeric values that will be compared to the point, one, two, and three values
three	threshold below which to display three stars
two	threshold below which to display two stars
one	threshold below which to display one star
point	threshold below which to display one point (NULL to deactivate)

Value

character vector containing the appropriate number of stars for each x value

Author(s)

Joseph Larmarange

Examples

```
x <- c(0.5, 0.1, 0.05, 0.01, 0.001)
signif_stars(x)
signif_stars(x, one = .15, point = NULL)
```

singleClassOrder *Order axis variables*

Description

Order axis variables by separation between one class and the rest (most separation to least).

Usage

```
singleClassOrder(classVar, axisVars, specClass = NULL)
```

Arguments

classVar	class variable (vector from original dataset)
axisVars	variables to be plotted as axes (data frame)
specClass	character string matching to level of classVar; instead of looking for separation between any class and the rest, will only look for separation between this class and the rest

Value

character vector of names of axisVars ordered such that the first variable has the most separation between one of the classes and the rest, and the last variable has the least (as measured by F-statistics from an ANOVA)

Author(s)

Jason Crowley

skewness *Sample skewness*

Description

Calculate the sample skewness of a vector while ignoring missing values.

Usage

```
skewness(x)
```

Arguments

x	numeric vector

Value

sample skewness of x

Author(s)

Jason Crowley

stat_cross *Compute cross-tabulation statistics*

Description

Computes statistics of a 2-dimensional matrix using augment.htest from **broom**.

Usage

```
stat_cross(
  mapping = NULL,
  data = NULL,
  geom = "point",
  position = "identity",
  ...,
  na.rm = TRUE,
  show.legend = NA,
  inherit.aes = TRUE,
  keep.zero.cells = FALSE
)
```

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)).
geom	Override the default connection between geom_point and stat_prop.
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 TRUE, 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().

keep.zero.cells
: If TRUE, cells with no observations are kept.

Aesthetics

stat_prop requires the **x** and the **y** aesthetics.

Computed variables

observed number of observations in x,y

prop proportion of total

row.prop row proportion

col.prop column proportion

expected expected count under the null hypothesis

resid Pearson's residual

std.resid standardized residual

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

d <- as.data.frame(Titanic)

# plot number of observations
p_(ggplot(d) +
 aes(x = Class, y = Survived, weight = Freq, size = after_stat(observed)) +
 stat_cross() +
 scale_size_area(max_size = 20))

# custom shape and fill colour based on chi-squared residuals
p_(ggplot(d) +
 aes(
   x = Class, y = Survived, weight = Freq,
   size = after_stat(observed), fill = after_stat(std.resid)
 ) +
 stat_cross(shape = 22) +
 scale_fill_steps2(breaks = c(-3, -2, 2, 3), show.limits = TRUE) +
 scale_size_area(max_size = 20))

# plotting the number of observations as a table
p_(ggplot(d) +
 aes(
   x = Class, y = Survived, weight = Freq, label = after_stat(observed)
 ) +
 geom_text(stat = "cross"))

# Row proportions with standardized residuals
p_(ggplot(d) +
  aes(
    x = Class, y = Survived, weight = Freq,
    label = scales::percent(after_stat(row.prop)),
    size = NULL, fill = after_stat(std.resid)
  ) +
  stat_cross(shape = 22, size = 30) +
```

```
  geom_text(stat = "cross") +
  scale_fill_steps2(breaks = c(-3, -2, 2, 3), show.limits = TRUE) +
  facet_grid(Sex ~ .) +
  labels(fill = "Standardized residuals") +
  theme_minimal())

# can work with continuous or character variables
data(tips, package = "reshape")
p_(ggplot(tips) +
  aes(x = tip, y = as.character(day), size = after_stat(observed)) +
  stat_cross(alpha = .1, color = "blue") +
  scale_size_area(max_size = 12))
```

`stat_prop` *Compute proportions according to custom denominator*

Description

`stat_prop` is a variation of `ggplot2::stat_count()` allowing to compute custom proportions according to the **by** aesthetic defining the denominator (i.e. all proportions for a same value of **by** will sum to 1). The by aesthetic should be a factor.

Usage

```
stat_prop(
  mapping = NULL,
  data = NULL,
  geom = "bar",
  position = "fill",
  ...,
  width = NULL,
  na.rm = FALSE,
  orientation = NA,
  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. A `function` can be created from a `formula` (e.g. `~ head(.x,10)`).

geom	Override the default connection between geom_bar and stat_prop.
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.
width	Bar width. By default, set to 90% of the resolution of the data.
na.rm	If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
orientation	The orientation of the layer. The default (NA) automatically determines the orientation from the aesthetic mapping. In the rare event that this fails it can be given explicitly by setting orientation to either "x" or "y". See the *Orientation* section for more detail.
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().

Aesthetics

stat_prop() understands the following aesthetics (required aesthetics are in bold):

- **x *or* y**
- **by** (this aesthetic should be a **factor**)
- group
- weight

Computed variables

count number of points in bin

prop computed proportion

Author(s)

Joseph Larmarange

See Also

ggplot2::stat_count()

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

d <- as.data.frame(Titanic)

p <- ggplot(d) +
  aes(x = Class, fill = Survived, weight = Freq, by = Class) +
```

```
  geom_bar(position = "fill") +
  geom_text(stat = "prop", position = position_fill(.5))
p_(p)
p_(p + facet_grid(~ Sex))

p_(ggplot(d) +
  aes(x = Class, fill = Survived, weight = Freq) +
  geom_bar(position = "dodge") +
  geom_text(
    aes(by = Survived), stat = "prop",
    position = position_dodge(0.9), vjust = "bottom"
 ))

p_(ggplot(d) +
  aes(x = Class, fill = Survived, weight = Freq, by = 1) +
  geom_bar() +
  geom_text(
    aes(label = scales::percent(after_stat(prop), accuracy = 1)),
    stat = "prop",
    position = position_stack(.5)
 ))
```

`stat_weighted_mean` *Compute weighted y mean*

Description

This statistic will compute the mean of **y** aesthetic for each unique value of **x**, taking into account **weight** aesthetic if provided.

Usage

```
stat_weighted_mean(
  mapping = NULL,
  data = NULL,
  geom = "point",
  position = "identity",
  ...,
  na.rm = FALSE,
  orientation = NA,
  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. A function can be created from a formula (e.g. ~ head(.x,10)).
geom	Use to override the default connection between geom_histogram()/geom_freqpoly() and stat_bin().
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.
orientation	The orientation of the layer. The default (NA) automatically determines the orientation from the aesthetic mapping. In the rare event that this fails it can be given explicitly by setting orientation to either "x" or "y". See the *Orientation* section for more detail.
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().

Computed variables

y weighted y (numerator / denominator)

numerator numerator

denominator denominator

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

data(tips, package = "reshape")

p_(ggplot(tips) +
  aes(x = day, y = total_bill) +
  geom_point())

p_(ggplot(tips) +
  aes(x = day, y = total_bill) +
  stat_weighted_mean())

p_(ggplot(tips) +
  aes(x = day, y = total_bill, group = 1) +
  stat_weighted_mean(geom = "line"))

p_(ggplot(tips) +
```

```
  aes(x = day, y = total_bill, colour = sex, group = sex) +
  stat_weighted_mean(geom = "line"))

p_(ggplot(tips) +
  aes(x = day, y = total_bill, fill = sex) +
  stat_weighted_mean(geom = "bar", position = "dodge"))

# computing a proportion on the fly
p_(ggplot(tips) +
  aes(x = day, y = as.integer(smoker == "Yes"), fill = sex) +
  stat_weighted_mean(geom = "bar", position = "dodge") +
  scale_y_continuous(labels = scales::percent))

# taking into account some weights
d <- as.data.frame(Titanic)
p_(ggplot(d) +
  aes(x = Class, y = as.integer(Survived == "Yes"), weight = Freq, fill = Sex) +
  geom_bar(stat = "weighted_mean", position = "dodge") +
  scale_y_continuous(labels = scales::percent) +
  labs(y = "Survived"))


## Not run:
cuse <- read.table("https://data.princeton.edu/wws509/datasets/cuse.dat", header = TRUE)
cuse$n <- cuse$notUsing + cuse$using
cuse$prop <- cuse$using / cuse$n

ggplot(cuse) +
  aes(x = education, y = prop, weight = n) +
  stat_weighted_mean()

ggplot(cuse) +
  aes(x = age, y = prop, weight = n, color = education) +
  stat_weighted_mean()

ggplot(cuse) +
  aes(x = education, y = prop, weight = n) +
  stat_weighted_mean(geom = "bar")

# add percentages above each bar
ggplot(cuse) +
  aes(x = age, y = prop, weight = n, fill = education) +
  stat_weighted_mean(geom = "bar") +
 geom_text(aes(label = scales::percent(after_stat(y))), stat = "weighted_mean", vjust = 0) +
  facet_grid(~ education)

## End(Not run)
```

`str.ggmatrix` `ggmatrix` *structure*

Description

View the condensed version of the `ggmatrix` object. The attribute "class" is ALWAYS altered to "_class" to avoid recursion.

Usage

```
## S3 method for class 'ggmatrix'
str(object, ..., raw = FALSE)
```

Arguments

object	ggmatrix object to be viewed
...	passed on to the default str method
raw	boolean to determine if the plots should be converted to text or kept as original objects

---

twitter_spambots *Twitter spambots*

Description

A network of spambots found on Twitter as part of a data mining project.

Usage

```
data(twitter_spambots)
```

Format

An object of class network with 120 edges and 94 vertices.

Details

Each node of the network is identified by the Twitter screen name of the account and further carries five vertex attributes:

- location user's location, as provided by the user
- lat latitude, based on the user's location
- lon longitude, based on the user's location
- followers number of Twitter accounts that follow this account
- friends number of Twitter accounts followed by the account

Author(s)

Amos Elberg

uppertriangle *Rearrange dataset as the preparation of* ggscatmat *function*

Description

Function for making the dataset used to plot the uppertriangle plots.

Usage

```
uppertriangle(
  data,
  columns = 1:ncol(data),
  color = NULL,
  corMethod = "pearson"
)
```

Arguments

data	a data matrix. Should contain numerical (continuous) data.
columns	an option to choose the column to be used in the raw dataset. Defaults to 1:ncol(data)
color	an option to choose a factor variable to be grouped with. Defaults to (NULL)
corMethod	method argument supplied to cor

Author(s)

Mengjia Ni, Di Cook

Examples

```
data(flea)
head(uppertriangle(flea, columns=2:4))
head(uppertriangle(flea))
head(uppertriangle(flea, color="species"))
```

v1_ggmatrix_theme *Modify a* ggmatrix *object by adding an* **ggplot2** *object to all*

Description

Modify a ggmatrix object by adding an **ggplot2** object to all

Usage

```
v1_ggmatrix_theme()
```

Examples

```
# Small function to display plots only if it's interactive
p_ <- GGally::print_if_interactive

p_(ggpairs(iris, 1:2) + v1_ggmatrix_theme())
# move the column names to the left and bottom
p_(ggpairs(iris, 1:2, switch = "both") + v1_ggmatrix_theme())
```

`vig_ggally` *View GGally vignettes*

Description

This function will open the directly to the vignette requested. If no `name` is provided, the index of all **GGally** vignettes will be opened.

Usage

```
vig_ggally(name)
```

Arguments

`name`	Vignette name to open. If no name is provided, the vignette index will be opened

Details

This method allows for vignettes to be hosted remotely, reducing **GGally**'s package size, and installation time.

Examples

```
# View `ggnostic` vignette
vig_ggally("ggnostic")

# View all vignettes by GGally
vig_ggally()
```

`wrap_fn_with_param_arg`

Wrap a function with different parameter values

Description

Wraps a function with the supplied parameters to force different default behavior. This is useful for functions that are supplied to ggpairs. It allows you to change the behavior of one function, rather than creating multiple functions with different parameter settings.

Usage

```
wrap_fn_with_param_arg(
  funcVal,
  params = NULL,
  funcArgName = deparse(substitute(funcVal))
)

wrapp(funcVal, params = NULL, funcArgName = deparse(substitute(funcVal)))

wrap(funcVal, ..., funcArgName = deparse(substitute(funcVal)))

wrap_fn_with_params(funcVal, ..., funcArgName = deparse(substitute(funcVal)))
```

Arguments

funcVal	function that the params will be applied to. The function should follow the api of function(data,mapping,...){}. funcVal is allowed to be a string of one of the ggally_NAME functions, such as "points" for ggally_points or "facetdensity" for ggally_facetdensity.
params	named vector or list of parameters to be applied to the funcVal
funcArgName	name of function to be displayed
...	named parameters to be supplied to wrap_fn_with_param_arg

Details

wrap is identical to wrap_fn_with_params. These function take the new parameters as arguments.

wrapp is identical to wrap_fn_with_param_arg. These functions take the new parameters as a single list.

The params and fn attributes are there for debugging purposes. If either attribute is altered, the function must be re-wrapped to have the changes take effect.

Value

a function(data,mapping,...){} that will wrap the original function with the parameters applied as arguments

Examples

```
 # small function to display plots only if it's interactive
 p_ <- GGally::print_if_interactive

# example function that prints 'val'
fn <- function(data, mapping, val = 2) {
  print(val)
}
fn(data = NULL, mapping = NULL) # 2

# wrap function to change default value 'val' to 5 instead of 2
wrapped_fn1 <- wrap(fn, val = 5)
wrapped_fn1(data = NULL, mapping = NULL) # 5
# you may still supply regular values
wrapped_fn1(data = NULL, mapping = NULL, val = 3) # 3
```

```
# wrap function to change 'val' to 5 using the arg list
wrapped_fn2 <- wrap_fn_with_param_arg(fn, params = list(val = 5))
wrapped_fn2(data = NULL, mapping = NULL) # 5

# change parameter settings in ggpairs for a particular function
## Goal output:
regularPlot <- ggally_points(
  iris,
  ggplot2::aes(Sepal.Length, Sepal.Width),
  size = 5, color = "red"
)
p_(regularPlot)

# Wrap ggally_points to have parameter values size = 5 and color = 'red'
w_ggally_points <- wrap(ggally_points, size = 5, color = "red")
wrappedPlot <- w_ggally_points(
  iris,
  ggplot2::aes(Sepal.Length, Sepal.Width)
)
p_(wrappedPlot)

# Double check the aes parameters are the same for the geom_point layer
identical(regularPlot$layers[[1]]$aes_params, wrappedPlot$layers[[1]]$aes_params)

# Use a wrapped function in ggpairs
pm <- ggpairs(iris, 1:3, lower = list(continuous = wrap(ggally_points, size = 5, color = "red")))
p_(pm)
pm <- ggpairs(iris, 1:3, lower = list(continuous = w_ggally_points))
p_(pm)
```

Index