Package 'ggpmisc'

July 5, 2026

Type Package

Title Miscellaneous Extensions to 'ggplot2'

Version 1.0.0

Date 2026-07-05

Maintainer Pedro J. Aphalo <pedro.aphalo@helsinki.fi>

Description Extensions to 'ggplot2' respecting the grammar of graphics paradigm. Statistics to locate and tag peaks and valleys and to label plots with the equation of a fitted polynomial model by ordinary least squares, major axis, quantile and robust and resistant regression approaches. Line and model equation for Normal mixture models. Labels for P-value, R^2 or adjusted R^2 or information criteria for fitted models; parametric and non-parametric correlation; ANOVA table or summary table for fitted models as plot insets; annotations for multiple pairwise comparisons with adjusted P-values. Model fit classes for which suitable methods are provided by package 'broom' and 'broom.mixed' are supported as well as user-defined wrappers on model fit functions, allowing model selection and conditional labelling. Scales and stats to build volcano and quadrant plots based on outcomes, fold changes, p-values and false discovery rates.

License GPL (>= 2)

LazyLoad true

ByteCompile true

Depends R (>= 4.2.0), ggpp (>= 0.6.1)

Imports grid, stats, ggplot2 (>= 3.6.0), scales (>= 1.4.0), rlang (>= 1.1.5), generics (>= 0.1.4), polynom (>= 1.4-1), confintr (>= 1.0.2), splus2R (>= 1.3-5), tibble (>= 3.3.1), plyr (>= 1.8.9), dplyr (>= 1.1.4), tidyr (>= 1.3.0), lubridate (>= 1.9.5), caTools (>= 1.18.3), quantreg (>= 6.1)

Suggests broom (>= 1.0.10), broom.mixed (>= 0.2.9.5), robustbase (>= 0.99-4-1), MASS (>= 7.3-60), lmodel2 (>= 1.7-4), nlme (>= 3.1-166), smatr (>= 3.4-8), lspline (>= 1.0.0), multcomp (>= 1.4-30), multcompView (>= 0.1-11), mixtools (>= 2.0.0.1), segmented (>= 2.2-1), onls (>= 0.1-4), gginnards (>= 0.2.0),

ggrepel (>= 0.9.8), ggtext (>= 0.1.2), xdvir (>= 0.1.2), marquee (>= 1.2.1), knitr (>= 1.50), rmarkdown (>= 2.28), ragg, testthat, vdiffr

URL https://docs.r4photobiology.info/ggpmisc/, https://github.com/aphalo/ggpmisc

BugReports https://github.com/aphalo/ggpmisc/issues

Encoding UTF-8

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Pedro J. Aphalo [aut, cre] (ORCID: <https://orcid.org/0000-0003-3385-972X>), Kamil Slowikowski [ctb] (ORCID: <https://orcid.org/0000-0002-2843-6370>), Samer Mouksassi [ctb] (ORCID: <https://orcid.org/0000-0002-7152-6654>)

Repository CRAN

Date/Publication 2026-07-05 21:40:02 UTC

Contents

ggpmisc-package . . . 3
check_output_type . . . 5
check_poly_formula . . . 7
coef.lmodel2 . . . 9
coefs2poly_eq . . . 10
confint.lmodel2 . . . 11
find_peaks . . . 12
find_quantiles . . . 15
find_spikes . . . 16
keep_tidy . . . 18
outcome2factor . . . 19
plain_label . . . 20
poly2character . . . 26
predict.lmodel2 . . . 27
scale_colour_logFC . . . 28
scale_colour_outcome . . . 31
scale_shape_outcome . . . 34
scale_x_logFC . . . 35
scale_y_Pvalue . . . 38
sprintf_dm . . . 40
stat_correlation . . . 41
stat_distrmix_eq . . . 49
stat_fit_augment . . . 57
stat_fit_deviations . . . 64

stat_fit_glance . . . 75
stat_fit_tb . . . 82
stat_fit_tidy . . . 91
stat_ma_eq . . . 97
stat_multcomp . . . 110
stat_peaks . . . 118
stat_poly_eq . . . 124
stat_quant_band . . . 137
stat_spikes . . . 151
swap_xy . . . 156
symmetric_limits . . . 156
typeset_numbers . . . 157
use_label . . . 158
xy_outcomes2factor . . . 161

Index **163**

`ggpmisc-package` *ggpmisc: Miscellaneous Extensions to 'ggplot2'*

Description

Extensions to 'ggplot2' respecting the grammar of graphics paradigm. Statistics to locate and tag peaks and valleys and to label plots with the equation of a fitted polynomial model by ordinary least squares, major axis, quantile and robust and resistant regression approaches. Line and model equation for Normal mixture models. Labels for P-value, R^2 or adjusted R^2 or information criteria for fitted models; parametric and non-parametric correlation; ANOVA table or summary table for fitted models as plot insets; annotations for multiple pairwise comparisons with adjusted P-values. Model fit classes for which suitable methods are provided by package 'broom' and 'broom.mixed' are supported as well as user-defined wrappers on model fit functions, allowing model selection and conditional labelling. Scales and stats to build volcano and quadrant plots based on outcomes, fold changes, p-values and false discovery rates.

Details

Package 'ggpmisc' is over 10 years-old but its development has tracked the changes in 'ggplot2' making possible the use of several new features soon after they became available. Support for additional model fitting functions has been added regularly.

The focus of package 'ggpmisc' is on statistical annotations, providing stats that generate labels useful to annotate plots and matching stats for consitenly adding prediction lines and bands. Model fitting is done by calling functions already available in R and other R packages. No new model fit method or algorithms are implemented, instead what 'ggpmisc' provides are new simpler ways of adding fitted values and other statistics as plot annotations.

Several geometries for annotations from package 'ggpp' are used by default in 'ggpmisc' statistics, with labels formatted by default ready to be parsed into R's plotmath expressions. However, other geometries can be also used. Two variations of Markdown-formatted labels work smoothly with geoms from package 'ggtext' or from package 'marquee'. LaTeX-formatted labels work smoothly

with package 'xdvir' and most likely also with other approaches to the use of 'LaTeX' and 'TeX' formatted labels. 'LaTeX'-formatted labels can be generated as bare maths-mode-encoded text, or enclosed in "fences" that enable either in-line or display-maths modes.

The label formatting functions used to implement the statistics and scales are exported and can be used as an aid in building customised labels and scales.

Extensions provided:

- Statistics for annotations for parametric and non-parametric correlations.
- Statistics for generation of labels for fitted models, including formatted equations. By default labels are R's plotmath expressions but LaTeX, markdown and plain text formatted labels are optionally returned.
- Matching statistics for plotting curves and confidence bands bands for the same fitted models.
- Statistics for adding ANOVA tables and fitted model summaries as inset tables in plots.
- Statistic for adding annotations based on pairwise multiple comparisons based on arbitrary contrasts and a choice of *P* adjustment methods.
- Statistics for locating and tagging "peaks" and "valleys" (local or global maxima and minima) and spikes (very narrow peaks or valleys).
- Access to functions and objects exported by package ggpp.

Note

The signatures of `stat_peaks()` and `stat_valleys()` from 'ggpmisc' are nearly identical to those of `stat_peaks()` and `stat_valleys()` from package 'ggspectra'. While those from 'ggpmisc' are designed for numeric or time objects mapped to the *x* aesthetic, those from 'ggspectra' are for light spectra and expect a numeric variable describing wavelength mapped to the *x* aesthetic.

Author(s)

Maintainer: Pedro J. Aphalo <pedro.aphalo@helsinki.fi> (ORCID)

Authors:

- Pedro J. Aphalo <pedro.aphalo@helsinki.fi> (ORCID)

Other contributors:

- Kamil Slowikowski (ORCID) [contributor]
- Samer Mouksassi <samermouksassi@gmail.com> (ORCID) [contributor]

See Also

Useful links:

- https://docs.r4photobiology.info/ggpmisc/
- https://github.com/aphalo/ggpmisc
- Report bugs at https://github.com/aphalo/ggpmisc/issues

Examples

```
ggplot(lynx, as.numeric = FALSE) + geom_line() +
stat_peaks(colour = "red") +
  stat_peaks(geom = "text", colour = "red", angle = 66,
             hjust = -0.1, x.label.fmt = "%Y") +
  ylim(NA, 8000)

formula <- y ~ poly(x, 2, raw = TRUE)
ggplot(cars, aes(speed, dist)) +
  geom_point() +
  stat_poly_line(formula = formula) +
  stat_poly_eq(use_label("eq", "R2", "P"),
               formula = formula,
               parse = TRUE) +
  labs(x = expression("Speed, "*x~("mph")),
       y = expression("Stopping distance, "*y~("ft")))

formula <- y ~ x
ggplot(PlantGrowth, aes(group, weight)) +
  stat_summary(fun.data = "mean_se") +
  stat_fit_tb(method = "lm",
              method.args = list(formula = formula),
              tb.type = "fit.anova",
              tb.vars = c(Term = "term", "df", "M.S." = "meansq",
                          "italic(F)" = "statistic",
                          "italic(p)" = "p.value"),
              tb.params = c("Group" = 1, "Error" = 2),
              table.theme = ttheme_gtbw(parse = TRUE)) +
  labs(x = "Group", y = "Dry weight of plants") +
  theme_classic()
```

---

check_output_type *Validate output type*

Description

Replace `NULL` `output.type` based on `geom` and validate other values. Convert synonyms and change into lower case mal-formed input.

Usage

```
check_output_type(
  output.type,
  geom = "text",
  supported.types = c("expression", "text", "markdown", "marquee", "numeric", "latex",
    "latex.eqn", "latex.deqn")
)
```

Arguments

output.type	character User-set argument or default from stat.
geom	character The name of the geom that will be used to render the labels.
supported.types	character vector of accepted values for user input.

Value

If output.type is NULL a suitable value based on the name of the geom is returned, defaulting to "expression". If not NULL, the value is passed through unchanged.

Output types

The formatting of character strings to be displayed in plots are marked as mathematical equations. Depending on the geom used, the mark-up needs to be encoded differently, or in some cases mark-up not applied.

"expression" The labels are encoded as character strings to be parsed into R's plotmath expressions.

"LaTeX", "TeX", "tikz", "latex" The labels are encoded as 'LaTeX' maths equations, without the "fences" for switching in math mode.

"latex.eqn" Same as "latex" but enclosed in single $, i.e., as in-line maths.

"latex.deqn" Same as "latex" but enclosed in double $$, i.e., as display maths.

"markdown" The labels are encoded as character strings using markdown syntax, with some embedded HTML.

"marquee" The labels are encoded as character strings using markdown syntax, with 'marquee' supported spans.

"text" The labels are plain ASCII character strings.

"numeric" No labels are generated. This value is accepted by the statistics, but not by the label formatting functions.

NULL The value used depends on the argument passed to geom.

If geom = "latex" (package 'xdvir') the output type used is "latex.eqn". If geom = "richtext" (package 'ggtext') or geom = "textbox" (package 'ggtext') the output type used is "markdown". If geom = "marquee" (package 'marquee') the output type used is "marquee". For all other values of geom the default is "expression". Invalid values as argument trigger an error.

Examples

```
check_output_type(NULL)
check_output_type("text")
check_output_type(NULL, geom = "text")
check_output_type(NULL, geom = "latex")
```

check_poly_formula *Validate model formula as a polynomial*

Description

Analyse a model formula to determine if it describes a polynomial with terms in order of increasing powers, and fulfils the expectations of the algorithm used to generate the equation-label.

Usage

```
check_poly_formula(
  formula,
  x.name = "x",
 warn.incr.poly.text = "'formula' not an increasing polynomial: 'eq.label' set to NA!",
  warn.transf.rhs.txt =
    paste0("'formula' rhs includes transformations requiring an argument for ",
    "'eq.x.rhs': 'eq.label' set to NA!."),
  warn.transf.lhs.txt =
    paste0("'formula' lhs includes transformations requiring an argument for ",
    "'eq.with.lhs': 'eq.label' set to NA!."),
  warn.spline.rhs.txt = paste0("'formula' rhs includes a spline base function: ",
    " 'eq.label' set to NA!."),
  warn.as.is.txt = paste0("Power (^) terms in 'formula' for a polynomial need to ",
    "be protected by 'I()': 'eq.label' set to NA!."),
  warn.poly.raw.txt = paste0("'poly()' in 'formula' has to be passed 'raw = TRUE': ",
    "'eq.label' set to NA!"),
  stop.pow.poly.text =
    "Both 'poly()' and power (^) terms in 'formula', use one or the other.",
  check.transf.rhs = TRUE,
  check.transf.lhs = TRUE
)
```

Arguments

formula	A model formula in x.name.
x.name	character The name of the explanatory variable in the formula.
warn.incr.poly.text, warn.transf.lhs.txt, warn.transf.rhs.txt, warn.spline.rhs.txt, warn.as.is.txt, warn.poly.raw.txt, stop.pow.poly.text	character Text for warnings and errors. If NULL the warning is not issued and failure reported silently to the caller.
check.transf.rhs, check.transf.lhs	logical flag enabling test for transformation of variables.

Details

The assumption is that this function will be called from within a 'ggplot2' compatible layer function, and that model formulas will always have a single explanatory variable, variables will be `x` and `y`. Its behaviour is undefined or erroneous in other cases. This validation check cannot be demonstrated to be always correct, as it is difficult to test, or even list all possible variations of supported vs. unsupported formulas.

Many valid model formulas that can be successfully fitted, cannot be automatically converted into correct character labels by functions in 'ggpmisc'. Thus, this function triggers a warning in case of a test failure, not an error, returning a logic value. If this value is `FALSE`, the statistics in 'ggpmisc' skip the generation of an equation label, setting it to `NA`, while returning other character labels and the numeric estimates of the fitted coefficients. Thus, the stats can be used also with models that are not polynomials or containing transformations, but in this case the model equation when needed needs to be assembled in user's code within a call to 'aes()'.

Model formulas with and without an intercept term are accepted as valid, as `+0`, `-1` and `+1` are accepted. If a single `as.is` power term is included or if arithmetic (`sqrt()`, `exp()`, `log()`), or trigonometric functions (`cos()`, `sin()`, `tan()`, etc.) are encountered a warning is issued about the need to pass a matching argument to parameter `eq.x.rhs` of the statistic.

If two or more terms are `as.is` (`I( )` protected) powers (`^`), they are expected to be in increasing order with no missing intermediate power terms.

Spline base functions `ns()`, `bs()` and `lspline()` are searched for and flagged, while `poly()` is accepted. If `poly()` is used in the model formula, a single term is expected. When calling function `poly()`, `raw = TRUE` must be passed to obtain suitable estimates for the fitted coefficients, and this is also checked.

When the formula *rhs* contains more than one power term, all power terms defined using `^` must be protected as `"as.is" I()`, as otherwise they are not powers but instead part of the formula specification.

If the warning text is `NULL` or `character(0)` no warning is issued, but the test is done. In contrast, `check.transf.rhs,check = FALSE` and `transf.lhs = FALSE` skip these two tests on with `raw = TRUE`.

Value

A logical, TRUE if the formula describes an increasing polynomial suitable for conversion into a text label, and FALSE otherwise. When validation fails, warnings are issued describing the problem encountered.

Examples

```
# polynomials
check_poly_formula(y ~ 1)
check_poly_formula(y ~ x)
check_poly_formula(y ~ x^3)
check_poly_formula(y ~ x + 0)
check_poly_formula(y ~ x - 1)
check_poly_formula(y ~ x + 1)
check_poly_formula(y ~ x + I(x^2))
check_poly_formula(y ~ 1 + x + I(x^2))
```

```
check_poly_formula(y ~ x + I(x^2) + I(x^3))
check_poly_formula(y ~ I(x) + I(x^2) + I(x^3))

# transformations on x, first degree polynomials
check_poly_formula(y ~ sqrt(x))
check_poly_formula(y ~ log(x))
check_poly_formula(y ~ I(x^2))

# incomplete or terms in decreasing/mixed order
check_poly_formula(y ~ I(x^2) + x)
check_poly_formula(y ~ I(x^2) + I(x^3))
check_poly_formula(y ~ I(x^2) + I(x^4))
check_poly_formula(y ~ x + I(x^3) + I(x^2))

# polynomials using poly()
check_poly_formula(y ~ poly(x, 2, raw = TRUE)) # label o.k.
check_poly_formula(y ~ poly(x, 2)) # orthogonal polynomial -> bad label

# spline terms not accepted
check_poly_formula(y ~ ns(x))
check_poly_formula(y ~ bs(x))
check_poly_formula(y ~ lspline(x))
```

`coef.lmodel2` *Extract Model Coefficients*

Description

`coef` is a generic function which extracts model coefficients from objects returned by modeling functions. `coefficients` is an alias for it.

Usage

```
## S3 method for class 'lmodel2'
coef(object, method = "MA", ...)
```

Arguments

`object`	a fitted model object.
`method`	character One of the methods available in `object`.
`...`	ignored by this method.

Details

Function `lmodel2()` from package 'lmodel2' returns a fitted model object of class `"lmodel2"` which differs from that returned by `lm()`. Here we implement a `coef()` method for objects of this class. It differs from de generic method and that for lm objects in having an additional formal parameter `method` that must be used to select estimates based on which of the methods supported

by `lmodel2()` are to be extracted. The returned object is identical in its structure to that returned by `coef.lm()`.

Value

A named numeric vector of length two.

See Also

`lmodel2`

`coefs2poly_eq` *Format a polynomial as an equation*

Description

Uses a vector of coefficients from a model fit of a polynomial to build the fitted model equation with embedded coefficient estimates.

Usage

```
coefs2poly_eq(
  coefs,
  coef.digits = 3L,
  coef.keep.zeros = TRUE,
  decreasing = getOption("ggpmisc.decreasing.poly.eq", FALSE),
  eq.x.rhs = "x",
  lhs = "y~`=`~",
  output.type = "expression",
  decimal.mark = "."
)
```

Arguments

`coefs`	numeric Terms always sorted by increasing powers.
`coef.digits`	integer
`coef.keep.zeros`	logical This flag refers to trailing zeros.
`decreasing`	logical It specifies the order of the terms in the returned character string; in increasing (default) or decreasing powers.
`eq.x.rhs`	character
`lhs`	character
`output.type`	character One of "expression", "latex", "tex", "text", "tikz", "markdown", "marquee".
`decimal.mark`	character

Value

A character string.

Note

Terms with zero-valued coefficients are dropped from the polynomial.

Examples

```
coefs2poly_eq(c(1, 2, 0, 4, 5, 2e-5))
coefs2poly_eq(c(1, 2, 0, 4, 5, 2e-5), output.type = "latex")
coefs2poly_eq(0:2)
coefs2poly_eq(0:2, decreasing = TRUE)
coefs2poly_eq(c(1, 2, 0, 4, 5), coef.keep.zeros = TRUE)
coefs2poly_eq(c(1, 2, 0, 4, 5), coef.keep.zeros = FALSE)
```

---

confint.lmodel2 *Confidence Intervals for Model Parameters*

Description

Computes confidence intervals for one or more parameters in a fitted model. This a method for objects inheriting from class "lmodel2".

Usage

```
## S3 method for class 'lmodel2'
confint(object, parm, level = 0.95, method = "MA", ...)
```

Arguments

object	a fitted model object.
parm	a specification of which parameters are to be given confidence intervals, either a vector of numbers or a vector of names. If missing, all parameters are considered.
level	the confidence level required. Currently only 0.95 accepted.
method	character One of the methods available in object.
...	ignored by this method.

Details

Function lmodel2() from package 'lmodel2' returns a fitted model object of class "lmodel2" which differs from that returned by lm(). Here we implement a confint() method for objects of this class. It differs from the generic method and that for lm objects in having an additional formal parameter method that must be used to select estimates based on which of the methods supported by lmodel2() are to be extracted. The returned object is identical in its structure to that returned by confint.lm().

Value

A data frame with two rows and three columns.

See Also

lmodel2

---

find_peaks *Find local or global maxima (peaks) or minima (valleys)*

Description

These functions find peaks (maxima) and valleys (minima) in a numeric vector, using a user selectable span and global and local size thresholds, returning a logical vector.

Usage

```
find_peaks(
  x,
  global.threshold = NULL,
  local.threshold = NULL,
  local.reference = "median",
  threshold.range = NULL,
  span = 3,
  strict = FALSE,
  na.rm = FALSE
)

find_valleys(
  x,
  global.threshold = NULL,
  local.threshold = NULL,
  local.reference = "median",
  threshold.range = NULL,
  span = 3,
  strict = FALSE,
  na.rm = FALSE
)
```

Arguments

x	numeric vector.
global.threshold	numeric A value belonging to class "AsIs" is interpreted as an absolute minimum height or depth expressed in data units. A bare numeric value (normally between 0.0 and 1.0), is interpreted as relative to threshold.range. In both cases it sets a *global* height (depth) threshold below which peaks (valleys) are

ignored. A bare negative numeric value indicates the *global* height (depth) threshold below which peaks (valleys) are be ignored. If global.threshold = NULL, no threshold is applied and all peaks returned.

local.threshold

numeric A value belonging to class "AsIs" is interpreted as an absolute minimum height (depth) expressed in data units relative to a within-window computed reference value. A bare numeric value (normally between 0.0 and 1.0), is interpreted as expressed in units relative to threshold.range. In both cases local.threshold sets a *local* height (depth) threshold below which peaks (valleys) are ignored. If local.threshold = NULL or if span spans the whole of x, no threshold is applied.

local.reference

character One of "median", "median.log", "median.sqrt", "farthest", "farthest.log" or "farthest.sqrt". The reference used to assess the height of the peak, is either the minimum/maximum value within the window or the median of all values in the window.

threshold.range

numeric vector If of length 2 or a longer vector range(threshold.range) is used to scale both thresholds. With NULL, the default, range(x) is used, and with a vector of length one range(threshold.range, x) is used, i.e., the range is expanded.

span odd positive integer A peak is defined as an element in a sequence which is greater than all other elements within a moving window of width span centred at that element. The default value is 5, meaning that a peak is taller than its four nearest neighbours. span = NULL extends the span to the whole length of x.

strict logical flag: if TRUE, an element must be strictly greater than all other values in its window to be considered a peak. Default: FALSE (since version 0.13.1).

na.rm logical indicating whether NA values should be stripped before searching for peaks.

Details

As find_valleys, stat_peaks and stat_valleys call find_peaks to search for peaks or valleys, this description applies to all four functions.

Function find_peaks is a wrapper built onto function peaks from **splus2R**, adds support for peak height thresholds and handles span = NULL and non-finite (including NA) values differently than splus2R::peaks. Instead of giving an error when na.rm = FALSE and x contains NA values, NA values are replaced with the smallest finite value in x. span = NULL is treated as a special case and selects max(x). Passing 'strict = TRUE' ensures that multiple global and within window maxima are ignored, and can result in no peaks being returned.#'

Two tests make it possible to ignore irrelevant peaks. One test (global.threshold) is based on the absolute height of the peaks and can be used in all cases to ignore globally low peaks. A second test (local.threshold) is available when the window defined by 'span' does not include all observations and can be used to ignore peaks that are not locally prominent. In this second approach the height of each peak is compared to a summary computed from other values within the window of width equal to span where it was found. In this second case, the reference value used within each window containing a peak is given by local.reference. Parameter threshold.range determines

how the bare numeric values passed as argument to global.threshold and local.threshold are scaled. The default, NULL uses the range of x. Thresholds for ignoring too small peaks are applied after peaks are searched for, and threshold values can in some cases result in no peaks being found. If either threshold is not available (NA) the returned value is a NA vector of the same length as x.

The local.threshold argument is used *as is* when local.reference is "median" or "farthest", i.e., the same distance between peak and reference is used as cut-off irrespective of the value of the reference. In cases when the prominence of peaks is positively correlated with the baseline, a local.threshold that increases together with increasing computed within window median or farthest value applies apply a less stringent height requirement in regions with overall low height. In this case, natural logarithm or square root weighting can be requested with 'local.reference' arguments '"median.log"', '"farthest.log"', '"median.sqrt"', and '"farthest.sqrt"' as arguments for local.reference.

Value

A vector of logical values of the same length as x. Values that are TRUE correspond to local peaks in vector x and can be used to extract the rows corresponding to peaks from a data frame.

Note

The default for parameter strict is FALSE in functions find_peaks() and find_valleys(), while it is strict = TRUE in peaks.

See Also

peaks.

Other peaks and valleys functions: find_spikes()

Examples

```
# lynx is a time.series object
lynx_num.df <-
  try_tibble(lynx,
             col.names = c("year", "lynx"),
             as.numeric = TRUE) # years -> as numeric

which(find_peaks(lynx_num.df$lynx, span = 5))
which(find_valleys(lynx_num.df$lynx, span = 5))
lynx_num.df[find_peaks(lynx_num.df$lynx, span = 5), ]
lynx_num.df[find_peaks(lynx_num.df$lynx, span = 51), ]
lynx_num.df[find_peaks(lynx_num.df$lynx, span = NULL), ]
lynx_num.df[find_peaks(lynx_num.df$lynx,
                       span = 15,
                       global.threshold = 2/3), ]
lynx_num.df[find_peaks(lynx_num.df$lynx,
                       span = 15,
                       global.threshold = I(4000)), ]
lynx_num.df[find_peaks(lynx_num.df$lynx,
                       span = 15,
                       local.threshold = 0.5), ]
```

`find_quantiles` *Find the location of quantiles*

Description

Find the location of quantiles in an univariate density function, such as within a call to `aes()`, to map regions under the curve that the quantiles delimit to the `fill` aesthetic in a ggplot layer.

Usage

```
find_quantiles(density, quantiles = c(0.025, 0.975), group = 1)
```

Arguments

`density`	numeric A fitted density function predicted at a large number of equal steps without truncation.
`quantiles`	numeric The probabilities of the quantiles to locate.
`group`	a 'factor' in the sense that `as.factor(f)` defines the grouping, or a list of such factors in which case their interaction is used for the grouping, used in a call to `split()`.

Details

A running cumulated density function (CDF) is computed as `cumsum(density)` assuming that `x` steps are uniform in size. Values are subsequently compared to the target quantiles, to identify the regions they delimit. No interpolation is done making it crucial that `density` is a long vector, as controlled by parameter `n` in `stat_density()`.

Unique and sorted values from the argument passed to `quantiles` are used. Values 0 and 1 are added if not present, thus, the number of regions returned is always one less than the length of these "normalized" quantiles.

In 'ggplot2' (>= 4.0.0), `geom_area()` when the `fill` aesthetic is mapped to a numeric variable or to a factor, the fill is rendered as a gradient, making it possible to highlight multiple quantiles within a single plot layer. *Gradient fills are supported in R (>= 4.1.0) and only by some graphic devices. This function is of use only if gradient fills are supported!* The capabilities of the currently active device can be tested with a call to `dev.capabilities()` checking the field `"patterns"`.

Function `find_quantiles()` is used in `stat_distrmix_line()` and `stat_distrmix_area()` to tag the regions limited by `quantiles`. If used on its own to create a mapping in a call to `aes()`, data groupings present in the ggplot must be described by the argument passed to `group`.

The approach used is approximate and relies on assumptions that are known to be fulfiled by the `density` *estimates returned by specific 'ggplot2' stats such as* `stat_density()`.

Value

A factor with levels indicating the regions delimited by the quantiles. The levels are labelled by ordinal numbers.

Note

The approach used in `find_quantiles()` is very different to that used in package 'ggdensity', which is based on the local density rather than the accumulated one.

Examples

```
# No grouping
ggplot(diamonds, aes(carat)) +
  stat_density(
    geom = "area", outline.type = "upper", colour = "black",
    aes(fill = after_stat(find_quantiles(density) != 2))) +
  scale_fill_grey(guide = "none", end = 0.3, start = 0.7) +
  expand_limits(x = 0)

# a grouping from the mapping of cut to the fill aesthetic
ggplot(diamonds, aes(carat, fill = cut)) +
  stat_density(
    geom = "area", outline.type = "upper", colour = "black",
    position = "stack",
    aes(alpha =
      after_stat(find_quantiles(density, c(0.1, 0.9), group) != 2)),
    show.legend = FALSE) +
  expand_limits(x = c(0, 5.5))
```

`find_spikes` *Find spikes in vector*

Description

Find spikes in a numeric vector using the algorithm of Whitaker and Hayes (2018). Spikes are values in spectra that are unusually high or low compared to neighbours. They are usually individual values or very short runs of similar "unusual" values. Spikes caused by cosmic radiation are a frequent problem in Raman spectra. Another source of spikes are "hot pixels" in CCD and diode arrays. Other kinds of accidental "outliers" can be also detected.

Usage

```
find_spikes(
  x,
  x.is.delta = FALSE,
  height.threshold = 10,
  z.threshold = 5,
  k = 20,
  spike.direction = "both",
  na.rm = FALSE
)
```

Arguments

x	numeric vector containing the data.
x.is.delta	logical Flag indicating whether x contains differences or original values.
height.threshold	numeric The minimum height of spikes expressed relative to the median amplitude of the baseline local variation of x.
z.threshold	numeric Modified local Z values larger than z.threshold are detected as boundaries of spikes.
k	integer width of median window used for smoothing; must be odd
spike.direction	character One of "up", "down", "both" or "skip", indicating which spikes are to be returned, if any.
na.rm	logical indicating whether NA values should be stripped before searching for spikes.

Details

Spikes are detected based on a modified Z score calculated from the differenced spectrum. The Z threshold used should be adjusted to the characteristics of the input and desired sensitivity. The lower the threshold the more stringent the test becomes, with shorter spikes being detected.

The algorithm uses running differences to detect abrupt changes in value, compared to an estimate of the baseline variation of the differences, approximating a baseline Z from MAD and a baseline value from the median differences. Currently, a single estimate of MAD is used but running medians, when posisble, as baseline. This comparison detects running differences that are unusually large, in most cases signalling a transition between values near the baseline and far from it, in both directions.

Transitions into- and out of spikes are distinguished based on the median of the non-differenced values, as a descriptor of the data baseline. As for the median of the differences, a running median is used when possible.

This function thus detects the start and end of each spike, and distinguishes upward and downward spikes.

k is the width in number of observations of the window used for running median smoothing to extract the baseline. A value several times the width of the broader spike but narrow enough to track broader peaks needs to be manually set in most cases.

With na.rm = TRUE, NA values are omitted before searching for spikes and set to 0L in the returned vector.

If all spikes are guaranteed to be one observation-wide and either going up or down from the baseline, it is possible to detect them based purely on the z.threshold by passing height.threshold = NA and either spike.direction = "up" or spike.direction = "down", which ensures very fast computation.

Value

An integer vector of the same length as x. Values that are 0, +1 or -1 corresponding to no-spike, upwards-spike, and downwards-spike in the data. Conversion to logical with as.logical() results in a vector with TRUE for spikes and FALSE otherwise.

References

Whitaker, D. A.; Hayes, K. (2018) A simple algorithm for despiking Raman spectra. Chemometrics and Intelligent Laboratory Systems, 179, 82-84. doi:10.1016/j.chemolab.2018.06.009.

See Also

Other peaks and valleys functions: `find_peaks()`

`keep_tidy` *Tidy, glance or augment an object keeping a trace of its origin*

Description

Methods implemented in package 'broom' to tidy, glance and augment the output from model fits return a consistently organized tibble with generic column names. Although this simplifies later steps in the data analysis and reporting, it drops key information needed for interpretation. `keep_tidy()` makes it possible to retain fields from the model fit object passed as argument to parameter `x` in the attribute `"fm"`. The class of `x` is always stored, and by default also fields `"call"`, `"terms"`, `"formula"`, `"fixed"` and `"random"` if available.

Usage

```
keep_tidy(x, ..., to.keep = c("call", "terms", "formula", "fixed", "random"))

keep_glance(x, ..., to.keep = c("call", "terms", "formula", "fixed", "random"))

keep_augment(
  x,
  ...,
  to.keep = c("call", "terms", "formula", "fixed", "random")
)
```

Arguments

`x`	An object for which `tidy()`, `glance` and/or `augment` method is available.
`...`	Other named arguments passed along to `tidy()`, `glance` or `augment`.
`to.keep`	character vector of field names in `x` to copy to attribute `"fm"` of the tibble returned by `tidy()`, `glance` or `augment`.

Details

Functions `keep_tidy()`, `keep_glance` or `keep_augment` are simple wrappers of the generic methods which make it possible to add to the returned values an attribute named `"fm"` preserving user selected fields and class of the model fit object. Fields names in `to.keep` missing in `x` are silently ignored.

Examples

```
# these examples can only be run if package 'broom' is available

if (requireNamespace("broom", quietly = TRUE)) {

  library(broom)

  mod <- lm(mpg ~ wt + qsec, data = mtcars)

  attr(keep_tidy(mod), "fm")[["class"]]
  attr(keep_glance(mod), "fm")[["class"]]
  attr(keep_augment(mod), "fm")[["class"]]

  attr(keep_tidy(summary(mod)), "fm")[["class"]]

  library(MASS)
  rmod <- rlm(mpg ~ wt + qsec, data = mtcars)
  attr(keep_tidy(rmod), "fm")[["class"]]

}
```

`outcome2factor` *Convert numeric ternary outcomes into a factor*

Description

Convert numeric ternary outcomes into a factor

Usage

```
outcome2factor(x, n.levels = 3L)

threshold2factor(x, n.levels = 3L, threshold = 0)
```

Arguments

`x`	a numeric vector of -1, 0, and +1 values, indicating down-regulation, uncertain response or up-regulation, or a numeric vector that can be converted into such values using a pair of thresholds.
`n.levels`	numeric Number of levels to create, either 3 or 2.
`threshold`	numeric vector Range enclosing the values to be considered uncertain.

Details

These functions convert the numerically encoded values into a factor with the three levels `"down"`, `"uncertain"` and `"up"`, or into a factor with two levels `de` and `uncertain` as expected by default by scales `scale_colour_outcome`, `scale_fill_outcome` and `scale_shape_outcome`. When `n.levels = 2` both -1 and +1 are merged to the same level of the factor with label `"de"`.

Note

These are convenience functions that only save some typing. The same result can be achieved by a direct call to `factor` and comparisons. These functions aim at making it easier to draw volcano and quadrant plots.

See Also

Other scales for omics data: `FC_format()`, `scale_colour_logFC()`, `scale_colour_outcome()`, `scale_shape_outcome()`, `scale_x_logFC()`, `scale_y_Pvalue()`, `xy_outcomes2factor()`

Examples

```
outcome2factor(c(-1, 1, 0, 1))
outcome2factor(c(-1, 1, 0, 1), n.levels = 2L)

threshold2factor(c(-0.1, -2, 0, +5))
threshold2factor(c(-0.1, -2, 0, +5), n.levels = 2L)
threshold2factor(c(-0.1, -2, 0, +5), threshold = c(-1, 1))
```

---

plain_label *Format numbers as character labels*

Description

These functions format numeric values as character labels including the symbol for statistical parameter estimates suitable for adding to plots. The labels can be formatted as strings to be parsed as plotmath expressions, or encoded using LaTeX or Markdown.

Usage

```
plain_label(
  value,
  value.name,
  digits = 3,
  fixed = FALSE,
  drop.trailing = FALSE,
  output.type = "expression",
  decimal.mark = getOption("OutDec", default = ".")
)

italic_label(
  value,
  value.name,
  digits = 3,
  fixed = FALSE,
  drop.trailing = FALSE,
```

```
  output.type = "expression",
  decimal.mark = getOption("OutDec", default = ".")
)

bold_label(
  value,
  value.name,
  digits = 3,
  fixed = FALSE,
  drop.trailing = FALSE,
  output.type = "expression",
  decimal.mark = getOption("OutDec", default = ".")
)

p_value_label(
  value,
  small.p = getOption("ggpmisc.small.p", default = FALSE),
  subscript = "",
  superscript = "",
  digits = 4,
  fixed = NULL,
  drop.trailing = FALSE,
  output.type = "expression",
  decimal.mark = getOption("OutDec", default = ".")
)

f_value_label(
  value,
  df1 = NULL,
  df2 = NULL,
  digits = 4,
  fixed = FALSE,
  drop.trailing = FALSE,
  output.type = "expression",
  decimal.mark = getOption("OutDec", default = ".")
)

t_value_label(
  value,
  df = NULL,
  digits = 4,
  fixed = FALSE,
  drop.trailing = FALSE,
  output.type = "expression",
  decimal.mark = getOption("OutDec", default = ".")
)

z_value_label(
```

```
  value,
  digits = 4,
  fixed = FALSE,
  drop.trailing = FALSE,
  output.type = "expression",
  decimal.mark = getOption("OutDec", default = ".")
)

S_value_label(
  value,
  digits = 4,
  fixed = FALSE,
  drop.trailing = FALSE,
  output.type = "expression",
  decimal.mark = getOption("OutDec", default = ".")
)

mean_value_label(
  value,
  digits = 4,
  fixed = FALSE,
  drop.trailing = FALSE,
  output.type = "expression",
  decimal.mark = getOption("OutDec", default = ".")
)

var_value_label(
  value,
  digits = 4,
  fixed = FALSE,
  drop.trailing = FALSE,
  output.type = "expression",
  decimal.mark = getOption("OutDec", default = ".")
)

sd_value_label(
  value,
  digits = 4,
  fixed = FALSE,
  drop.trailing = FALSE,
  output.type = "expression",
  decimal.mark = getOption("OutDec", default = ".")
)

se_value_label(
  value,
  digits = 4,
  fixed = FALSE,
```

```
  drop.trailing = FALSE,
  output.type = "expression",
  decimal.mark = getOption("OutDec", default = ".")
)

r_label(
  value,
  method = "pearson",
  small.r = getOption("ggpmisc.small.r", default = FALSE),
  digits = 3,
  fixed = TRUE,
  drop.trailing = FALSE,
  output.type = "expression",
  decimal.mark = getOption("OutDec", default = ".")
)

rr_label(
  value,
  small.r = getOption("ggpmisc.small.r", default = FALSE),
  digits = 3,
  pc.out = FALSE,
  fixed = TRUE,
  drop.trailing = FALSE,
  output.type = "expression",
  decimal.mark = getOption("OutDec", default = ".")
)

adj_rr_label(
  value,
  small.r = getOption("ggpmisc.small.r", default = FALSE),
  digits = 3,
  pc.out = FALSE,
  fixed = TRUE,
  drop.trailing = FALSE,
  output.type = "expression",
  decimal.mark = getOption("OutDec", default = ".")
)

rr_ci_label(
  value,
  conf.level,
  range.brackets = c("[", "]"),
  range.sep = NULL,
  digits = 2,
  fixed = TRUE,
  drop.trailing = FALSE,
  output.type = "expression",
  decimal.mark = getOption("OutDec", default = ".")
```

```
)

r_ci_label(
  value,
  conf.level,
  small.r = getOption("ggpmisc.small.r", default = FALSE),
  range.brackets = c("[", "]"),
  range.sep = NULL,
  digits = 2,
  fixed = TRUE,
  drop.trailing = FALSE,
  output.type = "expression",
  decimal.mark = getOption("OutDec", default = ".")
)
```

Arguments

`value`	numeric vector The value of the estimate(s), accepted vector length depends on the function.
`value.name`	character The symbol used to represent the value, or its name.
`digits`	integer Number of digits to which numeric values are formatetd.
`fixed`	logical Interpret `digits` as indicating a number of digits after the decimal mark or as the number of significant digits.
`drop.trailing`	logical If both `drop.trailing` = `TRUE` and `fixed` = `TRUE` trailing zeros are dropped.
`output.type`	character One of "expression", "latex", "tex", "text", "tikz", "markdown". "marquee".
`decimal.mark`	character Defaults to the value of R option `"OutDec"`.
`small.p`, `small.r`	logical If `TRUE` use lower case (p and r, r^2) instead of upper case (P and R, R^2),
`subscript`, `superscript`	character Text for a subscript and superscript to *P* symbol.
`df`, `df1`, `df2`	numeric The degrees of freedom of the estimate.
`method`	character The method used to estimate correlation, which selects the symbol used for the value.
`pc.out`	logical If `TRUE` format value in label as percent.
`conf.level`	numeric critical *P*-value expressed as fraction in [0..1].
`range.brackets`, `range.sep`	character Strings used to format a range.

Value

A character string with formatting, encoded to be parsed as an R plotmath expression, as plain text, as markdown or to be used with 'LaTeX' within **math mode**.

See Also

sprintf_dm

Examples

```
plain_label(value = 123, value.name = "n", output.type = "expression")
plain_label(value = 123, value.name = "n", output.type = "markdown")
plain_label(value = 123, value.name = "n", output.type = "latex")
italic_label(value = 123, value.name = "n", output.type = "expression")
italic_label(value = 123, value.name = "n", output.type = "markdown")
italic_label(value = 123, value.name = "n", output.type = "latex")
bold_label(value = 123, value.name = "n", output.type = "expression")
bold_label(value = 123, value.name = "n", output.type = "markdown")
bold_label(value = 123, value.name = "n", output.type = "latex")

plain_label(value = NA, value.name = "n", output.type = "expression")
plain_label(value = c(123, NA), value.name = "n", output.type = "latex")

plain_label(value = c(123, 1.2), value.name = "n", output.type = "expression")
plain_label(value = c(123, 1.2), value.name = "n", output.type = "markdown")
plain_label(value = c(123, 1.2), value.name = "n", output.type = "latex")
p_value_label(value = 0.345, digits = 2, output.type = "expression")
p_value_label(value = 0.345, digits = Inf, output.type = "expression")
p_value_label(value = 0.345, digits = 6, output.type = "expression")
p_value_label(value = 0.345, output.type = "markdown")
p_value_label(value = 0.345, output.type = "latex")
p_value_label(value = 0.345, subscript = "Holm")
p_value_label(value = 1e-25, digits = Inf, output.type = "expression")

f_value_label(value = 123.4567, digits = 2, output.type = "expression")
f_value_label(value = 123.4567, digits = Inf, output.type = "expression")
f_value_label(value = 123.4567, digits = 6, output.type = "expression")
f_value_label(value = 123.4567, output.type = "markdown")
f_value_label(value = 123.4567, output.type = "latex")
f_value_label(value = 123.4567, df1 = 3, df2 = 123,
              digits = 2, output.type = "expression")
f_value_label(value = 123.4567, df1 = 3, df2 = 123,
              digits = 2, output.type = "latex")

t_value_label(value = 123.4567, digits = 2, output.type = "expression")
t_value_label(value = 123.4567, digits = Inf, output.type = "expression")
t_value_label(value = 123.4567, digits = 6, output.type = "expression")
t_value_label(value = 123.4567, output.type = "markdown")
t_value_label(value = 123.4567, output.type = "latex")
t_value_label(value = 123.4567, df = 12,
              digits = 2, output.type = "expression")
t_value_label(value = 123.4567, df = 123,
              digits = 2, output.type = "latex")

r_label(value = 0.95, digits = 2, output.type = "expression")
r_label(value = -0.95, digits = 2, output.type = "expression")
r_label(value = 0.0001, digits = 2, output.type = "expression")
r_label(value = -0.0001, digits = 2, output.type = "expression")
r_label(value = 0.1234567890, digits = Inf, output.type = "expression")
r_label(value = 0.95, digits = 2, method = "pearson")
r_label(value = 0.95, digits = 2, method = "kendall")
```

```
r_label(value = 0.95, digits = 2, method = "spearman")

rr_label(value = 0.95, digits = 2, output.type = "expression")
rr_label(value = 0.0001, digits = 2, output.type = "expression")
rr_label(value = 1e-17, digits = Inf, output.type = "expression")

adj_rr_label(value = 0.95, digits = 2, output.type = "expression")
adj_rr_label(value = 0.0001, digits = 2, output.type = "expression")

rr_ci_label(value = c(0.3, 0.4), conf.level = 0.95)
rr_ci_label(value = c(0.3, 0.4), conf.level = 0.95, output.type = "text")
rr_ci_label(value = c(0.3, 0.4), conf.level = 0.95, range.sep = ",")

r_ci_label(value = c(-0.3, 0.4), conf.level = 0.95)
r_ci_label(value = c(-0.3, 0.4), conf.level = 0.95, output.type = "text")
r_ci_label(value = c(-0.3, 0.4), conf.level = 0.95, range.sep = ",")
r_ci_label(value = c(-1.0, 0.4), conf.level = 0.95, range.sep = ",")
```

`poly2character` *Convert a polynomial into character string*

Description

Differs from `polynom::as.character.polynomial()` in that trailing zeros are preserved.

Usage

```
poly2character(
  x,
  decreasing = getOption("ggpmisc.decreasing.poly.eq", FALSE),
  digits = 3,
  keep.zeros = TRUE
)
```

Arguments

`x`	a `polynomial` object.
`decreasing`	logical It specifies the order of the terms; in increasing (default) or decreasing powers.
`digits`	integer Giving the number of significant digits to use for printing.
`keep.zeros`	logical It indicates if zeros are to be retained in the formatted coefficients.

Value

A `character` string.

Note

This is an edit of the code in package 'polynom' so that trailing zeros are retained during the conversion. It is not defined using a different name so as not to interfere with the original.

Examples

```
poly2character(1:3)
poly2character(1:3, decreasing = TRUE)
```

---

predict.lmodel2 *Model Predictions*

Description

`predict` is a generic function for predictions from the results of various model fitting functions. `predict.lmodel2` is the method for model fit objects of class `"lmodel2"`.

Usage

```
## S3 method for class 'lmodel2'
predict(
  object,
  method = "MA",
  newdata = NULL,
  interval = c("none", "confidence"),
  level = 0.95,
  ...
)
```

Arguments

`object`	a fitted model object.
`method`	character One of the methods available in `object`.
`newdata`	An optional data frame in which to look for variables with which to predict. If omitted, the fitted values are used.
`interval`	Type of interval calculation.
`level`	the confidence level required. Currently only 0.95 accepted.
`...`	ignored by this method.

Details

Function `lmodel2()` from package 'lmodel2' returns a fitted model object of class `"lmodel2"` which differs from that returned by `lm()`. Here we implement a `predict()` method for objects of this class. It differs from the generic method and that for `lm` objects in having an additional formal parameter `method` that must be used to select which of the methods supported by `lmodel2()` are to be used in the prediction. The returned object is similar in its structure to that returned by `predict.lm()` but lacking names or rownames.

Value

If `interval = "none"` a numeric vector is returned, while if `interval = "confidence"` a data frame with columns `fit`, `lwr` and `upr` is returned.

See Also

`lmodel2`

`scale_colour_logFC` *Colour and fill scales for log fold change data*

Description

Continuous scales for `colour` and `fill` aesthetics with defaults suitable for values expressed as log2 fold change in `data` and fold-change in tick labels. Supports tick labels and data expressed in any combination of fold-change, log2 fold-change and log10 fold-change. Supports addition of units to legend title passed as argument to the `name` formal parameter.

Usage

```
scale_colour_logFC(
  name = "Abundance of y%unit",
  breaks = NULL,
  labels = NULL,
  limits = symmetric_limits,
  oob = scales::squish,
  expand = expansion(mult = 0.05, add = 0),
  log.base.labels = FALSE,
  log.base.data = 2L,
  midpoint = NULL,
  low.colour = "dodgerblue2",
  mid.colour = "grey50",
  high.colour = "red",
  na.colour = "black",
  aesthetics = "colour",
  ...
)

scale_color_logFC(
  name = "Abundance of y%unit",
  breaks = NULL,
  labels = NULL,
  limits = symmetric_limits,
  oob = scales::squish,
  expand = expansion(mult = 0.05, add = 0),
  log.base.labels = FALSE,
  log.base.data = 2L,
```

```
  midpoint = NULL,
  low.colour = "dodgerblue2",
  mid.colour = "grey50",
  high.colour = "red",
  na.colour = "black",
  aesthetics = "colour",
  ...
)

scale_fill_logFC(
  name = "Abundance of y%unit",
  breaks = NULL,
  labels = NULL,
  limits = symmetric_limits,
  oob = scales::squish,
  expand = expansion(mult = 0.05, add = 0),
  log.base.labels = FALSE,
  log.base.data = 2L,
  midpoint = 1,
  low.colour = "dodgerblue2",
  mid.colour = "grey50",
  high.colour = "red",
  na.colour = "black",
  aesthetics = "fill",
  ...
)
```

Arguments

name	The name of the scale without units, used for the legend title.
breaks	The positions of ticks or a function to generate them. Default varies depending on argument passed to log.base.labels. if supplied as a numeric vector they should be given using the data as passed to parameter data.
labels	The tick labels or a function to generate them from the tick positions. The default is function that uses the arguments passed to log.base.data and log.base.labels to generate suitable labels.
limits	limits One of: NULL to use the default scale range from ggplot2. A numeric vector of length two providing limits of the scale, using NA to refer to the existing minimum or maximum. A function that accepts the existing (automatic) limits and returns new limits. The default is function symmetric_limits() which keep 1 at the middle of the axis..
oob	Function that handles limits outside of the scale limits (out of bounds). The default squishes out-of-bounds values to the boundary.
expand	Vector of range expansion constants used to add some padding around the data, to ensure that they are placed some distance away from the axes. The default is to expand the scale by 15% on each end for log-fold-data, so as to leave space for counts annotations.

`log.base.labels`, `log.base.data`
: integer or logical Base of logarithms used to express fold-change values in tick labels and in `data`. Use `FALSE` for no logarithm transformation.

`midpoint`
: numeric Value at the middle of the colour gradient, defaults to FC = 1, assuming data is expressed as logarithm.

`low.colour`, `mid.colour`, `high.colour`, `na.colour`
: character Colour definitions to use for the gradient extremes and middle.

`aesthetics`
: Character string or vector of character strings listing the name(s) of the aesthetic(s) that this scale works with. This can be useful, for example, to apply colour settings to the colour and fill aesthetics at the same time, via aesthetics = c("colour", "fill").

`...`
: other named arguments passed to `scale_y_continuous`.

Details

These scales only alter default arguments of `scale_colour_gradient2()` and `scale_fill_gradient2()`. Please, see documentation for `scale_continuous` for details. The name argument supports the use of `"%unit"` at the end of the string to automatically add a units string, otherwise user-supplied values for names, breaks, and labels work as usual. Tick labels in the legend are built based on the transformation already applied to the data (log2 by default) and a possibly different log transformation (default is fold-change with no transformation). The default for handling out of bounds values is to "squish" them to the extreme of the scale, which is different from the default used in 'ggplot2'.

See Also

Other scales for omics data: `FC_format()`, `outcome2factor()`, `scale_colour_outcome()`, `scale_shape_outcome()`, `scale_x_logFC()`, `scale_y_Pvalue()`, `xy_outcomes2factor()`

Examples

```
set.seed(12346)
my.df <- data.frame(x = rnorm(50, sd = 4), y = rnorm(50, sd = 4))
# we assume that both x and y values are expressed as log2 fold change

ggplot(my.df, aes(x, y, colour = y)) +
  geom_point(shape = "circle", size = 2.5) +
  scale_x_logFC() +
  scale_y_logFC() +
  scale_colour_logFC()

ggplot(my.df, aes(x, y, fill = y)) +
  geom_point(shape = "circle filled", colour = "black", size = 2.5) +
  scale_x_logFC() +
  scale_y_logFC() +
  scale_fill_logFC()

my.labels <-
  scales::trans_format(function(x) {log10(2^x)}, scales::math_format())
ggplot(my.df, aes(x, y, colour = y)) +
  geom_point() +
```

```
  scale_x_logFC(labels = my.labels) +
  scale_y_logFC(labels = my.labels) +
  scale_colour_logFC(labels = my.labels)

ggplot(my.df, aes(x, y, colour = y)) +
  geom_point() +
  scale_x_logFC(log.base.labels = 2) +
  scale_y_logFC(log.base.labels = 2) +
  scale_colour_logFC(log.base.labels = 2)

ggplot(my.df, aes(x, y, colour = y)) +
  geom_point() +
  scale_x_logFC(log.base.labels = 10) +
  scale_y_logFC(log.base.labels = 10) +
  scale_colour_logFC(log.base.labels = 10)

ggplot(my.df, aes(x, y, colour = y)) +
  geom_point() +
  scale_x_logFC(log.base.labels = 10) +
  scale_y_logFC(log.base.labels = 10) +
  scale_colour_logFC(log.base.labels = 10,
                     labels = FC_format(log.base.labels = 10,
                                        log.base.data = 2L,
                                        fmt = "% .*g"))

# override default arguments.
ggplot(my.df, aes(x, y, colour = y)) +
  geom_point() +
  scale_x_logFC() +
  scale_y_logFC() +
  scale_colour_logFC(name = "Change",
                     labels = function(x) {paste(2^x, "fold")})
```

`scale_colour_outcome` *Colour and fill scales for ternary outcomes*

Description

Manual scales for colour and fill aesthetics with defaults suitable for the three way outcome from some statistical tests.

Usage

```
scale_colour_outcome(
  ...,
  name = "Outcome",
  ns.colour = "grey80",
  up.colour = "red",
  down.colour = "dodgerblue2",
```

```
  de.colour = "goldenrod",
  na.colour = "black",
  values = "outcome:updown",
  drop = TRUE,
  aesthetics = "colour"
)

scale_color_outcome(
  ...,
  name = "Outcome",
  ns.colour = "grey80",
  up.colour = "red",
  down.colour = "dodgerblue2",
  de.colour = "goldenrod",
  na.colour = "black",
  values = "outcome:updown",
  drop = TRUE,
  aesthetics = "colour"
)

scale_fill_outcome(
  ...,
  name = "Outcome",
  ns.colour = "grey80",
  up.colour = "red",
  down.colour = "dodgerblue2",
  de.colour = "goldenrod",
  na.colour = "black",
  values = "outcome:both",
  drop = TRUE,
  aesthetics = "fill"
)
```

Arguments

`...`	other named arguments passed to `scale_colour_manual`.
`name`	The name of the scale, used for the axis-label.
`ns.colour`, `down.colour`, `up.colour`, `de.colour`	The colour definitions to use for each of the three possible outcomes.
`na.colour`	colour definition used for NA.
`values`	a set of aesthetic values to map data values to. The values will be matched in order (usually alphabetical) with the limits of the scale, or with breaks if provided. If this is a named vector, then the values will be matched based on the names instead. Data values that don't match will be given na.value. In addition the special values `"outcome:updown"`, `"outcome:de"` and `"outcome:both"` set predefined values, with `"outcome:both"` as default.
`drop`	logical Should unused factor levels be omitted from the scale? The default,

TRUE, uses the levels that appear in the data; FALSE uses all the levels in the factor.

`aesthetics` Character string or vector of character strings listing the name(s) of the aesthetic(s) that this scale works with. This can be useful, for example, to apply colour settings to the colour and fill aesthetics at the same time, via aesthetics = c("colour", "fill").

Details

These scales only alter the `breaks`, `values`, and `na.value` default arguments of `scale_colour_manual()` and `scale_fill_manual()`. Please, see documentation for `scale_manual` for details.

Note

In 'ggplot2' (3.3.4, 3.3.5, 3.3.6) `scale_colour_manual()` and `scale_fill_manual()` do not obey `drop`, most likely due to a bug as this worked in version 3.3.3 and earlier. This results in spureous levels in the plot legend when using versions 3.3.4, 3.3.5, 3.3.6 of 'ggplot2'.

See Also

Other scales for omics data: `FC_format()`, `outcome2factor()`, `scale_colour_logFC()`, `scale_shape_outcome()`, `scale_x_logFC()`, `scale_y_Pvalue()`, `xy_outcomes2factor()`

Examples

```
set.seed(12346)
outcome <- sample(c(-1, 0, +1), 50, replace = TRUE)
my.df <- data.frame(x = rnorm(50),
                    y = rnorm(50),
                    outcome2 = outcome2factor(outcome, n.levels = 2),
                    outcome3 = outcome2factor(outcome))

ggplot(my.df, aes(x, y, colour = outcome3)) +
  geom_point() +
  scale_colour_outcome() +
  theme_bw()

ggplot(my.df, aes(x, y, colour = outcome2)) +
  geom_point() +
  scale_colour_outcome() +
  theme_bw()

ggplot(my.df, aes(x, y, fill = outcome3)) +
  geom_point(shape = 21) +
  scale_fill_outcome() +
  theme_bw()
```

scale_shape_outcome *Shape scale for ternary outcomes*

Description

Manual scales for colour and fill aesthetics with defaults suitable for the three way outcome from some statistical tests.

Usage

```
scale_shape_outcome(
  ...,
  name = "Outcome",
  ns.shape = "circle filled",
  up.shape = "triangle filled",
  down.shape = "triangle down filled",
  de.shape = "square filled",
  na.shape = "cross"
)
```

Arguments

...	other named arguments passed to scale_manual.
name	The name of the scale, used for the axis-label.
ns.shape, down.shape, up.shape, de.shape	The shapes to use for each of the three possible outcomes.
na.shape	Shape used for NA.

Details

These scales only alter the values, and na.value default arguments of scale_shape_manual(). Please, see documentation for scale_manual for details.

See Also

Other scales for omics data: FC_format(), outcome2factor(), scale_colour_logFC(), scale_colour_outcome(), scale_x_logFC(), scale_y_Pvalue(), xy_outcomes2factor()

Examples

```
set.seed(12346)
outcome <- sample(c(-1, 0, +1), 50, replace = TRUE)
my.df <- data.frame(x = rnorm(50),
                    y = rnorm(50),
                    outcome2 = outcome2factor(outcome, n.levels = 2),
                    outcome3 = outcome2factor(outcome))
```

```
ggplot(my.df, aes(x, y, shape = outcome3)) +
  geom_point() +
  scale_shape_outcome() +
  theme_bw()

ggplot(my.df, aes(x, y, shape = outcome3)) +
  geom_point() +
  scale_shape_outcome(guide = FALSE) +
  theme_bw()

ggplot(my.df, aes(x, y, shape = outcome2)) +
  geom_point(size = 2) +
  scale_shape_outcome() +
  theme_bw()

ggplot(my.df, aes(x, y, shape = outcome3, fill = outcome2)) +
  geom_point() +
  scale_shape_outcome() +
  scale_fill_outcome() +
  theme_bw()

ggplot(my.df, aes(x, y, shape = outcome3, fill = outcome2)) +
  geom_point() +
  scale_shape_outcome(name = "direction") +
  scale_fill_outcome(name = "significance") +
  theme_bw()
```

`scale_x_logFC` *Position scales for log fold change data*

Description

Continuous scales for x and y aesthetics with defaults suitable for values expressed as log2 fold change in `data` and fold-change in tick labels. Supports tick labels and data expressed in any combination of fold-change, log2 fold-change and log10 fold-change. Supports addition of units to axis labels passed as argument to the `name` formal parameter.

Usage

```
scale_x_logFC(
  name = "Abundance of x%unit",
  breaks = NULL,
  labels = NULL,
  limits = symmetric_limits,
  oob = scales::squish,
  expand = expansion(mult = 0.05, add = 0),
  log.base.labels = FALSE,
  log.base.data = 2L,
```

```
  ...
)

scale_y_logFC(
  name = "Abundance of y%unit",
  breaks = NULL,
  labels = NULL,
  limits = symmetric_limits,
  oob = scales::squish,
  expand = expansion(mult = 0.05, add = 0),
  log.base.labels = FALSE,
  log.base.data = 2L,
  ...
)
```

Arguments

name	The name of the scale without units, used for the axis-label.
breaks	The positions of ticks or a function to generate them. Default varies depending on argument passed to log.base.labels. if supplied as a numeric vector they should be given using the data as passed to parameter data.
labels	The tick labels or a function to generate them from the tick positions. The default is function that uses the arguments passed to log.base.data and log.base.labels to generate suitable labels.
limits	limits One of: NULL to use the default scale range from ggplot2. A numeric vector of length two providing limits of the scale, using NA to refer to the existing minimum or maximum. A function that accepts the existing (automatic) limits and returns new limits. The default is function symmetric_limits() which keep 1 at the middle of the axis..
oob	Function that handles limits outside of the scale limits (out of bounds). The default squishes out-of-bounds values to the boundary.
expand	Vector of range expansion constants used to add some padding around the data, to ensure that they are placed some distance away from the axes. The default is to expand the scale by 15% on each end for log-fold-data, so as to leave space for counts annotations.
log.base.labels, log.base.data	integer or logical Base of logarithms used to express fold-change values in tick labels and in data. Use FALSE for no logarithm transformation.
...	other named arguments passed to scale_y_continuous.

Details

These scales only alter default arguments of scale_x_continuous() and scale_y_continuous(). Please, see documentation for scale_continuous for details. The name argument supports the use of "%unit" at the end of the string to automatically add a units string, otherwise user-supplied values for names, breaks, and labels work as usual. Tick labels are built based on the transformation already applied to the data (log2 by default) and a possibly different log transformation (default is

fold-change with no transformation). The default for handling out of bounds values is to "squish" them to the extreme of the scale, which is different from the default used in 'ggplot2'.

See Also

Other scales for omics data: FC_format(), outcome2factor(), scale_colour_logFC(), scale_colour_outcome(), scale_shape_outcome(), scale_y_Pvalue(), xy_outcomes2factor()

Examples

```
set.seed(12346)
my.df <- data.frame(x = rnorm(50, sd = 4), y = rnorm(50, sd = 4))
# we assume that both x and y values are expressed as log2 fold change

ggplot(my.df, aes(x, y)) +
  geom_point() +
  scale_x_logFC() +
  scale_y_logFC()

ggplot(my.df, aes(x, y)) +
  geom_point() +
  scale_x_logFC(labels = scales::trans_format(function(x) {log10(2^x)},
                         scales::math_format())) +
  scale_y_logFC(labels = scales::trans_format(function(x) {log10(2^x)},
                         scales::math_format()))

ggplot(my.df, aes(x, y)) +
  geom_point() +
  scale_x_logFC(log.base.labels = 2) +
  scale_y_logFC(log.base.labels = 2)

ggplot(my.df, aes(x, y)) +
  geom_point() +
  scale_x_logFC("A concentration%unit", log.base.labels = 10) +
  scale_y_logFC("B concentration%unit", log.base.labels = 10)

ggplot(my.df, aes(x, y)) +
  geom_point() +
  scale_x_logFC("A concentration%unit", breaks = NULL) +
  scale_y_logFC("B concentration%unit", breaks = NULL)

# taking into account that data are expressed as log2 FC.
ggplot(my.df, aes(x, y)) +
  geom_point() +
  scale_x_logFC("A concentration%unit", breaks = log2(c(1/100, 1, 100))) +
  scale_y_logFC("B concentration%unit", breaks = log2(c(1/100, 1, 100)))

ggplot(my.df, aes(x, y)) +
  geom_point() +
  scale_x_logFC(labels = scales::trans_format(function(x) {log10(2^x)},
                         scales::math_format())) +
  scale_y_logFC(labels = scales::trans_format(function(x) {log10(2^x)},
```

```
                        scales::math_format()))

# override "special" default arguments.
ggplot(my.df, aes(x, y)) +
  geom_point() +
  scale_x_logFC("A concentration",
                breaks = waiver(),
                labels = waiver()) +
  scale_y_logFC("B concentration",
                breaks = waiver(),
                labels = waiver())

ggplot(my.df, aes(x, y)) +
  geom_point() +
  scale_x_logFC() +
  scale_y_logFC() +
  geom_quadrant_lines() +
  stat_quadrant_counts(size = 3.5)
```

`scale_y_Pvalue` *Convenience scale for P-values*

Description

Scales for `x` and `y` aesthetics mapped to *P*-values and false discovery rates (FDR), suitable for volcano plots as used for transcriptomics and metabolomics data.

Usage

```
scale_y_Pvalue(
  ...,
  name = expression(italic(P) - plain(value)),
  transform = NULL,
  breaks = NULL,
  labels = NULL,
  limits = c(1, 1e-20),
  oob = NULL,
  expand = NULL
)

scale_y_FDR(
  ...,
  name = "False discovery rate",
  transform = NULL,
  breaks = NULL,
  labels = NULL,
  limits = c(1, 1e-10),
```

```
  oob = NULL,
  expand = NULL
)

scale_x_Pvalue(
  ...,
  name = expression(italic(P) - plain(value)),
  transform = NULL,
  breaks = NULL,
  labels = NULL,
  limits = c(1, 1e-20),
  oob = NULL,
  expand = NULL
)

scale_x_FDR(
  ...,
  name = "False discovery rate",
  transform = NULL,
  breaks = NULL,
  labels = NULL,
  limits = c(1, 1e-10),
  oob = NULL,
  expand = NULL
)
```

Arguments

`...`	other named arguments passed to `scale_y_continuous`.
`name`	The name of the scale without units, used for the axis-label.
`transform`	Either the name of a transformation object, or the object itself. Use NULL for the default.
`breaks`	The positions of ticks or a function to generate them. Default varies depending on argument passed to `log.base.labels`.
`labels`	The tick labels or a function to generate them from the tick positions. The default is function that uses the arguments passed to `log.base.data` and `log.base.labels` to generate suitable labels.
`limits`	Use one of: `NULL` to use the default scale range, a numeric vector of length two providing limits of the scale; `NA` to refer to the existing minimum or maximum; a function that accepts the existing (automatic) limits and returns new limits.
`oob`	Function that handles limits outside of the scale limits (out of bounds). The default squishes out-of-bounds values to the boundary.
`expand`	Vector of range expansion constants used to add some padding around the data, to ensure that they are placed some distance away from the axes. The default is to expand the scale by 15% on each end for log-fold-data, so as to leave space for counts annotations.

Details

These scales only reaplace default arguments of `scale_x_continuous()` and `scale_y_continuous()`. Please, see documentation for `scale_continuous` for details.

These scales set transformations for suitable for plotting log-*P*-value, log-fold-change and FDR (false discovery rate) and matching tick labels (`breaks`and `labels` and scale `names` (axis titles).

See Also

Other scales for omics data: `FC_format()`, `outcome2factor()`, `scale_colour_logFC()`, `scale_colour_outcome()`, `scale_shape_outcome()`, `scale_x_logFC()`, `xy_outcomes2factor()`

Examples

```
set.seed(12346)
my.df <- data.frame(x = rnorm(50, sd = 4),
                    y = 10^-runif(50, min = 0, max = 20))

ggplot(my.df, aes(x, y)) +
  geom_point() +
  scale_x_logFC() +
  scale_y_Pvalue()

ggplot(my.df, aes(x, y)) +
  geom_point() +
  scale_x_logFC() +
  scale_y_FDR(limits = c(NA, 1e-20))
```

---

sprintf_dm *Format numeric values as strings*

Description

Using `sprintf` flexibly format numbers as character strings encoded for parsing into R expressions or using LaTeX or markdown notation.

Usage

```
sprintf_dm(fmt, ..., decimal.mark = getOption("OutDec", default = "."))

value2char(
  value,
  digits = Inf,
  format = "g",
  drop.trailing = FALSE,
  output.type = "expression",
  decimal.mark = getOption("OutDec", default = ".")
)
```

Arguments

fmt	character as in sprintf().
...	as in sprintf().
decimal.mark	character If NULL or NA no substitution is attempted and the value returned by sprintf() is returned as is.
value	numeric The value of the estimate.
digits	integer Number of digits to which numeric values are formatted.
format	character One of "e", "f" or "g" for exponential, fixed, or significant digits formatting.
drop.trailing	logical If drop.trailing = TRUE trailing zeros are dropped with format = "f" by not adding # to the format string.
output.type	character One of "expression", "latex", "tex", "text", "tikz", "markdown", "marquee".

Details

These functions are used to format the character strings returned, which can be used as labels in plots. Encoding used for the formatting is selected by the argument passed to output.type, thus, supporting different R graphic devices.

See Also

sprintf

Examples

```
sprintf_dm("%2.3f", 2.34)
sprintf_dm("%2.3f", 2.34, decimal.mark = ",")


value2char(2.34)
value2char(2.34, digits = 3, format = "g")
value2char(2.34, digits = 3, format = "f")
value2char(2.34, output.type = "text")
value2char(2.34, output.type = "text", format = "f")
value2char(2.34, output.type = "text", format = "g")
```

---

stat_correlation *Correlation test annotations*

Description

Statistic stat_correlation() applies stats::cor.test() respecting grouping with method = "pearson" default but alternatively using "kendall" or "spearman" methods. It adds textual labels to a plot.

Usage

```
stat_correlation(
  mapping = NULL,
  data = NULL,
  geom = "text_npc",
  position = "identity",
  ...,
  method = "pearson",
  n.min = 2L,
  alternative = "two.sided",
  exact = NULL,
  r.conf.level = ifelse(method == "pearson", 0.95, NA),
  continuity = FALSE,
  fit.seed = NA,
  small.r = getOption("ggpmisc.small.r", default = FALSE),
  small.p = getOption("ggpmisc.small.p", default = FALSE),
  coef.keep.zeros = TRUE,
  r.digits = 2,
  t.digits = 3,
  p.digits = 3,
  CI.brackets = c("[", "]"),
  label.x = "left",
  label.y = "top",
  hstep = 0,
  vstep = NULL,
  output.type = NULL,
  boot.R = ifelse(method == "pearson", 0, 999),
  na.rm = FALSE,
  parse = NULL,
  show.legend = FALSE,
  inherit.aes = TRUE
)
```

Arguments

mapping	The aesthetic mapping, usually constructed with aes(). Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset, only needed if you want to override the plot defaults.
geom	The geometric object to use display the data
position	The position adjustment to use for overlapping points on this layer.
...	other arguments passed on to layer. This can include aesthetics whose values you want to set, not map. See layer for more details.
method	character One of "pearson", "kendall" or "spearman".
n.min	integer Minimum number of distinct values in the explanatory variable (on the rhs of formula) for fitting to the attempted.
alternative	character One of "two.sided", "less" or "greater".

exact — logical Whether an exact p-value should be computed. Used for Kendall's tau and Spearman's rho.

r.conf.level — numeric Confidence level for the returned confidence interval. If set to NA computation of CI is skipped.

continuity — logical If TRUE , a continuity correction is used for Kendall's tau and Spearman's rho when not computed exactly.

fit.seed — RNG seed argument passed to set.seed(). Defaults to NA, indicating that set.seed() should not be called.

small.r, small.p — logical Flags to switch use of lower case r and p for coefficient of correlation (only for method = "pearson") and p-value.

coef.keep.zeros — logical Keep or drop trailing zeros when formatting the correlation coefficients and t-value, z-value or S-value (see note below).

r.digits, t.digits, p.digits — integer Number of digits after the decimal point to use for R, r.squared, tau or rho and P-value in labels. If Inf, use exponential notation with three decimal places.

CI.brackets — character vector of length 2. The opening and closing brackets used for the CI label.

label.x, label.y — numeric with range 0..1 "normalized parent coordinates" (npc units) or character if using geom_text_npc() or geom_label_npc(). If using geom_text() or geom_label() numeric in native data units. If too short they will be recycled.

hstep, vstep — numeric in npc units, the horizontal and vertical step used between labels for different groups.

output.type — character One of "expression", "text", "markdown", "marquee", "latex", "latex.eqn", "latex.deqn" or "numeric".

boot.R — interger The number of bootstrap resamples. Set to zero for no bootstrap estimates for the CI.

na.rm — a logical indicating whether NA values should be stripped before the computation proceeds.

parse — logical Passed to the geom. If TRUE, the labels will be parsed into expressions and displayed as described in plotmath. Default is TRUE if output.type = "expression" and FALSE otherwise.

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.

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.

Details

This statistic can be used to annotate a plot with the correlation coefficient and the outcome of its test of significance. It supports Pearson, Kendall and Spearman methods to compute correlation. This

statistic generates labels as R expressions by default but LaTeX (use TikZ device), markdown (use package 'ggtext') and plain text are also supported, as well as numeric values for user-generated text labels. The character labels include the symbol describing the quantity together with the numeric value. For the confidence interval (CI) the default is to follow the APA recommendation of using square brackets. As the CI is computed by bootstrapping, `fit.seed` if different to `NA` immediately before this computation.

The value of `parse` is set automatically based on `output-type`, but if you assemble labels that need parsing from `numeric` output, the default needs to be overridden. By default the value of `output.type` is guessed from the name of the geometry.

A ggplot statistic receives as `data` a data frame that is not the one passed as argument by the user, but instead a data frame with the variables mapped to aesthetics. `cor.test()` is always applied to the variables mapped to the `x` and `y` aesthetics, so the scales used for `x` and `y` should both be continuous scales rather than discrete.

Computed variables

If output.type is `"numeric"` the returned tibble contains the columns listed below with variations depending on the `method`. If the model fit function used does not return a value, the variable is set to `NA_real_`.

x,npcx x position

y,npcy y position

r, and cor, tau or rho numeric values for correlation coefficient estimates

t.value and its df, z.value or S.value numeric values for statistic estimates

p.value, n numeric values.

r.conf.level numeric value, as fraction of one.

r.confint.low Confidence interval limit for `r`.

r.confint.high Confidence interval limit for `r`.

grp.label Set according to mapping in `aes`.

method.label Set according `method` used.

method, test character values

If output.type different from `"numeric"` the returned tibble contains in addition to the columns listed above those listed below. If the numeric value is missing the label is set to `character(0L)`.

r.label, and cor.label, tau.label or rho.label Correlation coefficient as a character string.

t.value.label, z.value.label or S.value.label t-value and degrees of freedom, z-value or S-value as a character string.

p.value.label P-value for test against zero, as a character string.

r.confint.label, and cor.conint.label, tau.confint.label or rho.confint.label Confidence interval for `r` (only with `method = "pearson"`).

n.label Number of observations used in the fit, as a character string.

grp.label Set according to mapping in `aes`, as a character string.

To explore the computed values returned for a given input we suggest the use of `geom_debug` as shown in the last examples below.

Position of labels

When data are grouped by mapping a factor to an aesthetic, e.g., `colour`, `shape` and/or `linetype` the model is fitted separately to each group, and for each group a whole set of labels is generated. If the argument passed to `label.y` is a vector of length 1, this value determines the position of the equation and/or other labels for the first group, and the positions of the labels for the remaining groups are generated by adding `vspace` based on the group number. If the argument passed to `label.y` is a vector of length > 1, it is used unchanged, possibly extended by recycling, ignoring `vstep`.

If the labels are rotated by 90 degrees then the automatic stepping is best based on `hstep` with `vstep = 0`. Similarly as described above, if `label.x` is a vector of length > 1, it is used unchanged, possibly extended by recycling, ignoring `hstep`.

When using facets and with a grouping that does not repeat in each panel, the automatic positioning in most cases will not be the desired one. Manual positioning using a vector of length > 1 for `label.x` and/or `label.y` is the currently available workaround.

Output types

The formatting of character strings to be displayed in plots are marked as mathematical equations. Depending on the geom used, the mark-up needs to be encoded differently, or in some cases mark-up not applied.

`"expression"` The labels are encoded as character strings to be parsed into R's plotmath expressions.

`"LaTeX"`, `"TeX"`, `"tikz"`, `"latex"` The labels are encoded as 'LaTeX' maths equations, without the "fences" for switching in math mode.

`"latex.eqn"` Same as `"latex"` but enclosed in single `$`, i.e., as in-line maths.

`"latex.deqn"` Same as `"latex"` but enclosed in double `$$`, i.e., as display maths.

`"markdown"` The labels are encoded as character strings using markdown syntax, with some embedded HTML.

`"marquee"` The labels are encoded as character strings using markdown syntax, with 'marquee' supported spans.

`"text"` The labels are plain ASCII character strings.

`"numeric"` No labels are generated. This value is accepted by the statistics, but not by the label formatting functions.

`NULL` The value used depends on the argument passed to `geom`.

If `geom = "latex"` (package 'xdvir') the output type used is `"latex.eqn"`. If `geom = "richtext"` (package 'ggtext') or `geom = "textbox"` (package 'ggtext') the output type used is `"markdown"`. If `geom = "marquee"` (package 'marquee') the output type used is `"marquee"`. For all other values of `geom` the default is `"expression"`. Invalid values as argument trigger an error.

Which variables are available for mapping?

Computed variables and their names can vary depending on the `method` used to fit a model or the `output.type` in use. They can also depend for a given `method` on other arguments passed when

fitting a model or extracting estimates and other computed values. In many cases, when values are not available, the variables are filled with NA values.

In the statistics returning formatted strings for use as annotations, a message is issued by default in interactive R sessions, listing the short names for available formatted labels as recognized by functions use_label() and f_use_label(), except when output.type = "numeric" is passed, in which case the names of all variables accessible by after_stat() within a call to aes() are listed. This default ("nicknames") can be changed by setting R option "ggpmisc.stat.vars.message" to one of "names", "colnames" or "none".

In the statistics that plot a prediction or more generally mainly return numeric variables, a message is issued by default in interactive R sessions, listing the names of all variables accessible by after_stat() within a call to aes() with at least some non-missing values. This default ("colnames") can be changed by setting R option "ggpmisc.stat.vars.message" to "none".

Aesthetics

stat_correlation() understands the following aesthetics. Required aesthetics are displayed in bold and defaults are displayed for optional aesthetics:

- x
- y
- group → inferred
- grp.label
- hjust → "inward"
- label → after_stat(r.label)
- npcx → after_stat(npcx)
- npcy → after_stat(npcy)
- vjust → "inward"

Learn more about setting these aesthetics in vignette("ggplot2-specs").

Note

Currently coef.keep.zeros is ignored, with trailing zeros always retained in the character labels returned but not protected from being dropped by R when these character strings are parsed into plotmath expressions (i.e., when output.type = "expression").

See Also

cor.test for details on the computations.

Examples

```
# generate artificial data
set.seed(4321)
x <- (1:100) / 10
y <- x + rnorm(length(x))
my.data <- data.frame(x = x,
                      y = y,
```

```
                      y.desc = - y,
                      group = c("A", "B"))

# by default only R is displayed
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_correlation()

ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_correlation(small.r = TRUE)

ggplot(my.data, aes(x, y.desc)) +
  geom_point() +
  stat_correlation(label.x = "right")

# non-default methods
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_correlation(method = "kendall")

ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_correlation(method = "spearman")

# use_label() can map a user selected label
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_correlation(use_label("R2"))

# use_label() can assemble and map a combined label
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_correlation(use_label("R", "P", "n", "method"))

ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_correlation(use_label("R", "R.CI"))

ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_correlation(use_label("R", "R.CI"),
                   r.conf.level = 0.95)

ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_correlation(use_label("R", "R.CI"),
                   method = "kendall",
                   r.conf.level = 0.95)

ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_correlation(use_label("R", "R.CI"),
```

```
                  method = "spearman",
                  r.conf.level = 0.95)

# f_use_label() provides additional flexibility
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_correlation(
    f_use_label("R", "R.CI",
      format = "\"Estimate: \"*%s*\" with confidence \"*%s"),
      method = "spearman", r.conf.level = 0.95)

# manually assemble and map a specific label using paste() and aes()
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_correlation(aes(label =
                         after_stat(
                           paste(r.label, p.value.label, n.label,
                                 sep = "*\", \"*"))))

# manually format and map a specific label using sprintf() and aes()
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_correlation(aes(label =
                         after_stat(
                           sprintf("%s*\" with \"*%s*\" for \"*%s",
                                   r.label, p.value.label, t.value.label))))

# Inspecting the returned data using geom_debug_group()
# This provides a quick way of printing the returned data frame.

gginnards.installed <- requireNamespace("gginnards", quietly = TRUE)

if (gginnards.installed)
  library(gginnards)

# the whole of computed data
if (gginnards.installed)
  ggplot(my.data, aes(x, y)) +
    geom_point() +
    stat_correlation(geom = "debug_group")

if (gginnards.installed)
  ggplot(my.data, aes(x, y)) +
    geom_point() +
    stat_correlation(geom = "debug_group", method = "pearson")

if (gginnards.installed)
  ggplot(my.data, aes(x, y)) +
    geom_point() +
    stat_correlation(geom = "debug_group", output.type = "numeric")

if (gginnards.installed)
  ggplot(my.data, aes(x, y)) +
```

```
    geom_point() +
    stat_correlation(geom = "debug_group", output.type = "markdown")

if (gginnards.installed)
  ggplot(my.data, aes(x, y)) +
    geom_point() +
    stat_correlation(geom = "debug_group", output.type = "LaTeX")
```

`stat_distrmix_eq` *Mixture model prediction and annotations*

Description

Fit an univariate mixture of Normals model. `stat_distrmix_line()` and `stat_distrmix_area()` add a layer with prediction lines or areas, respectively, while `stat_distrmix_eq()` adds a layer with textual labels of parameter estimates.

Usage

```
stat_distrmix_eq(
  mapping = NULL,
  data = NULL,
  geom = "text_npc",
  position = "identity",
  ...,
  orientation = NA,
  method = "normalmixEM",
  method.args = list(),
  n.min = 10L * k,
  level = 0.95,
  k = 2,
  free.mean = TRUE,
  free.sd = TRUE,
  se = FALSE,
  fit.seed = NA,
  fm.values = TRUE,
  components = NULL,
  eq.with.lhs = TRUE,
  eq.digits = 2,
  label.x = "left",
  label.y = "top",
  hstep = 0,
  vstep = NULL,
  output.type = NULL,
  na.rm = FALSE,
  parse = NULL,
  show.legend = NA,
```

```
  inherit.aes = TRUE
)

stat_distrmix_line(
  mapping = NULL,
  data = NULL,
  geom = "line",
  position = "identity",
  ...,
  orientation = NA,
  method = "normalmixEM",
  se = NULL,
  quantiles = NA,
  fit.seed = NA,
  fm.values = FALSE,
  n = 500,
  fullrange = TRUE,
  level = 0.95,
  method.args = list(),
  k = 2,
  free.mean = TRUE,
  free.sd = TRUE,
  components = "all",
  n.min = 10L * k,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

stat_distrmix_area(
  mapping = NULL,
  data = NULL,
  geom = "area",
  position = "identity",
  ...,
  orientation = NA,
  method = "normalmixEM",
  se = NULL,
  quantiles = NA,
  fit.seed = NA,
  fm.values = FALSE,
  n = 500,
  fullrange = TRUE,
  level = 0.95,
  method.args = list(),
  k = 2,
  free.mean = TRUE,
  free.sd = TRUE,
```

```
  components = "sum",
  n.min = 10L * k,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	The aesthetic mapping, usually constructed with aes(). Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset, only needed if you want to override the plot defaults.
geom	The geometric object to use display the data
position	The position adjustment to use for overlapping points on this layer.
...	other arguments passed on to layer. This can include aesthetics whose values you want to set, not map. See layer for more details.
orientation	character Either "x" or "y" controlling the aesthetic to which the density model is fit. With the default orientation = NA the orientation used is based on the mapping and nearly always correct.
method	function or character If character, "normalmixEM" or the name of a model fit function are accepted, possibly followed by the fit function's method argument separated by a colon. The function must return a model fit object of class "mixEM".
method.args	named list with additional arguments. Not data or weights which are always passed through aesthetic mappings.
n.min	integer Minimum number of distinct values in the variable for fitting to the attempted. The default depends on k.
level	Level of confidence interval to use (0.95 by default).
k	integer Number of mixture components to fit.
free.mean, free.sd	logical If TRUE, allow the fitted mean and/or fitted sd to vary among the component Normal distributions.
se	logical If TRUE standard errors for the parameter estimates are returned in addition to the parameter estimates.
fit.seed	RNG seed argument passed to set.seed(). Defaults to NA, indicating that set.seed() should not be called.
fm.values	logical Add parameter estimates and their standard errors to the returned values ('FALSE' by default.)
components	character One of "all", "sum", or "members" select which densities are returned.
eq.with.lhs	If character the string is pasted to the front of the equation label before parsing or a logical (see note).
eq.digits	integer Number of digits after the decimal point to use for parameters in labels. If Inf, use exponential notation with three decimal places.

label.x, label.y
: numeric with range 0..1 "normalized parent coordinates" (npc units) or character if using geom_text_npc() or geom_label_npc(). If using geom_text() or geom_label() numeric in native data units. If too short they will be recycled.

hstep, vstep
: numeric in npc units, the horizontal and vertical step used between labels for different groups.

output.type
: character One of "expression", "text", "markdown", "marquee", "latex", "latex.eqn", "latex.deqn" or "numeric".

na.rm
: a logical indicating whether NA values should be stripped before the computation proceeds.

parse
: logical Passed to the geom. If TRUE, the labels will be parsed into expressions and displayed as described in plotmath. Default is TRUE if output.type = "expression" and FALSE otherwise.

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.

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.

quantiles
: numeric The boundaries for quantiles given by *P*-values in the range 0 to 1, with NULL being equivalent to quantiles = c(0, 1).

n
: Number of points at which to predict with the fitted model.

fullrange
: logical Should the fit prediction span the full range of the plot, or just the range of the explanatory variable?

Details

stat_distrmix_line() and stat_distrmix_area() are similar to stat_density but they fit the Normal distribution to observations. In addition to a single Normal distribution they can fit a mixture of two or more Normal distributions, using an approach related to clustering. Defaults related to how fitting is done are consistent between stat_distrmix_line(), stat_distrmix_area() and stat_distrmix_eq(). stat_distrmix_eq() can be used to add matched textual annotations, while stat_distrmix_line() and stat_distrmix_area() only differ in their default arguments, including the geom.

If k >= 2 a mixture-of-Normals model is fitted with normalmixEM(), while if k == 1 a single Normal distribution is fitted with function fitdistr(). Only for k == 1 the SE values are exact estimates.

In stat_distrmix_line(), predictions are computed to cover >= 0.999 of the integral in all cases, trimming to the range of the data with fullrange = FALSE is done as the last step in the computations. This ensures correct estimates of the cumulated density (CDF) and of quantiles, whose locations are estimated based on the CDF.

Parameter fit.seed if not NA is used in a call to set.seed() immediately before calling the model fit function. As the fitting procedure makes use of the (pseudo-)random number generator (RNG), convergence can depend on it, and in such cases setting fit.seed to the same value in stat_distrmix_line() and in stat_distrmix_eq() can ensure consistency, and more generally, reproducibility.

The minimum number of observations with distinct values in the data variable can be set through parameter `n.min`. The default depends on `k`, the number of components in the mix. Model fits with too few observations are unreliable, thus, using larger values of `n.min` than the default is wise. The value of `n`, instead, sets the number of predicted values, which affects the smoothness of the plotted curve and the accuracy with which the location of the quantiles is predicted.

Value

The value returned by `stat_distrmix_line()` and by `stat_distrmix_area()` is a data frame, with `n` predictions values or possibly fewer in the case when `fullrange = FALSE`. Predictions for each component of the mixture plus for their sum are returned in long form.

The value returned by `stat_distrmix_eq()` is a data frame, with one row of estimates for each group of data in the plot.

Variables computed by `stat_distrmix_line()`

Some of the variables can have missing values or depend on `orientation` and/or `method`.

x `n` or fewer values

component A factor indexing the components and/or their sum

density predicted density values

quant.splits integer, numbering sequentially the regions separated by the the `quantiles`

.

If `fm.values = TRUE` is passed then columns with diagnosis and parameters estimates are added, with the same value in each row within a group:

converged `logical` indicating if convergence was achieved

n `numeric` the number of `x` values

.size `numeric` the number of `density` values

fm.class `character` the most derived class of the fitted model object

fm.method `character` the method, as given by the `ft` field of the fitted model objects

This provides a simple and robust approach to achieve effects like colouring or hiding annotations by group depending on the outcome of model fitting.

Variables computed by `stat_distrmix_eq()`

Some of the variables depend on the orientation:

x the location of text labels

y the location of text labels

eq.label `character` string for equations

n.label `character` string for number of observations

method.label `character` string for model fit method

lambda `numeric` the estimate of the contribution of the component of the mixture towards the joint density

mu numeric the estimate of the mean

sigma numeric the estimate of the standard deviation

component A factor indexing the components of the mixture and/or their sum

If se = TRUE is passed then columns with standard errors for the parameter estimates are also returned:

lambda.se numeric the estimate of the contribution of the component of the mixture towards the joint density

mu.se numeric the estimate of the mean

sigma.se numeric the estimate of the standard deviation

If fm.values = TRUE is passed then columns with diagnosis and parameters estimates are added:

converged logical indicating if convergence was achieved

n numeric the number of x values

.size numeric the number of density values

fm.class character the most derived class of the fitted model object

fm.method character the method, as given by the ft field of the fitted model objects

Output types

The formatting of character strings to be displayed in plots are marked as mathematical equations. Depending on the geom used, the mark-up needs to be encoded differently, or in some cases mark-up not applied.

"expression" The labels are encoded as character strings to be parsed into R's plotmath expressions.

"LaTeX", "TeX", "tikz", "latex" The labels are encoded as 'LaTeX' maths equations, without the "fences" for switching in math mode.

"latex.eqn" Same as "latex" but enclosed in single $, i.e., as in-line maths.

"latex.deqn" Same as "latex" but enclosed in double $$, i.e., as display maths.

"markdown" The labels are encoded as character strings using markdown syntax, with some embedded HTML.

"marquee" The labels are encoded as character strings using markdown syntax, with 'marquee' supported spans.

"text" The labels are plain ASCII character strings.

"numeric" No labels are generated. This value is accepted by the statistics, but not by the label formatting functions.

NULL The value used depends on the argument passed to geom.

If geom = "latex" (package 'xdvir') the output type used is "latex.eqn". If geom = "richtext" (package 'ggtext') or geom = "textbox" (package 'ggtext') the output type used is "markdown". If geom = "marquee" (package 'marquee') the output type used is "marquee". For all other values of geom the default is "expression". Invalid values as argument trigger an error.

Position of labels

When data are grouped by mapping a factor to an aesthetic, e.g., `colour`, `shape` and/or `linetype` the model is fitted separately to each group, and for each group a whole set of labels is generated. If the argument passed to `label.y` is a vector of length 1, this value determines the position of the equation and/or other labels for the first group, and the positions of the labels for the remaining groups are generated by adding `vspace` based on the group number. If the argument passed to `label.y` is a vector of length > 1, it is used unchanged, possibly extended by recycling, ignoring `vstep`.

If the labels are rotated by 90 degrees then the automatic stepping is best based on `hstep` with `vstep = 0`. Similarly as described above, if `label.x` is a vector of length > 1, it is used unchanged, possibly extended by recycling, ignoring `hstep`.

When using facets and with a grouping that does not repeat in each panel, the automatic positioning in most cases will not be the desired one. Manual positioning using a vector of length > 1 for `label.x` and/or `label.y` is the currently available workaround.

Which variables are available for mapping?

Computed variables and their names can vary depending on the `method` used to fit a model or the `output.type` in use. They can also depend for a given `method` on other arguments passed when fitting a model or extracting estimates and other computed values. In many cases, when values are not available, the variables are filled with `NA` values.

In the statistics returning formatted strings for use as annotations, a message is issued by default in interactive R sessions, listing the short names for available formatted labels as recognized by functions `use_label()` and `f_use_label()`, except when `output.type = "numeric"` is passed, in which case the names of all variables accessible by `after_stat()` within a call to `aes()` are listed. This default (`"nicknames"`) can be changed by setting R option `"ggpmisc.stat.vars.message"` to one of `"names"`, `"colnames"` or `"none"`.

In the statistics that plot a prediction or more generally mainly return numeric variables, a message is issued by default in interactive R sessions, listing the names of all variables accessible by `after_stat()` within a call to `aes()` with at least some non-missing values. This default (`"colnames"`) can be changed by setting R option `"ggpmisc.stat.vars.message"` to `"none"`.

Aesthetics

`stat_distrmix_eq()` understands the following aesthetics. Required aesthetics are displayed in bold and defaults are displayed for optional aesthetics:

- `x` ***or*** `y`
- `group` → `after_stat(component)`
- `hjust` → `"inward"`
- `label` → `after_stat(eq.label)`
- `npcx` → `after_stat(npcx)`
- `npcy` → `after_stat(npcy)`
- `vjust` → `"inward"`

`stat_distrmix_line()` understands the following aesthetics. Required aesthetics are displayed in bold and defaults are displayed for optional aesthetics:

- x ***or*** y
- group → after_stat(component)
- weight → NULL

Learn more about setting these aesthetics in vignette("ggplot2-specs").

See Also

For the underlying computations see for mixes of two or more Normal distributions normalmixEM() and for a single Normal distribution fitdistr().

statistics from 'ggpmisc' for model fit annotations: stat_poly_eq(), stat_quant_eq(), stat_ma_eq() and stat_distrmix_eq(), and for model fit predictions: stat_poly_line(), stat_quant_line(), stat_quant_band(), stat_ma_line() and stat_distrmix_line().

Examples

```
ggplot(faithful, aes(x = waiting)) +
  stat_distrmix_line() +
  stat_distrmix_eq()

ggplot(faithful, aes(x = waiting)) +
  stat_distrmix_area() +
  stat_distrmix_eq()

ggplot(faithful, aes(x = waiting)) +
  stat_distrmix_area(aes(fill = after_stat(quant.splits != 2)),
                     quantiles = c(0.025, 0.975),
                     show.legend = FALSE)

ggplot(faithful, aes(x = waiting)) +
  stat_distrmix_line(components = "sum") +
  stat_distrmix_eq(use_label("eq", "n", "method"))

ggplot(faithful, aes(x = waiting)) +
  stat_distrmix_line(components = "sum") +
  stat_distrmix_eq(geom = "label_npc")

ggplot(faithful, aes(x = waiting)) +
  stat_distrmix_line(components = "sum") +
  stat_distrmix_eq(geom = "text", label.x = "center", label.y = "bottom")

ggplot(faithful, aes(x = waiting)) +
  stat_distrmix_line(components = "sum") +
  stat_distrmix_eq(geom = "text", hjust = "inward")

ggplot(faithful, aes(x = waiting)) +
  stat_distrmix_line(components = "members") +
  stat_distrmix_eq(components = "members")

ggplot(faithful, aes(x = waiting)) +
```

```
  stat_distrmix_line(components = "members") +
  stat_distrmix_eq(components = "members", se = TRUE)

ggplot(faithful, aes(y = waiting)) +
  stat_distrmix_line(components = "sum") +
  stat_distrmix_eq(label.x = "right")

ggplot(faithful, aes(x = waiting)) +
  geom_histogram(aes(y = after_stat(density)), bins = 20) +
  stat_distrmix_line(aes(colour = after_stat(component),
                         fill = after_stat(component)),
                     geom = "area", linewidth = 1, alpha = 0.25) +
  stat_distrmix_eq(aes(colour = after_stat(component)))

ggplot(faithful, aes(x = waiting)) +
  stat_distrmix_line(aes(colour = after_stat(component),
                         fill = after_stat(component)),
                     geom = "area", linewidth = 1, alpha = 0.25,
                     components = "members") +
 stat_distrmix_eq(aes(colour = after_stat(component)),
                     components = "members")

ggplot(faithful, aes(x = waiting)) +
  stat_distrmix_line(geom = "area", linewidth = 1, alpha = 0.25,
                     colour = "black", outline.type = "upper",
                     components = "sum", se = FALSE) +
 stat_distrmix_eq(components = "sum")

# special case of no mixture
ggplot(subset(faithful, waiting > 66), aes(x = waiting)) +
  stat_distrmix_line(k = 1) +
  stat_distrmix_eq(k = 1)

ggplot(subset(faithful, waiting > 66), aes(x = waiting)) +
  stat_distrmix_line(k = 1) +
  stat_distrmix_eq(k = 1, se = TRUE)
```

`stat_fit_augment` *Augment data with fitted values and statistics*

Description

`stat_fit_augment()` fits a model and returns a "tidy" version of the model's data with prediction added, using `augmnent()` methods from packages 'broom', 'broom.mixed', or other sources. The prediction can be added to the plot as a line. The role played by 'augnebt()' is similar to that played by R's `fitted()`, `residuals()`, and in some cases also `weights()`, and `predict()`.

Usage

```
stat_fit_augment(
  mapping = NULL,
  data = NULL,
  geom = "smooth",
  position = "identity",
  ...,
  orientation = NA,
  method = "lm",
  method.args = list(formula = y ~ x),
  n.min = 2L,
  fit.seed = NA,
  augment.args = list(),
  level = 0.95,
  y.out = ".fitted",
  na.rm = FALSE,
  show.legend = FALSE,
  inherit.aes = TRUE
)
```

Arguments

mapping	The aesthetic mapping, usually constructed with aes(). Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset, only needed if you want to override the plot defaults.
geom	The geometric object to use display the data
position	The position adjustment to use for overlapping points on this layer.
...	other arguments passed on to layer. This can include aesthetics whose values you want to set, not map. See layer for more details.
orientation	character Either "x" or "y" controlling the default for formula. The letter indicates the aesthetic considered the explanatory variable in the model fit.
method	function or character If character, "lm", "rlm", "lmrob", "lts", "gls", "ma", "sma", "segreg", "rq" or the name of a model fit function are accepted, possibly followed by the fit function's method argument separated by a colon (e.g. "rlm:M"). If a function is different to lm(), rlm(), ltsReg(), gls(), ma, sma, it must have formal parameters named formula, data, and weights. See Details.
method.args, augment.args	list of arguments to pass to method and to to broom::augment.
n.min	integer Minimum number of distinct values in the explanatory variable (on the rhs of formula) for fitting to the attempted.
fit.seed	RNG seed argument passed to set.seed(). Defaults to NA, indicating that set.seed() should not be called.
level	Level of confidence interval to use (0.95 by default).
y.out	character (or numeric) index to column to return as y.

na.rm	a logical indicating whether NA values should be stripped before the computation proceeds.
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.
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.

Details

stat_fit_augment() together with stat_fit_glance() and stat_fit_tidy(), based on package 'broom' can be used with a broad range of model fitting functions as supported at any given time by 'broom'. In contrast to stat_poly_eq() which can generate text or expression labels automatically, for these functions the mapping of aesthetic label needs to be explicitly supplied in the call, and labels built on the fly.

Although arguments passed to parameter augment.args will be passed to augment() whether they are silently ignored or obeyed depends on each specialization of augment(). In particular, passing 'newdata' to obtain a prediction can in some cases result in other statisitcs being wrongly computed! Please, do carefully read the documentation for the version of augment() corresponding to the method used to fit the model. Be aware that se_fit = FALSE is the default in these methods even when supported.

This statistics is meant to be used in situations when all of stat_poly_line(), stat_quant_line(), stat_ma_line(), stat_quant_band(), stat_fit_residuals() and stat_fit_fitted() are unsuitable.

Warning! Not all augment() method specializations are defined in package 'broom'. augment() specializations for mixed models fits of classes "lme", "nlme", "lme4" and many others are defined in package 'broom.mixed'.

Value

A data.frame is returned, with a variable number of rows. The output of augment() is returned as is, except for y which is set based on y.out returned by augment() and y.observed which preserves the y returned by the generics::augment methods. This renaming is needed so that the geom works as expected.

To explore the values returned by this statistic, which vary depending on the model fitting function and model formula we suggest the use of geom_debug. An example is shown below.

Handling of grouping

stat_fit_augment() applies the function given by method separately to each group of observations; in 'ggplot2' factors mapped to aesthetics generate a separate group for each level. Because of this, stat_fit_augment() is not useful for annotating plots with results from t.test() or ANOVA or ANCOVA (e.g., when a factor is mapped to the *x* or *y* aesthetics). In such cases use instead stat_fit_tb() which applies the model fitting per panel.

Model formula and model fitting

A ggplot statistic receives as `data` a data frame that is not the one passed as argument by the user, but instead a data frame with the variables mapped to aesthetics. In `stat_poly_eq()` the compute function is applied by group, each call "seeing" the subset of `data` for an individual group. As supported models are for regression lines, variables mapped to `x` and `y` should both be continuous, i.e., numeric or date time and model formulas defined using `x` and `y` as variable names.

The interpretation of the argument passed to `formula` is enhanced compared to `stat_smooth()`. Formulas with `x` as explanatory variable work as in `stat_smooth()` but formulas with `y` as explanatory variable are also accepted. `orientation` is set automatically based on which explanatory variable appears in the formula. Spline-based smoothers are only partially supported.

Model fit methods supported

Several model fit functions are supported explicitly (see tables), and some of their differences smoothed out. Compatibility is checked late, based on the class of the returned fitted model object. This makes it possible to use wrapper functions that do model selection or other adjustments to the fit procedure on a per panel or per group basis. Moreover, if the value returned as model fit object is `NULL` or `NA`, plotting is skipped on a per group within panel basis.

In the case of fitted model objects of classes not explicitly supported, an attempt is made to find the usual accessors and/or fitted object members, and if found, either complete or partial support is frequently achieved. In this case a message is issued encouraging users to check the validity of the values extracted as the structure of fitted model objects belonging to different classes and the values returned by their accessors can vary, potentially resulting in decoding errors leading to the return of wrong values for estimates.

The argument to parameter `method` can be either the name of a function object, possibly using double colon notation in case its package is not attached, or a character string matching the function name for functions in the search path. This approach makes it possible to support model fit functions that are not dependencies of 'ggpmisc'. Either by attaching the package where the function is defined and passing it by name or as string, or using double colon notation when passing the name of the function.

User-defined functions can be passed as argument to parameter `method` as long as they have parameters `formula`, `data` `subset` and possibly `weights`. Additional arguments can be passed to any method as a named list through parameter `method.args`. As in `stat_smooth()` prior `weights` are passed to the model fit functions' `weights` (plural!) parameter by mapping a numeric variable to plot aesthetic `weight` (singular!).

Tables 1 lists natively supported model fit functions, with the caveat that only some 'broom' methods' specializations have been actually tested with statistics from 'ggpmisc'. In addition, the statistics based on 'broom' methods require the user to tailor their behaviour by passing additional arguments in the call and occasionally some detective work to find out the names of variables in the returned data frame as these names are set by methods from 'broom'.

Table 1. Model fit methods supported by the different statistics available in package 'ggpmisc'. Column *f* indicates whether computations are done by group (G) or by plot panel (P).

Statistic	*f*	Supported model fit methods
`stat_poly_line()`	G	"lm", "rlm", "lts", "sma", "ma", "gls", "nls", "onls", *others with methods* `predict()` or `fitt`
`stat_poly_eq()`	G	"lm", "rlm", "lts", "sma", "ma", "gls", "nls", "onls", *others with needed accesors*

stat_quant_line()	G	"rq", "rqss"
stat_quant_band()	G	"rq", "rqss"
stat_quant_eq()	G	"rq", "rqss"
stat_ma_line()	G	"SMA", "MA", "RMA", "OLS"
stat_ma_eq()	G	"SMA", "MA", "RMA", "OLS"
stat_fit_residuals()	G	"lm", "rlm", "lts", "sma", "ma", "gls", "rq", "rqss", "nls", "onls", *others with method* residu
stat_fit_fitted()	G	"lm", "rlm", "lts", "gls", "rq", "rqss", "nls", "onls", *others with method* fitted()
stat_fit_deviations()	G	"lm", "rlm", "lts", "gls", "rq", "rqss", "nls", "onls", *others with methods* fitted() and weigh
stat_fit_augment()	G	*any with 'broom' method* augment()
stat_fit_glance()	G	*any with 'broom' method* glance()
stat_fit_tidy()	G	*any with 'broom' method* tidy()
stat_fit_tb()	P	*any with 'broom' method* tidy()

The single colon notation is based on parsing the name and is available when passing the name of the fit method as a character string. In a string such as "head:tail" the "head" gives the name of the model fit function and the "tail" gives the argument to pass it's `method` parameter. This is only a convenience, as `method.args` can be also used. In some methods, i.e., splines, the default `formula = y ~ x` needs to be overridden by the user.

Table 2 lists the correspondence of pre-defined *method names* to model fit method functions. As mentioned above, these are only a subset of the model fit methods that are expected to work. When using these names there is no need for users to attach additional packages but the packages must be available (installed).

Table 2. Available predefined method names, the model fit functions they call, the packages where the functions reside, the class of the returned fitted model object and the arguments that can be passed to their `method` parameter using single colon notation.

Predefined method names	Model fit methods	R package	Object class
"lm", "lm:qr"	lm()	'stats'	"lm"
"rlm", "rlm:M", "rlm:MM"	rlm()	'MASS'	"rlm" ("lm")
"lts", "ltsReg"	ltsReg()	'robustbase'	"lts"
"ma", "sma", "sma:SMA", "sma:MA", "sma:OLS"	sma()	'smatr'	"ma" or "sma"
"gls", "gls:REML", "gls:ML"	gls()	'nlme'	"gls"
"rq", "rq:sfn", "rq:sfnc", "rq:lasso"	rq()	'quantreg'	"rq"
"rqss", "rqss:sfn", "rqss:sfnc", "rqss:lasso"	rqss()	'quantreg'	"rqss"
"SMA", "MA", "RMA", "OLS"	lmodel2()	'lmodel2'	("list")

Which variables are available for mapping?

Computed variables and their names vary following those used in the specializations of 'broom' methods. A message is issued by default in interactive R sessions, listing the names of all variables accessible by `after_stat()` within a call to `aes()` with at least some non-missing values. This default (`"colnames"`) can be changed by setting R option `"ggpmisc.stat.vars.message"` to `"none"`.

Aesthetics

`stat_fit_augment()` understands the following aesthetics. Required aesthetics are displayed in bold and defaults are displayed for optional aesthetics:

- x
- y
- group → inferred
- ymax → `after_stat(y + .se.fit * t.value)`
- ymin → `after_stat(y - .se.fit * t.value)`

Learn more about setting these aesthetics in `vignette("ggplot2-specs")`.

See Also

Package broom for details on how the tidying of the result of model fits is done.

Other 'broom'-based statistics for model fits: `stat_fit_glance()`, `stat_fit_tb()`, `stat_fit_tidy()`

Examples

```
# Package 'broom' needs to be installed to run these examples.
# We check availability before running them to avoid errors.

broom.installed <- requireNamespace("broom", quietly = TRUE)
gginnards.installed <- requireNamespace("gginnards", quietly = TRUE)

if (broom.installed) {
  library(broom)
}

# Regression by panel example
if (broom.installed)
  ggplot(mtcars, aes(x = disp, y = mpg)) +
    geom_point(aes(colour = factor(cyl))) +
    stat_fit_augment(method = "lm",
                     method.args = list(formula = y ~ x))

# Flipped regression by panel example
if (broom.installed)
  ggplot(mtcars, aes(x = disp, y = mpg)) +
    geom_point(aes(colour = factor(cyl))) +
    stat_fit_augment(method = "lm",
                     orientation = "y",
                     method.args = list(formula = y ~ x))

if (broom.installed)
  ggplot(mtcars, aes(x = disp, y = mpg)) +
    geom_point(aes(colour = factor(cyl))) +
    stat_fit_augment(method = "lm",
                     augment.args = list(se_fit = TRUE),
```

```
                    method.args = list(formula = y ~ x + I(x^2)))

# Residuals from regression by panel example
if (broom.installed)
  ggplot(mtcars, aes(x = disp, y = mpg)) +
    geom_hline(yintercept = 0, linetype = "dotted") +
    stat_fit_augment(geom = "point",
                    method = "lm",
                    method.args = list(formula = y ~ x),
                    y.out = ".resid")

# Regression by group example
if (broom.installed)
  ggplot(mtcars, aes(x = disp, y = mpg, colour = factor(cyl))) +
    geom_point() +
    stat_fit_augment(method = "lm",
                    augment.args = list(se_fit = TRUE),
                    method.args = list(formula = y ~ x))

# Residuals from regression by group example
if (broom.installed)
  ggplot(mtcars, aes(x = disp, y = mpg, colour = factor(cyl))) +
    geom_hline(yintercept = 0, linetype = "dotted") +
    stat_fit_augment(geom = "point",
                    method.args = list(formula = y ~ x),
                    y.out = ".resid")

# Weighted regression example
if (broom.installed)
  ggplot(mtcars, aes(x = disp, y = mpg, weight = cyl)) +
    geom_point(aes(colour = factor(cyl))) +
    stat_fit_augment(method = "lm",
                    method.args = list(formula = y ~ x,
                                       weights = quote(weight)))

# Residuals from weighted regression example
if (broom.installed)
  ggplot(mtcars, aes(x = disp, y = mpg, weight = cyl)) +
    geom_hline(yintercept = 0, linetype = "dotted") +
    stat_fit_augment(geom = "point",
                    method.args = list(formula = y ~ x,
                                       weights = quote(weight)),
                    y.out = ".resid")

# Inspecting the returned data using geom_debug_group()
  if (gginnards.installed) {
    library(gginnards)
}

# Regression by panel, inspecting data
if (broom.installed & gginnards.installed) {
    ggplot(mtcars, aes(x = disp, y = mpg)) +
      geom_point(aes(colour = factor(cyl))) +
```

```
      stat_fit_augment(method = "lm",
                       method.args = list(formula = y ~ x),
                       geom = "debug_group",
                       dbgfun.data = colnames)
}
```

`stat_fit_deviations` *Residuals and fitted values from model fit*

Description

Statistic `stat_fit_residuals()` fits a model and plots residuals vs. `x`. Statistic `stat_fit_deviations()` fits a model and and highlighting residuals as segments in a *y* vs. *x* plot. Statistic `stat_fit_fitted()` plots the fitted values vs. *x*.

Usage

```
stat_fit_deviations(
  mapping = NULL,
  data = NULL,
  geom = "segment",
  position = "identity",
  ...,
  orientation = NA,
  method = "lm",
  method.args = list(),
  n.min = 2L,
  formula = NULL,
  fit.seed = NA,
  na.rm = FALSE,
  show.legend = TRUE,
  inherit.aes = TRUE
)

stat_fit_fitted(
  mapping = NULL,
  data = NULL,
  geom = "point",
  position = "identity",
  orientation = NA,
  ...,
  method = "lm",
  method.args = list(),
  n.min = 2L,
  formula = NULL,
  fit.seed = NA,
```

```
  na.rm = FALSE,
  show.legend = FALSE,
  inherit.aes = TRUE
)

stat_fit_residuals(
  mapping = NULL,
  data = NULL,
  geom = "point",
  position = "identity",
  ...,
  orientation = NA,
  method = "lm",
  method.args = list(),
  n.min = 2L,
  formula = NULL,
  fit.seed = NA,
  resid.type = NULL,
  weighted = FALSE,
  na.rm = FALSE,
  show.legend = TRUE,
  inherit.aes = TRUE
)
```

Arguments

mapping	The aesthetic mapping, usually constructed with aes(). Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset, only needed if you want to override the plot defaults.
geom	The geometric object to use display the data
position	The position adjustment to use for overlapping points on this layer.
...	other arguments passed on to layer. This can include aesthetics whose values you want to set, not map. See layer for more details.
orientation	character Either "x" or "y" controlling the default for formula. The letter indicates the aesthetic considered the explanatory variable in the model fit.
method	function or character If character, "lm", "rlm", "lmrob", "lts", "gls", "ma", "sma", "segreg", "rq" or the name of a model fit function are accepted, possibly followed by the fit function's method argument separated by a colon (e.g. "rlm:M"). If a function is different to lm(), rlm(), ltsReg(), gls(), ma, sma, it must have formal parameters named formula, data, and weights. See Details.
method.args	named list with additional arguments. Not data or weights which are always passed through aesthetic mappings.
n.min	integer Minimum number of distinct values in the explanatory variable (on the rhs of formula) for fitting to the attempted.
formula	a formula object. Using aesthetic names x and y instead of original variable names.

fit.seed	RNG seed argument passed to set.seed(). Defaults to NA, indicating that set.seed() should not be called.
na.rm	a logical indicating whether NA values should be stripped before the computation proceeds.
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.
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.
resid.type	character passed to residuals() as argument for type (defaults to "working" except if weighted = TRUE when it is forced to "deviance").
weighted	logical If true weighted residuals will be returned.

Details

stat_fit_deviations() can be used to highlight residuals as segments in a plot of a fitted model prediction. This statistic returns the original x and y values and the fitted y or x values depending on the *orientation*, together with prior and posterior weights.

stat_fit_fitted() can be used to highlight as points the fitted values. This statistic returns the original x or y values and the fitted y or x values depending on the *orientation*.

stat_fit_residuals() plots residuals as points. It applies to the fitted model object methods residuals() or weighted.residuals() depending on the argument passed to parameter weighted. This statistic returns the original x and y values and residuals depending on the *orientation*, together with prior and posterior weights.

Value

The returned value is always a data frame with the same number of rows as the argument passed to data, except for the case failure of the model fitting, in which case a data frame with no rows is returned. The columns returned vary between the three statistics, and for each statistic depending on the orientation..

To explore the values returned by statistics we suggest the use of geom_debug_group().

Prior and posterior weights

Two types of weights are possible: prior ones supplied in the call, and posterior weights (called "robustness weights" in robust regression methods) implicitly or explicitly used by fit methods to address heterogeneity of error variance, including the presence of outlier observations . Not all the supported methods accepts prior weights and gls() returns posterior weights that are not in 0..1 like in the case of most other fits. When not accessible weights are set to 1 when known to be equal to 1, which is the most frequent case, or to NA otherwise.

How weights are applied to residuals depends on the method used to fit the model. For ordinary least squares (OLS), weights are applied to the squares of the residuals, so the weighted residuals are obtained by multiplying the "deviance" residuals by the square root of the weights. When residuals are penalized differently to fit a model, the weighted residuals need to be computed accordingly.

Variables returned by `stat_fit_residuals()`

x x coordinates of observations

y y coordinates of observations

x.resid x residuals from fitted values

y.resid y residuals from fitted values

weights the weights passed as input to `lm()`, `rlm()`, `lmrob()`, or to other model fit functions using aesthetic weight. More generally the value returned by method `weights()` applied to the model fit object

posterior.weights the "weights" of the applied minimization criterion relative to those of OLS in `rlm()` or `lmrob()` or the divisor weights from `gls()`, `lme()` or `nlme()`

Variables returned by `stat_fit_deviations()`

x x coordinates of observations

y y coordinates of observations

x.fitted x coordinates of fitted values

y.fitted y coordinates of fitted values

weights the weights passed as input to `lm()`, `rlm()`, or `lmrob()`, using aesthetic weight. More generally the value returned by `weights()`

posterior.weights the "weights" of the applied minimization criterion relative to those of OLS in `rlm()`, or `lmrob()`

Variables returned by `stat_fit_fitted()`

x x coordinates of observations or fitted

y y coordinates of observations or fitted

Model formula and model fitting

A ggplot statistic receives as `data` a data frame that is not the one passed as argument by the user, but instead a data frame with the variables mapped to aesthetics. In `stat_poly_eq()` the compute function is applied by group, each call "seeing" the subset of `data` for an individual group. As supported models are for regression lines, variables mapped to `x` and `y` should both be continuous, i.e., numeric or date time and model formulas defined using `x` and `y` as variable names.

The interpretation of the argument passed to `formula` is enhanced compared to `stat_smooth()`. Formulas with `x` as explanatory variable work as in `stat_smooth()` but formulas with `y` as explanatory variable are also accepted. `orientation` is set automatically based on which explanatory variable appears in the formula. Spline-based smoothers are only partially supported.

Model fit methods supported

Several model fit functions are supported explicitly (see tables), and some of their differences smoothed out. Compatibility is checked late, based on the class of the returned fitted model object. This makes it possible to use wrapper functions that do model selection or other adjustments

to the fit procedure on a per panel or per group basis. Moreover, if the value returned as model fit object is NULL or NA, plotting is skipped on a per group within panel basis.

In the case of fitted model objects of classes not explicitly supported, an attempt is made to find the usual accessors and/or fitted object members, and if found, either complete or partial support is frequently achieved. In this case a message is issued encouraging users to check the validity of the values extracted as the structure of fitted model objects belonging to different classes and the values returned by their accessors can vary, potentially resulting in decoding errors leading to the return of wrong values for estimates.

The argument to parameter method can be either the name of a function object, possibly using double colon notation in case its package is not attached, or a character string matching the function name for functions in the search path. This approach makes it possible to support model fit functions that are not dependencies of 'ggpmisc'. Either by attaching the package where the function is defined and passing it by name or as string, or using double colon notation when passing the name of the function.

User-defined functions can be passed as argument to parameter method as long as they have parameters formula, data subset and possibly weights. Additional arguments can be passed to any method as a named list through parameter method.args. As in stat_smooth() prior weights are passed to the model fit functions' weights (plural!) parameter by mapping a numeric variable to plot aesthetic weight (singular!).

Tables 1 lists natively supported model fit functions, with the caveat that only some 'broom' methods' specializations have been actually tested with statistics from 'ggpmisc'. In addition, the statistics based on 'broom' methods require the user to tailor their behaviour by passing additional arguments in the call and occasionally some detective work to find out the names of variables in the returned data frame as these names are set by methods from 'broom'.

Table 1. Model fit methods supported by the different statistics available in package 'ggpmisc'. Column *f* indicates whether computations are done by group (G) or by plot panel (P).

Statistic	*f*	Supported model fit methods
stat_poly_line()	G	"lm", "rlm", "lts", "sma", "ma", "gls", "nls", "onls", *others with methods* predict() or fitt
stat_poly_eq()	G	"lm", "rlm", "lts", "sma", "ma", "gls", "nls", "onls", *others with needed accesors*
stat_quant_line()	G	"rq", "rqss"
stat_quant_band()	G	"rq", "rqss"
stat_quant_eq()	G	"rq", "rqss"
stat_ma_line()	G	"SMA", "MA", "RMA", "OLS"
stat_ma_eq()	G	"SMA", "MA", "RMA", "OLS"
stat_fit_residuals()	G	"lm", "rlm", "lts", "sma", "ma", "gls", "rq", "rqss", "nls", "onls", *others with method* residu
stat_fit_fitted()	G	"lm", "rlm", "lts", "gls", "rq", "rqss", "nls", "onls", *others with method* fitted()
stat_fit_deviations()	G	"lm", "rlm", "lts", "gls", "rq", "rqss", "nls", "onls", *others with methods* fitted() and weigh
stat_fit_augment()	G	*any with 'broom' method* augment()
stat_fit_glance()	G	*any with 'broom' method* glance()
stat_fit_tidy()	G	*any with 'broom' method* tidy()
stat_fit_tb()	P	*any with 'broom' method* tidy()

The single colon notation is based on parsing the name and is available when passing the name of the fit method as a character string. In a string such as "head:tail" the "head" gives the name of the model fit function and the "tail" gives the argument to pass it's method parameter. This is only a

convenience, as `method.args` can be also used. In some methods, i.e., splines, the default `formula = y ~ x` needs to be overridden by the user.

Table 2 lists the correspondence of pre-defined *method names* to model fit method functions. As mentioned above, these are only a subset of the model fit methods that are expected to work. When using these names there is no need for users to attach additional packages but the packages must be available (installed).

Table 2. Available predefined method names, the model fit functions they call, the packages where the functions reside, the class of the returned fitted model object and the arguments that can be passed to their `method` parameter using single colon notation.

Predefined method names	Model fit methods	R package	Object class
"lm", "lm:qr"	`lm()`	'stats'	"lm"
"rlm", "rlm:M", "rlm:MM"	`rlm()`	'MASS'	"rlm" ("lm")
"lts", "ltsReg"	`ltsReg()`	'robustbase'	"lts"
"ma", "sma", "sma:SMA", "sma:MA", "sma:OLS"	`sma()`	'smatr'	"ma" or "sma"
"gls", "gls:REML", "gls:ML"	`gls()`	'nlme'	"gls"
"rq", "rq:sfn", "rq:sfnc", "rq:lasso"	`rq()`	'quantreg'	"rq"
"rqss", "rqss:sfn", "rqss:sfnc", "rqss:lasso"	`rqss()`	'quantreg'	"rqss"
"SMA", "MA", "RMA", "OLS"	`lmodel2()`	'lmodel2'	("list")

Which variables are available for mapping?

Computed variables and their names vary following those used in the specializations of 'broom' methods. A message is issued by default in interactive R sessions, listing the names of all variables accessible by `after_stat()` within a call to `aes()` with at least some non-missing values. This default (`"colnames"`) can be changed by setting R option `"ggpmisc.stat.vars.message"` to `"none"`.

Model equation label

By default the equation label uses as symbols the names of the aesthetics, `x` and `y`. However, `"x"` and `"y"` can be substituted by providing a replacement character string for the right-hand-side and left-hand-side through `eq.x.rhs` and `eq.with.lhs`, respectively. For backward compatibility a logical is also accepted as argument for `eq.with.lhs`, with `FALSE` suppressing the left-hand-side.

If the model `formula` includes a transformation of the explanatory variable in its right-hand-side (rhs), a matching argument should be passed to parameter `eq.x.rhs` as its default value would result in an equation label that does not reflect the applied transformation. In most cases, a transformation should not be applied within the left hand side (lhs) of the model formula, but instead in the mapping of the response variable within `aes`. In this case it may be necessary to also pass a matching argument to parameter `eq.with.lhs`.

Parameter `orientation` is redundant as the orientation can be set by the `formula` but is included for consistency with `ggplot2::stat_smooth()`.

Position of labels

When data are grouped by mapping a factor to an aesthetic, e.g., `colour`, `shape` and/or `linetype` the model is fitted separately to each group, and for each group a whole set of labels is generated.

If the argument passed to `label.y` is a vector of length 1, this value determines the position of the equation and/or other labels for the first group, and the positions of the labels for the remaining groups are generated by adding `vspace` based on the group number. If the argument passed to `label.y` is a vector of length > 1, it is used unchanged, possibly extended by recycling, ignoring `vstep`.

If the labels are rotated by 90 degrees then the automatic stepping is best based on `hstep` with `vstep = 0`. Similarly as described above, if `label.x` is a vector of length > 1, it is used unchanged, possibly extended by recycling, ignoring `hstep`.

When using facets and with a grouping that does not repeat in each panel, the automatic positioning in most cases will not be the desired one. Manual positioning using a vector of length > 1 for `label.x` and/or `label.y` is the currently available workaround.

Range of the prediction line

The range of the prediction line is controlled by parameters `fullrange` and `limit.to`. `fullrange` is backwards compatible both with earlier versions of 'ggpmisc' and with `stat_smooth()` from 'ggplot2'; an argument passed to `limit.to` overrides `fullrange` making it possible to constrain the range to that of `x`, `y`, or both simultaneously, with `"x"`, `"y"`, or `"xy"`, respectively, as argument. `limit.to` also accepts a numeric vector of values to be used as `newdata` when computing the prediction. Limiting the range based on both aesthetics is the best approach for major axis regression (MA, SMA, RMA) but can occasionally be useful also with some other methods when slopes are very steep and error variance in the explanatory variable is large. A numeric vector can be used to predict the response at specific values of the explanatory variable. If a single or very few values are predicted, it can be necessary to override the default `geom = "smooth"` with `geom = "pointrange"`.

Variables returned by `stat_poly_line()`

y or x predicted value

ymin or xmin lower confidence limit around the fitted line

ymax or xmax upper confidence limit around the fitted line

se standard error

If `fm.values = TRUE` is passed then columns based on the summary of the model fit are added, with the same value in each row within a group. This is wasteful and disabled by default, but provides a simple and robust approach to achieve effects like colouring or hiding of the model fit line based on P, R^2, R^2_{adj} or the number of observations in a fit.

Variables returned by `stat_poly_eq()`

For all `output.type` arguments the following values are returned.

x,npcx x position

y,npcy y position

coefs fitted coefficients, named numeric vector as a list member

r.squared, rr.confint.level, rr.confint.low, rr.confint.high, adj.r.squared, f.value, f.df1, f.df2, p.value, AIC, BIC, n, knot numeric values, from the model fit object

grp.label Set according to mapping in `aes`.

knots list containing a numeric vector of knot or "psi" x-value for linear splines

fm.method name of method used, character

fm.class most derived class or the fitted model object, character

fm.formula.chr formatted model formula, character

If `output.type` is not `"numeric"` the returned tibble contains in addition to those above the columns listed below, each containing a single character string. The markup used depends on the value of `output.type`.

eq.label equation for the fitted polynomial as a character string to be parsed or `NA`

rr.label R^2 of the fitted model as a character string to be parsed

adj.rr.label Adjusted R^2 of the fitted model as a character string to be parsed

rr.confint.label Confidence interval for R^2 of the fitted model as a character string to be parsed

f.value.label F value and degrees of freedom for the fitted model as a whole.

p.value.label P-value for the F-value above.

AIC.label AIC for the fitted model.

BIC.label BIC for the fitted model.

n.label Number of observations used in the fit.

knots.label The knots or change points in segmented regression.

grp.label Set according to mapping in `aes`.

method.label Set according `method` used.

If output.type is `"numeric"` the returned tibble contains columns listed below in addition to the base ones. If the model fit function used does not return a value, the variable is set to `NA_real_`.

coef.ls list containing the "coefficients" matrix from the summary of the fit object

b_0.constant TRUE is polynomial is forced through the origin

b_i One or more columns with the coefficient estimates

To explore the computed values returned for a given input we suggest the use of `geom_debug()` as shown in the last examples below.

Output types

The formatting of character strings to be displayed in plots are marked as mathematical equations. Depending on the geom used, the mark-up needs to be encoded differently, or in some cases mark-up not applied.

`"expression"` The labels are encoded as character strings to be parsed into R's plotmath expressions.

`"LaTeX"`, `"TeX"`, `"tikz"`, `"latex"` The labels are encoded as 'LaTeX' maths equations, without the "fences" for switching in math mode.

`"latex.eqn"` Same as `"latex"` but enclosed in single `$`, i.e., as in-line maths.

`"latex.deqn"` Same as `"latex"` but enclosed in double `$$`, i.e., as display maths.

"markdown" The labels are encoded as character strings using markdown syntax, with some embedded HTML.

"marquee" The labels are encoded as character strings using markdown syntax, with 'marquee' supported spans.

"text" The labels are plain ASCII character strings.

"numeric" No labels are generated. This value is accepted by the statistics, but not by the label formatting functions.

NULL The value used depends on the argument passed to geom.

If geom = "latex" (package 'xdvir') the output type used is "latex.eqn". If geom = "richtext" (package 'ggtext') or geom = "textbox" (package 'ggtext') the output type used is "markdown". If geom = "marquee" (package 'marquee') the output type used is "marquee". For all other values of geom the default is "expression". Invalid values as argument trigger an error.

Aesthetics

stat_fit_residuals() understands the following aesthetics. Required aesthetics are displayed in bold and defaults are displayed for optional aesthetics:

- x
- y
- group → inferred

stat_fit_deviations() understands the following aesthetics. Required aesthetics are displayed in bold and defaults are displayed for optional aesthetics:

- x
- y
- group → inferred
- xend → after_stat(x.fitted)
- yend → after_stat(y.fitted)

stat_fit_fitted() understands the following aesthetics. Required aesthetics are displayed in bold and defaults are displayed for optional aesthetics:

- x
- y
- group → inferred

Learn more about setting these aesthetics in vignette("ggplot2-specs").

Note

In the case of method = "rq" quantiles are fixed at tau = 0.5 unless method.args has length > 0. Parameter orientation is redundant as it only affects the default for formula but is included for consistency with ggplot2.

References

Originally written as an answer to question 7549694 at Stackoverflow but enhanced based on suggestions from several users and my own needs.

See Also

`residuals()` and `weights()` and their specializations for the `method` used.

Please, see the articles at online-only documentation for additional use examples and guidance.

Examples

```
# generate artificial data
set.seed(4321)
x <- 1:100
y <- (x + x^2 + x^3) + rnorm(length(x), mean = 0, sd = mean(x^3) / 4)
my.data <- data.frame(x, y)

# give a name to a formula
my.formula <- y ~ poly(x, 3, raw = TRUE)
my.y.formula <- x ~ poly(y, 3, raw = TRUE)

# plot residuals from linear model
ggplot(my.data, aes(x, y)) +
  stat_poly_line(method = "lm", formula = my.formula) +
  stat_fit_deviations(method = "lm", formula = my.formula, colour = "red") +
  geom_point()

# plot residuals from linear model with y as explanatory variable
ggplot(my.data, aes(x, y)) +
  stat_poly_line(method = "lm", formula = my.y.formula) +
  stat_fit_deviations(method = "lm", formula = my.y.formula, colour = "red") +
  geom_point()

# plot robust regression
ggplot(my.data, aes(x, y)) +
  stat_poly_line(formula = my.formula, method = "rlm") +
  stat_fit_deviations(formula = my.formula, method = "rlm", colour = "red") +
  geom_point()

# plot robust regression with weights indicated by colour
my.data.outlier <- my.data
my.data.outlier[6, "y"] <- my.data.outlier[6, "y"] * 5
ggplot(my.data.outlier, aes(x, y)) +
  stat_poly_line(method = MASS::rlm, formula = my.formula) +
  stat_fit_deviations(formula = my.formula, method = "rlm",
                      mapping = aes(colour = after_stat(posterior.weights)),
                      show.legend = TRUE) +
  scale_color_gradient(low = "red", high = "blue", limits = c(0, 1),
                       guide = "colourbar") +
  geom_point()
```

```
# plot quantile regression (= median regression)
ggplot(my.data, aes(x, y)) +
  stat_quantile(formula = my.formula, quantiles = 0.5) +
  stat_fit_deviations(formula = my.formula, method = "rq", colour = "red") +
  geom_point()

# plot quantile regression (= "quartile" regression)
ggplot(my.data, aes(x, y)) +
  stat_quantile(formula = my.formula, quantiles = 0.75) +
  stat_fit_deviations(formula = my.formula, colour = "red",
                      method = "rq", method.args = list(tau = 0.75)) +
  geom_point()

# plot residuals from linear model
ggplot(my.data, aes(x, y)) +
  geom_hline(yintercept = 0, linetype = "dashed") +
  stat_fit_residuals(formula = my.formula)

# plot residuals from linear model with y as explanatory variable
ggplot(my.data, aes(x, y)) +
  geom_vline(xintercept = 0, linetype = "dashed") +
  stat_fit_residuals(formula = my.y.formula) +
  coord_flip()

ggplot(my.data, aes(x, y)) +
  geom_hline(yintercept = 0, linetype = "dashed") +
  stat_fit_residuals(formula = my.formula, resid.type = "response")

# plot residuals with weights indicated by colour
my.data.outlier <- my.data
my.data.outlier[6, "y"] <- my.data.outlier[6, "y"] * 5
ggplot(my.data.outlier, aes(x, y)) +
  stat_fit_residuals(formula = my.formula, method = "rlm",
                     mapping = aes(colour = after_stat(posterior.weights)),
                     show.legend = TRUE) +
  scale_color_gradient(low = "red", high = "blue", limits = c(0, 1),
                       guide = "colourbar")

# plot weighted residuals with weights indicated by colour
ggplot(my.data.outlier) +
  stat_fit_residuals(formula = my.formula, method = "rlm",
                     mapping = aes(x = x,
                                   y = stage(start = y, after_stat = y * weights),
                                   colour = after_stat(posterior.weights)),
                     show.legend = TRUE) +
  scale_color_gradient(low = "red", high = "blue", limits = c(0, 1),
                       guide = "colourbar")

# inspecting the returned data
gginnards.installed <- requireNamespace("gginnards", quietly = TRUE)

if (gginnards.installed)
  library(gginnards)
```

```
# plot, using geom_debug_group() to explore the after_stat data
if (gginnards.installed)
  ggplot(my.data, aes(x, y)) +
    stat_fit_deviations(formula = my.formula,
                        geom = "debug_group")

if (gginnards.installed)
  ggplot(my.data.outlier, aes(x, y)) +
    stat_fit_deviations(formula = my.formula, method = "rlm",
                        geom = "debug_group")

if (gginnards.installed)
  ggplot(my.data, aes(x, y)) +
   stat_fit_residuals(formula = my.formula, resid.type = "working",
                      geom = "debug_group")

if (gginnards.installed)
  ggplot(my.data, aes(x, y)) +
    stat_fit_residuals(formula = my.formula, method = "rlm",
                       geom = "debug_group")

if (gginnards.installed)
  ggplot(my.data, aes(x, y)) +
   stat_fit_fitted(formula = my.formula,
                   geom = "debug_group")
```

`stat_fit_glance` *One row summary data frame for a fitted model*

Description

`stat_fit_glance()` fits a model and returns a "tidy" version of the model's fit, using 'glance() methods from packages 'broom', 'broom.mixed', or other sources. The role played by 'glance()' is similar to that played by R's `anova()`.

Usage

```
stat_fit_glance(
  mapping = NULL,
  data = NULL,
  geom = "text_npc",
  position = "identity",
  ...,
  orientation = NA,
  method = "lm",
  method.args = list(formula = y ~ x),
  n.min = 2L,
```

```
  fit.seed = NA,
  glance.args = list(),
  label.x = "left",
  label.y = "top",
  hstep = 0,
  vstep = 0.075,
  na.rm = FALSE,
  show.legend = FALSE,
  inherit.aes = TRUE
)
```

Arguments

mapping	The aesthetic mapping, usually constructed with aes(). Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset, only needed if you want to override the plot defaults.
geom	The geometric object to use display the data
position	The position adjustment to use for overlapping points on this layer.
...	other arguments passed on to layer. This can include aesthetics whose values you want to set, not map. See layer for more details.
orientation	character Either "x" or "y" controlling the default for formula. The letter indicates the aesthetic considered the explanatory variable in the model fit.
method	function or character If character, "lm", "rlm", "lmrob", "lts", "gls", "ma", "sma", "segreg", "rq" or the name of a model fit function are accepted, possibly followed by the fit function's method argument separated by a colon (e.g. "rlm:M"). If a function is different to lm(), rlm(), ltsReg(), gls(), ma, sma, it must have formal parameters named formula, data, and weights. See Details.
method.args, glance.args	list of named arguments to pass to method and to [generics::glance()], respectively.
n.min	integer Minimum number of distinct values in the explanatory variable (on the rhs of formula) for fitting to the attempted.
fit.seed	RNG seed argument passed to set.seed(). Defaults to NA, indicating that set.seed() should not be called.
label.x, label.y	numeric with range 0..1 "normalized parent coordinates" (npc units) or character if using geom_text_npc() or geom_label_npc(). If using geom_text() or geom_label() numeric in native data units. If too short they will be recycled.
hstep, vstep	numeric in npc units, the horizontal and vertical step used between labels for different groups.
na.rm	a logical indicating whether NA values should be stripped before the computation proceeds.
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.
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.

Details

stat_fit_glance() together with stat_fit_tidy() and stat_fit_augment(), based on package 'broom' can be used with a broad range of model fitting functions as supported at any given time by package 'broom'. In contrast to stat_poly_eq() which can generate text or expression labels automatically, for these functions the mapping of aesthetic label needs to be explicitly supplied in the call, and labels built on the fly in the mapping to geom aesthetics.

Although arguments passed to parameter glance.args are passed to glance() whether they are silently ignored or obeyed depends on each specialization of glance(), so do carefully read the documentation for the version of glance() corresponding to the method used to fit the model.

This statistics is meant to be used in situations when all of stat_poly_eq(), stat_quant_eq(), stat_ma_eq() and stat_correlation() are unsuitable.

Warning! Not all glance() methods are defined in package 'broom'. glance() specializations for mixed models fits of classes "lme", "nlme", "lme4" and many others are defined in package 'broom.mixed'.

Value

The output of the glance() methods is returned almost as is in the data object, as a data frame. The names of the columns in the returned data are consistent with those returned by method glance() from package 'broom', that will frequently differ from the name of values returned by the print methods corresponding to the fit or test function used. To explore the values returned by this statistic including the name of variables/columns, which vary depending on the model fitting function and model formula we suggest the use of geom_debug. An example is shown below.

Handling of grouping

stat_fit_glance applies the function given by method separately to each group of observations, and factors mapped to aesthetics, including x and y aesthetics, create a separate group for each factor level. Because of this, stat_fit_glance is not useful for annotating plots with results from t.test(), ANOVA or ANCOVA. In such cases use the stat_fit_tb() statistic which applies the model fitting per panel.

Which variables are available for mapping?

Computed variables and their names vary following those used in the specializations of 'broom' methods. A message is issued by default in interactive R sessions, listing the names of all variables accessible by after_stat() within a call to aes() with at least some non-missing values. This default ("colnames") can be changed by setting R option "ggpmisc.stat.vars.message" to "none".

Model formula and model fitting

A ggplot statistic receives as data a data frame that is not the one passed as argument by the user, but instead a data frame with the variables mapped to aesthetics. In stat_poly_eq() the compute function is applied by group, each call "seeing" the subset of data for an individual group. As supported models are for regression lines, variables mapped to x and y should both be continuous, i.e., numeric or date time and model formulas defined using x and y as variable names.

The interpretation of the argument passed to `formula` is enhanced compared to `stat_smooth()`. Formulas with `x` as explanatory variable work as in `stat_smooth()` but formulas with `y` as explanatory variable are also accepted. `orientation` is set automatically based on which explanatory variable appears in the formula. Spline-based smoothers are only partially supported.

Model fit methods supported

Several model fit functions are supported explicitly (see tables), and some of their differences smoothed out. Compatibility is checked late, based on the class of the returned fitted model object. This makes it possible to use wrapper functions that do model selection or other adjustments to the fit procedure on a per panel or per group basis. Moreover, if the value returned as model fit object is `NULL` or `NA`, plotting is skipped on a per group within panel basis.

In the case of fitted model objects of classes not explicitly supported, an attempt is made to find the usual accessors and/or fitted object members, and if found, either complete or partial support is frequently achieved. In this case a message is issued encouraging users to check the validity of the values extracted as the structure of fitted model objects belonging to different classes and the values returned by their accessors can vary, potentially resulting in decoding errors leading to the return of wrong values for estimates.

The argument to parameter `method` can be either the name of a function object, possibly using double colon notation in case its package is not attached, or a character string matching the function name for functions in the search path. This approach makes it possible to support model fit functions that are not dependencies of 'ggpmisc'. Either by attaching the package where the function is defined and passing it by name or as string, or using double colon notation when passing the name of the function.

User-defined functions can be passed as argument to parameter `method` as long as they have parameters `formula`, `data` `subset` and possibly `weights`. Additional arguments can be passed to any method as a named list through parameter `method.args`. As in `stat_smooth()` prior `weights` are passed to the model fit functions' `weights` (plural!) parameter by mapping a numeric variable to plot aesthetic `weight` (singular!).

Tables 1 lists natively supported model fit functions, with the caveat that only some 'broom' methods' specializations have been actually tested with statistics from 'ggpmisc'. In addition, the statistics based on 'broom' methods require the user to tailor their behaviour by passing additional arguments in the call and occasionally some detective work to find out the names of variables in the returned data frame as these names are set by methods from 'broom'.

Table 1. Model fit methods supported by the different statistics available in package 'ggpmisc'. Column *f* indicates whether computations are done by group (G) or by plot panel (P).

Statistic	*f*	Supported model fit methods
`stat_poly_line()`	G	"lm", "rlm", "lts", "sma", "ma", "gls", "nls", "onls", *others with methods* `predict()` or `fitt`
`stat_poly_eq()`	G	"lm", "rlm", "lts", "sma", "ma", "gls", "nls", "onls", *others with needed accesors*
`stat_quant_line()`	G	"rq", "rqss"
`stat_quant_band()`	G	"rq", "rqss"
`stat_quant_eq()`	G	"rq", "rqss"
`stat_ma_line()`	G	"SMA", "MA", "RMA", "OLS"
`stat_ma_eq()`	G	"SMA", "MA", "RMA", "OLS"
`stat_fit_residuals()`	G	"lm", "rlm", "lts", "sma", "ma", "gls", "rq", "rqss", "nls", "onls", *others with method* `residu`
`stat_fit_fitted()`	G	"lm", "rlm", "lts", "gls", "rq", "rqss", "nls", "onls", *others with method* `fitted()`

stat_fit_deviations()	G	"lm", "rlm", "lts", "gls", "rq", "rqss", "nls", "onls", *others with methods* fitted() and weigh
stat_fit_augment()	G	*any with 'broom' method* augment()
stat_fit_glance()	G	*any with 'broom' method* glance()
stat_fit_tidy()	G	*any with 'broom' method* tidy()
stat_fit_tb()	P	*any with 'broom' method* tidy()

The single colon notation is based on parsing the name and is available when passing the name of the fit method as a character string. In a string such as "head:tail" the "head" gives the name of the model fit function and the "tail" gives the argument to pass it's method parameter. This is only a convenience, as method.args can be also used. In some methods, i.e., splines, the default formula = y ~ x needs to be overridden by the user.

Table 2 lists the correspondence of pre-defined *method names* to model fit method functions. As mentioned above, these are only a subset of the model fit methods that are expected to work. When using these names there is no need for users to attach additional packages but the packages must be available (installed).

Table 2. Available predefined method names, the model fit functions they call, the packages where the functions reside, the class of the returned fitted model object and the arguments that can be passed to their method parameter using single colon notation.

Predefined method names	Model fit methods	R package	Object class
"lm", "lm:qr"	lm()	'stats'	"lm"
"rlm", "rlm:M", "rlm:MM"	rlm()	'MASS'	"rlm" ("lm")
"lts", "ltsReg"	ltsReg()	'robustbase'	"lts"
"ma", "sma", "sma:SMA", "sma:MA", "sma:OLS"	sma()	'smatr'	"ma" or "sma"
"gls", "gls:REML", "gls:ML"	gls()	'nlme'	"gls"
"rq", "rq:sfn", "rq:sfnc", "rq:lasso"	rq()	'quantreg'	"rq"
"rqss", "rqss:sfn", "rqss:sfnc", "rqss:lasso"	rqss()	'quantreg'	"rqss"
"SMA", "MA", "RMA", "OLS"	lmodel2()	'lmodel2'	("list")

Aesthetics

stat_fit_glance() understands the following aesthetics. Required aesthetics are displayed in bold and defaults are displayed for optional aesthetics:

- x
- y
- group → inferred
- hjust → "inward"
- npcx → after_stat(npcx)
- npcy → after_stat(npcy)
- vjust → "inward"

Learn more about setting these aesthetics in vignette("ggplot2-specs").

See Also

Package broom for details on how the tidying of the result of model fits is done.

Other 'broom'-based statistics for model fits: stat_fit_augment(), stat_fit_tb(), stat_fit_tidy()

Examples

```
# package 'broom' needs to be installed to run these examples

broom.installed <- requireNamespace("broom", quietly = TRUE)
gginnards.installed <- requireNamespace("gginnards", quietly = TRUE)

if (broom.installed) {
  library(broom)
}

# Regression by panel example
# simpler alternative: stat_poly_eq()
#
if (broom.installed)
  ggplot(mtcars, aes(x = disp, y = mpg)) +
    stat_poly_line() +
    geom_point(aes(colour = factor(cyl))) +
    stat_fit_glance(method = "lm",
                    label.y = "bottom",
                    method.args = list(formula = y ~ x),
                    mapping =
                      aes(label =
                        after_stat(
                          sprintf('italic(r)^2~"="~%.3f~~italic(P)~"="~%.2g',
                                  r.squared, p.value))),
                    parse = TRUE)

# Flipped regression by panel example
# simpler alternative: stat_poly_eq()
#
if (broom.installed)
  ggplot(mtcars, aes(x = disp, y = mpg)) +
    stat_poly_line(orientation = "y") +
    geom_point(aes(colour = factor(cyl))) +
    stat_fit_glance(method = "lm",
                    orientation = "y",
                    label.y = "bottom",
                    method.args = list(formula = y ~ x),
                    mapping =
                      aes(label =
                        after_stat(
                          sprintf('italic(r)^2~"="~%.3f~~italic(P)~"="~%.2g',
                                  r.squared, p.value))),
                    parse = TRUE)

# Regression by group example
# simpler alternative: stat_poly_eq()
```

```
#
if (broom.installed)
  ggplot(mtcars, aes(x = disp, y = mpg, colour = factor(cyl))) +
    stat_poly_line() +
    geom_point() +
    stat_fit_glance(method = "lm",
                    label.y = "bottom",
                    method.args = list(formula = y ~ x),
                    mapping = aes(label =
                                  after_stat(
                                    sprintf('r^2~"="~%.3f~~italic(P)~"="~%.2g',
                                    r.squared, p.value))),
                    parse = TRUE)

# Weighted regression example
# simpler alternative: stat_poly_eq()
#
if (broom.installed)
  ggplot(mtcars, aes(x = disp, y = mpg, weight = cyl)) +
    stat_poly_line() +
    geom_point(aes(colour = factor(cyl))) +
    stat_fit_glance(method = "lm",
                    label.y = "bottom",
                    method.args = list(formula = y ~ x,
                                       weights = quote(weight)),
                    mapping = aes(label =
                      after_stat(
                        sprintf('r^2~"="~%.3f~~italic(P)~"="~%.2g',
                                r.squared, p.value))),
                    parse = TRUE)

# correlation test
# simpler alternative: stat_correlation()
#
if (broom.installed)
  ggplot(mtcars, aes(x = disp, y = mpg)) +
    geom_point() +
    stat_fit_glance(method = "cor.test",
                    label.y = "bottom",
                    method.args = list(formula = ~ x + y),
                    mapping = aes(label =
                      after_stat(
                        sprintf('r[Pearson]~"="~%.3f~~italic(P)~"="~%.2g',
                                 estimate, p.value))),
                    parse = TRUE)

if (broom.installed)
  ggplot(mtcars, aes(x = disp, y = mpg)) +
    geom_point() +
    stat_fit_glance(method = "cor.test",
                    label.y = "bottom",
                    method.args = list(formula = ~ x + y,
                                       method = "spearman",
```

```
                                exact = FALSE),
                    mapping = aes(label =
                      after_stat(
                        sprintf('r[Spearman]~"="~%.3f~~italic(P)~"="~%.2g',
                                estimate, p.value))),
                    parse = TRUE)

# Inspecting the returned data using geom_debug_group()
#
if (gginnards.installed) {
    library(gginnards)
}

if (broom.installed && gginnards.installed) {
    ggplot(mtcars, aes(x = disp, y = mpg)) +
      stat_smooth(method = "lm") +
      geom_point(aes(colour = factor(cyl))) +
      stat_fit_glance(method = "lm",
                      method.args = list(formula = y ~ x),
                      geom = "debug_group")
}
```

---

stat_fit_tb *Fitted-model summary and ANOVA tables*

Description

stat_fit_tb() fits a model and returns a "tidy" version of the model's summary or ANOVA table, using 'tidy() methods from packages 'broom', 'broom.mixed', or other 'broom' extensions. The annotation is added to the plots in tabular form.

Usage

```
stat_fit_tb(
  mapping = NULL,
  data = NULL,
  geom = "table_npc",
  position = "identity",
  ...,
  orientation = NA,
  method = "lm",
  method.args = list(formula = y ~ x),
  n.min = 2L,
  fit.seed = NA,
  tidy.args = list(),
  tb.type = "fit.summary",
  tb.vars = NULL,
  tb.params = NULL,
```

```
  digits = 3,
  p.digits = digits,
  label.x = "center",
  label.y = "top",
  table.theme = NULL,
  table.rownames = FALSE,
  table.colnames = TRUE,
  table.hjust = 1,
  parse = FALSE,
  na.rm = FALSE,
  show.legend = FALSE,
  inherit.aes = TRUE
)
```

Arguments

mapping	The aesthetic mapping, usually constructed with aes(). Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset, only needed if you want to override the plot defaults.
geom	The geometric object to use display the data
position	The position adjustment to use for overlapping points on this layer.
...	other arguments passed on to layer. This can include aesthetics whose values you want to set, not map. See layer for more details.
orientation	character Either "x" or "y" controlling the default for formula. The letter indicates the aesthetic considered the explanatory variable in the model fit.
method	function or character If character, "lm", "rlm", "lmrob", "lts", "gls", "ma", "sma", "segreg", "rq" or the name of a model fit function are accepted, possibly followed by the fit function's method argument separated by a colon (e.g. "rlm:M"). If a function is different to lm(), rlm(), ltsReg(), gls(), ma, sma, it must have formal parameters named formula, data, and weights. See Details.
method.args, tidy.args	lists of arguments to pass to method and to tidy().
n.min	integer Minimum number of distinct values in the explanatory variable (on the rhs of formula) for fitting to the attempted.
fit.seed	RNG seed argument passed to set.seed(). Defaults to NA, indicating that set.seed() should not be called.
tb.type	character One of "fit.summary", "fit.anova" or "fit.coefs".
tb.vars, tb.params	character or numeric vectors, optionally named, used to select and/or rename the columns or the parameters in the table returned.
digits	integer indicating the number of significant digits to be used for all numeric values in the table.
p.digits	integer indicating the number of decimal places to round p-values to, with those rounded to zero displayed as the next larger possible value preceded by "<". If p.digits is outside the range 1..22 no rounding takes place.

label.x, label.y
: numeric with range 0..1 "normalized parent coordinates" (npc units) or character if using geom_text_npc() or geom_label_npc(). If using geom_text() or geom_label() numeric in native data units. If too short they will be recycled.

table.theme
: NULL, list or function A 'gridExtra' ttheme definition, or a constructor for a ttheme or NULL for default.

table.rownames, table.colnames
: logical flag to enable or disabling printing of row names and column names.

table.hjust
: numeric Horizontal justification for the core and column headings of the table.

parse
: logical Passed to the geom. If TRUE, the labels will be parsed into expressions and displayed as described in plotmath. Default is TRUE if output.type = "expression" and FALSE otherwise.

na.rm
: a logical indicating whether NA values should be stripped before the computation proceeds.

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.

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.

Details

stat_fit_tb() Applies a model fitting function per panel, using the grouping factors from aesthetic mappings in the fitted model. When continuous variables are mapped to both x and y the argument passed to orientation determines if the data are flipped or not before fitting the model. When a factor is mapped to one of x or y the orientation is set automatically, with the factor used as the *x* in the model equation. In the first case a regression model is fitted, and in the second case an analysis of variance is performed to test for differences among groups.

The argument to method can be any fit method for which a suitable tidy() method is available, including non-linear regression. Fit methods retain their default arguments unless overridden with named arguments passed in list as argument to method.args.

As an ANOVA or summary table of estimated parameter values takes a considerable space in the plotting canvas, in most cases either the scale expansion or its limits need to be manually modified to ensure that other elements in the plot, such as the observations are not occluded or interfere. The effect can be different: manually setting the limits fixes them, while adding a multiplicative expansion to the scale modifies the width of the empty margin between the limits and the edge of the plotting area, with limits set automatically based on the data.

Value

A tibble with columns named fm.tb (a tibble returned by tidy() with possibly renamed and subset columns and rows, within a list), fm.tb.type (copy of argument passed to tb.type), fm.class (the class of the fitted model object), fm.method (the fit function's name), fm.call (the call if available), x and y.

To explore the values returned by this statistic, which vary depending on the model fitting function and model formula we suggest the use of geom_debug.

Computed variables

The output of `tidy()` is returned as a single "cell" in a tibble (i.e., a tibble nested within a tibble). The returned `data` object contains a single tibble, containing the result from a single model fit to all data in a panel. If grouping is present, it is ignored in the sense of returning a single table, but the grouping aesthetic can be a term in the fitted model.

Model formula and model fitting

A ggplot statistic receives as `data` a data frame that is not the one passed as argument by the user, but instead a data frame with the variables mapped to aesthetics. In `stat_poly_eq()` the compute function is applied by group, each call "seeing" the subset of `data` for an individual group. As supported models are for regression lines, variables mapped to `x` and `y` should both be continuous, i.e., numeric or date time and model formulas defined using `x` and `y` as variable names.

The interpretation of the argument passed to `formula` is enhanced compared to `stat_smooth()`. Formulas with `x` as explanatory variable work as in `stat_smooth()` but formulas with `y` as explanatory variable are also accepted. `orientation` is set automatically based on which explanatory variable appears in the formula. Spline-based smoothers are only partially supported.

Model fit methods supported

Several model fit functions are supported explicitly (see tables), and some of their differences smoothed out. Compatibility is checked late, based on the class of the returned fitted model object. This makes it possible to use wrapper functions that do model selection or other adjustments to the fit procedure on a per panel or per group basis. Moreover, if the value returned as model fit object is `NULL` or `NA`, plotting is skipped on a per group within panel basis.

In the case of fitted model objects of classes not explicitly supported, an attempt is made to find the usual accessors and/or fitted object members, and if found, either complete or partial support is frequently achieved. In this case a message is issued encouraging users to check the validity of the values extracted as the structure of fitted model objects belonging to different classes and the values returned by their accessors can vary, potentially resulting in decoding errors leading to the return of wrong values for estimates.

The argument to parameter `method` can be either the name of a function object, possibly using double colon notation in case its package is not attached, or a character string matching the function name for functions in the search path. This approach makes it possible to support model fit functions that are not dependencies of 'ggpmisc'. Either by attaching the package where the function is defined and passing it by name or as string, or using double colon notation when passing the name of the function.

User-defined functions can be passed as argument to parameter `method` as long as they have parameters `formula`, `data` `subset` and possibly `weights`. Additional arguments can be passed to any method as a named list through parameter `method.args`. As in `stat_smooth()` prior `weights` are passed to the model fit functions' `weights` (plural!) parameter by mapping a numeric variable to plot aesthetic `weight` (singular!).

Tables 1 lists natively supported model fit functions, with the caveat that only some 'broom' methods' specializations have been actually tested with statistics from 'ggpmisc'. In addition, the statistics based on 'broom' methods require the user to tailor their behaviour by passing additional arguments in the call and occasionally some detective work to find out the names of variables in the returned data frame as these names are set by methods from 'broom'.

Table 1. Model fit methods supported by the different statistics available in package 'ggpmisc'. Column *f* indicates whether computations are done by group (G) or by plot panel (P).

Statistic	*f*	Supported model fit methods
stat_poly_line()	G	"lm", "rlm", "lts", "sma", "ma", "gls", "nls", "onls", *others with methods* predict() or fitt
stat_poly_eq()	G	"lm", "rlm", "lts", "sma", "ma", "gls", "nls", "onls", *others with needed accesors*
stat_quant_line()	G	"rq", "rqss"
stat_quant_band()	G	"rq", "rqss"
stat_quant_eq()	G	"rq", "rqss"
stat_ma_line()	G	"SMA", "MA", "RMA", "OLS"
stat_ma_eq()	G	"SMA", "MA", "RMA", "OLS"
stat_fit_residuals()	G	"lm", "rlm", "lts", "sma", "ma", "gls", "rq", "rqss", "nls", "onls", *others with method* residu
stat_fit_fitted()	G	"lm", "rlm", "lts", "gls", "rq", "rqss", "nls", "onls", *others with method* fitted()
stat_fit_deviations()	G	"lm", "rlm", "lts", "gls", "rq", "rqss", "nls", "onls", *others with methods* fitted() and weigh
stat_fit_augment()	G	*any with 'broom' method* augment()
stat_fit_glance()	G	*any with 'broom' method* glance()
stat_fit_tidy()	G	*any with 'broom' method* tidy()
stat_fit_tb()	P	*any with 'broom' method* tidy()

The single colon notation is based on parsing the name and is available when passing the name of the fit method as a character string. In a string such as "head:tail" the "head" gives the name of the model fit function and the "tail" gives the argument to pass it's `method` parameter. This is only a convenience, as `method.args` can be also used. In some methods, i.e., splines, the default `formula = y ~ x` needs to be overridden by the user.

Table 2 lists the correspondence of pre-defined *method names* to model fit method functions. As mentioned above, these are only a subset of the model fit methods that are expected to work. When using these names there is no need for users to attach additional packages but the packages must be available (installed).

Table 2. Available predefined method names, the model fit functions they call, the packages where the functions reside, the class of the returned fitted model object and the arguments that can be passed to their `method` parameter using single colon notation.

Predefined method names	Model fit methods	R package	Object class
"lm", "lm:qr"	lm()	'stats'	"lm"
"rlm", "rlm:M", "rlm:MM"	rlm()	'MASS'	"rlm" ("lm")
"lts", "ltsReg"	ltsReg()	'robustbase'	"lts"
"ma", "sma", "sma:SMA", "sma:MA", "sma:OLS"	sma()	'smatr'	"ma" or "sma"
"gls", "gls:REML", "gls:ML"	gls()	'nlme'	"gls"
"rq", "rq:sfn", "rq:sfnc", "rq:lasso"	rq()	'quantreg'	"rq"
"rqss", "rqss:sfn", "rqss:sfnc", "rqss:lasso"	rqss()	'quantreg'	"rqss"
"SMA", "MA", "RMA", "OLS"	lmodel2()	'lmodel2'	("list")

Which variables are available for mapping?

Computed variables and their names vary following those used in the specializations of 'broom' methods. A message is issued by default in interactive R sessions, listing the names of all variables accessible by `after_stat()` within a call to `aes()` with at least some non-missing values.

This default ("colnames") can be changed by setting R option "ggpmisc.stat.vars.message" to "none".

Aesthetics

stat_fit_tb() understands the following aesthetics. Required aesthetics are displayed in bold and defaults are displayed for optional aesthetics:

- x
- y
- group → inferred
- hjust → "inward"
- label → after_stat(fm.tb)
- vjust → "inward"

Learn more about setting these aesthetics in vignette("ggplot2-specs").

See Also

Package broom for details on how the tidying of the result of model fits is done. See geom_table for details on how inset tables respond to mapped aesthetics and table themes. For details on predefined table themes see ttheme_gtdefault.

Other 'broom'-based statistics for model fits: stat_fit_augment(), stat_fit_glance(), stat_fit_tidy()

Examples

```
# Package 'broom' needs to be installed to run these examples.
# We check availability before running them to avoid errors.
broom.installed <- requireNamespace("broom", quietly = TRUE)

if (broom.installed)
  library(broom)

# data for examples
  response <- c(44.4, 45.9, 41.9, 53.3, 44.7, 44.1, 50.7, 45.2, 60.1)
  covariate <- sqrt(response) + rnorm(9)
  group <- factor(c(rep("A", 4), rep("B", 5)))
  my.df <- data.frame(response, group, covariate)

## covariate is a numeric or continuous variable
# Linear regression fit summary, all defaults
if (broom.installed)
  ggplot(my.df, aes(covariate, response)) +
    geom_point() +
    stat_fit_tb() +
    expand_limits(y = 65) # make space

if (broom.installed)
  ggplot(my.df, aes(covariate, response)) +
    geom_point() +
```

```
    stat_fit_tb(orientation = "y") + # data flipped for regression
    expand_limits(y = 65) # make space

if (broom.installed)
  ggplot(my.df, aes(covariate, response)) +
    geom_point() +
    stat_fit_tb() +
    scale_y_continuous(expand = expansion(mult = c(0.075, 0.3))) # make space

# Linear regression fit summary, with default formatting
if (broom.installed)
  ggplot(my.df, aes(covariate, response)) +
    geom_point() +
    stat_fit_tb(tb.type = "fit.summary") +
    expand_limits(y = 65)

# Linear regression fit summary, with manual table formatting
if (broom.installed)
  ggplot(my.df, aes(covariate, response)) +
    geom_point() +
    stat_fit_tb(digits = 2,
                p.digits = 4,
                tb.params = c("intercept" = 1, "covariate" = 2),
                tb.vars = c(Term = 1, Estimate = 2,
                            "italic(s)" = 3, "italic(t)" = 4,
                            "italic(P)" = 5),
                parse = TRUE) +
    expand_limits(y = 65)

# Linear regression ANOVA table, with default formatting
if (broom.installed)
  ggplot(my.df, aes(covariate, response)) +
    geom_point() +
    stat_fit_tb(tb.type = "fit.anova") +
    expand_limits(y = 65)

# Linear regression ANOVA table, with manual table formatting
if (broom.installed)
  ggplot(my.df, aes(covariate, response)) +
    geom_point() +
    stat_fit_tb(tb.type = "fit.anova",
                tb.params = c("Covariate" = 1, 2),
                tb.vars = c(Effect = 1, d.f. = 2,
                            "M.S." = 4, "italic(F)" = 5,
                            "italic(P)" = 6),
                parse = TRUE) +
    expand_limits(y = 65)

# Linear regression fit coeficients, with default formatting
if (broom.installed)
  ggplot(my.df, aes(covariate, response)) +
    geom_point() +
    stat_fit_tb(tb.type = "fit.coefs") +
```

```
    expand_limits(y = 65)

# Linear regression fit coeficients, with manual table formatting
if (broom.installed)
  ggplot(my.df, aes(covariate, response)) +
    geom_point() +
    stat_fit_tb(tb.type = "fit.coefs",
                tb.params = c(a = 1, b = 2),
                tb.vars = c(Term = 1, Estimate = 2)) +
    expand_limits(y = 65)

## response is also a numeric or continuous variable
# Polynomial regression, with default formatting
if (broom.installed)
  ggplot(my.df, aes(covariate, response)) +
    geom_point() +
    stat_fit_tb(method.args = list(formula = y ~ poly(x, 2))) +
    expand_limits(y = 65)

# Polynomial regression, with manual table formatting
if (broom.installed)
  ggplot(my.df, aes(covariate, response)) +
    geom_point() +
    stat_fit_tb(method.args = list(formula = y ~ poly(x, 2)),
                tb.params = c("x^0" = 1, "x^1" = 2, "x^2" = 3),
                tb.vars = c("Term" = 1, "Estimate" = 2, "S.E." = 3,
                            "italic(t)" = 4, "italic(P)" = 5),
                parse = TRUE) +
    expand_limits(y = 65)

## group is a factor or discrete variable
# ANOVA summary, with default formatting
if (broom.installed)
  ggplot(my.df, aes(group, response)) +
    geom_point() +
    stat_fit_tb() +
    expand_limits(y = 65)

if (broom.installed)
  ggplot(my.df, aes(response, group)) +
    geom_point() +
    stat_fit_tb() +
    expand_limits(y = 2.5)

# ANOVA table, with default formatting
if (broom.installed)
  ggplot(my.df, aes(group, response)) +
    geom_point() +
    stat_fit_tb(tb.type = "fit.anova") +
    expand_limits(y = 65)

# ANOVA table, with manual table formatting
if (broom.installed)
```

```
  ggplot(my.df, aes(group, response)) +
    geom_point() +
    stat_fit_tb(tb.type = "fit.anova",
                tb.vars = c(Effect = "term", "df", "italic(F)" = "statistic",
                            "italic(P)" = "p.value"),
                tb.params = c(Group = 1, Error = 2),
                parse = TRUE)

# ANOVA table, with manual table formatting
# using column names with partial matching
if (broom.installed)
  ggplot(my.df, aes(group, response)) +
    geom_point() +
    stat_fit_tb(tb.type = "fit.anova",
                tb.vars = c(Effect = "term", "df", "italic(F)" = "stat",
                            "italic(P)" = "p"),
                tb.params = c(Group = "x", Error = "Resid"),
                parse = TRUE)

# ANOVA summary, with default formatting
if (broom.installed)
  ggplot(my.df, aes(group, response)) +
    geom_point() +
    stat_fit_tb() +
    expand_limits(y = 65)

## covariate is a numeric variable and group is a factor
# ANCOVA (covariate not plotted) ANOVA table, with default formatting
if (broom.installed)
  ggplot(my.df, aes(group, response, z = covariate)) +
    geom_point() +
    stat_fit_tb(tb.type = "fit.anova",
                method.args = list(formula = y ~ x + z))

# ANCOVA (covariate not plotted) ANOVA table, with manual table formatting
if (broom.installed)
  ggplot(my.df, aes(group, response, z = covariate)) +
    geom_point() +
    stat_fit_tb(tb.type = "fit.anova",
                method.args = list(formula = y ~ x + z),
                tb.vars = c(Effect = 1, d.f. = 2,
                            "M.S." = 4, "italic(F)" = 5,
                            "italic(P)" = 6),
                tb.params = c(Group = 1,
                              Covariate = 2,
                              Error = 3),
                parse = TRUE)

## group is a factor or discrete variable
# t-test, minimal output, with manual table formatting
if (broom.installed)
  ggplot(my.df, aes(group, response)) +
    geom_point() +
```

```
    stat_fit_tb(method = "t.test",
              tb.vars = c("italic(t)" = "statistic",
                          "italic(P)" = "p.value"),
              parse = TRUE)

# t-test, more detailed output, with manual table formatting
if (broom.installed)
  ggplot(my.df, aes(group, response)) +
    geom_point() +
    stat_fit_tb(method = "t.test",
              tb.vars = c("\"Delta \"*italic(x)" = "estimate",
                          "CI low" = "conf.low", "CI high" = "conf.high",
                          "italic(t)" = "statistic",
                          "italic(P)" = "p.value"),
              parse = TRUE) +
    expand_limits(y = 65)

# t-test (equal variances assumed), minimal output, with manual
# table formatting
if (broom.installed)
  ggplot(my.df, aes(group, response)) +
    geom_point() +
    stat_fit_tb(method = "t.test",
                method.args = list(formula = y ~ x, var.equal = TRUE),
                tb.vars = c("italic(t)" = "statistic",
                            "italic(P)" = "p.value"),
                parse = TRUE)

## covariate is a numeric or continuous variable
# Linear regression using a table theme and non-default position
if (broom.installed)
  ggplot(my.df, aes(covariate, response)) +
    geom_point() +
    stat_fit_tb(table.theme = ttheme_gtlight,
                npcx = "left", npcy = "bottom") +
    expand_limits(y = 35)
```

`stat_fit_tidy` *One row data frame with fitted parameter estimates*

Description

`stat_fit_tidy()` fits a model and returns a "tidy" version of the model's summary, using `tidy()` method specializations from packages 'broom', 'broom.mixed', or other sources. The role played by 'tidy()' in this statistics is similar to that played by R's `summary()`.

Usage

```
stat_fit_tidy(
```

```
  mapping = NULL,
  data = NULL,
  geom = "text_npc",
  position = "identity",
  ...,
  orientation = NA,
  method = "lm",
  method.args = list(formula = y ~ x),
  n.min = 2L,
  fit.seed = NA,
  tidy.args = list(),
  label.x = "left",
  label.y = "top",
  hstep = 0,
  vstep = NULL,
  sanitize.names = FALSE,
  na.rm = FALSE,
  show.legend = FALSE,
  inherit.aes = TRUE
)
```

Arguments

mapping	The aesthetic mapping, usually constructed with aes(). Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset, only needed if you want to override the plot defaults.
geom	The geometric object to use display the data
position	The position adjustment to use for overlapping points on this layer.
...	other arguments passed on to layer. This can include aesthetics whose values you want to set, not map. See layer for more details.
orientation	character Either "x" or "y" controlling the default for formula. The letter indicates the aesthetic considered the explanatory variable in the model fit.
method	function or character If character, "lm", "rlm", "lmrob", "lts", "gls", "ma", "sma", "segreg", "rq" or the name of a model fit function are accepted, possibly followed by the fit function's method argument separated by a colon (e.g. "rlm:M"). If a function is different to lm(), rlm(), ltsReg(), gls(), ma, sma, it must have formal parameters named formula, data, and weights. See Details.
method.args, tidy.args	list of arguments to pass to method, and to [generics::tidy], respectively.
n.min	integer Minimum number of distinct values in the explanatory variable (on the rhs of formula) for fitting to the attempted.
fit.seed	RNG seed argument passed to set.seed(). Defaults to NA, indicating that set.seed() should not be called.
label.x, label.y	numeric with range 0..1 "normalized parent coordinates" (npc units) or character if using geom_text_npc() or geom_label_npc(). If using geom_text() or geom_label() numeric in native data units. If too short they will be recycled.

hstep, vstep	numeric in npc units, the horizontal and vertical step used between labels for different groups.
sanitize.names	logical If true sanitize column names in the returned data with R's make.names() function.
na.rm	a logical indicating whether NA values should be stripped before the computation proceeds.
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.
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.

Details

stat_fit_tidy() together with stat_fit_glance() and stat_fit_augment, based on package 'broom' can be used with a broad range of model fitting functions as supported at any given time by 'broom'. In contrast to stat_poly_eq() which can generate formatted text labels automatically, for these functions the mapping of aesthetic label needs to be explicitly supplied in the call, and labels built on the fly.

Although arguments passed to parameter tidy.args will be passed to tidy() whether they are silently ignored or obeyed depends on each specialization of tidy(), so do carefully read the documentation for the version of tidy() corresponding to the method used to fit the model. You will also need to manually install the package, such as 'broom', where the tidier you intend to use are defined.

This statistics is meant to be used in situations when all of stat_poly_eq(), stat_quant_eq(), stat_ma_eq() and stat_correlation() are unsuitable.

Warning! Not all tidy() methods are defined in package 'broom'. glance() specializations for mixed models fits of classes "lme", "nlme", "lme4" and many others are defined in package 'broom.mixed'.

Value

The output of tidy() is returned after reshaping it into a single row. The returned data object has one row for each group within a panel. To use the intercept, note that output of tidy() is renamed from (Intercept) to Intercept. Otherwise, the names of the columns in the returned data are based on those returned by the tidy() method for the model fit class returned by the fit function. These will frequently differ from the name of values returned by the print methods corresponding to the fit or test function used. Names of columns as returned by default are not always syntactically valid R names making it necessary to use back ticks to access them. Syntactically valid names are guaranteed if sanitize.names = TRUE is added to the call.

To explore the values returned by this statistic, which vary depending on the model fitting function and model formula we suggest the use of geom_debug. An example is shown below.

Handling of grouping

stat_fit_tidy applies the function given by method separately to each group of observations; in ggplot2 factors mapped to aesthetics generate a separate group for each level. Because of this,

stat_fit_tidy is not useful for annotating plots with results from t.test() or ANOVA or ANCOVA. In such cases use instead stat_fit_tb() which applies the model fitting per panel.

Model formula and model fitting

A ggplot statistic receives as data a data frame that is not the one passed as argument by the user, but instead a data frame with the variables mapped to aesthetics. In stat_poly_eq() the compute function is applied by group, each call "seeing" the subset of data for an individual group. As supported models are for regression lines, variables mapped to x and y should both be continuous, i.e., numeric or date time and model formulas defined using x and y as variable names.

The interpretation of the argument passed to formula is enhanced compared to stat_smooth(). Formulas with x as explanatory variable work as in stat_smooth() but formulas with y as explanatory variable are also accepted. orientation is set automatically based on which explanatory variable appears in the formula. Spline-based smoothers are only partially supported.

Model fit methods supported

Several model fit functions are supported explicitly (see tables), and some of their differences smoothed out. Compatibility is checked late, based on the class of the returned fitted model object. This makes it possible to use wrapper functions that do model selection or other adjustments to the fit procedure on a per panel or per group basis. Moreover, if the value returned as model fit object is NULL or NA, plotting is skipped on a per group within panel basis.

In the case of fitted model objects of classes not explicitly supported, an attempt is made to find the usual accessors and/or fitted object members, and if found, either complete or partial support is frequently achieved. In this case a message is issued encouraging users to check the validity of the values extracted as the structure of fitted model objects belonging to different classes and the values returned by their accessors can vary, potentially resulting in decoding errors leading to the return of wrong values for estimates.

The argument to parameter method can be either the name of a function object, possibly using double colon notation in case its package is not attached, or a character string matching the function name for functions in the search path. This approach makes it possible to support model fit functions that are not dependencies of 'ggpmisc'. Either by attaching the package where the function is defined and passing it by name or as string, or using double colon notation when passing the name of the function.

User-defined functions can be passed as argument to parameter method as long as they have parameters formula, data subset and possibly weights. Additional arguments can be passed to any method as a named list through parameter method.args. As in stat_smooth() prior weights are passed to the model fit functions' weights (plural!) parameter by mapping a numeric variable to plot aesthetic weight (singular!).

Tables 1 lists natively supported model fit functions, with the caveat that only some 'broom' methods' specializations have been actually tested with statistics from 'ggpmisc'. In addition, the statistics based on 'broom' methods require the user to tailor their behaviour by passing additional arguments in the call and occasionally some detective work to find out the names of variables in the returned data frame as these names are set by methods from 'broom'.

Table 1. Model fit methods supported by the different statistics available in package 'ggpmisc'. Column *f* indicates whether computations are done by group (G) or by plot panel (P).

Statistic	*f*	Supported model fit methods
stat_poly_line()	G	"lm", "rlm", "lts", "sma", "ma", "gls", "nls", "onls", *others with methods* predict() or fitt
stat_poly_eq()	G	"lm", "rlm", "lts", "sma", "ma", "gls", "nls", "onls", *others with needed accesors*
stat_quant_line()	G	"rq", "rqss"
stat_quant_band()	G	"rq", "rqss"
stat_quant_eq()	G	"rq", "rqss"
stat_ma_line()	G	"SMA", "MA", "RMA", "OLS"
stat_ma_eq()	G	"SMA", "MA", "RMA", "OLS"
stat_fit_residuals()	G	"lm", "rlm", "lts", "sma", "ma", "gls", "rq", "rqss", "nls", "onls", *others with method* residu
stat_fit_fitted()	G	"lm", "rlm", "lts", "gls", "rq", "rqss", "nls", "onls", *others with method* fitted()
stat_fit_deviations()	G	"lm", "rlm", "lts", "gls", "rq", "rqss", "nls", "onls", *others with methods* fitted() and weigh
stat_fit_augment()	G	*any with 'broom' method* augment()
stat_fit_glance()	G	*any with 'broom' method* glance()
stat_fit_tidy()	G	*any with 'broom' method* tidy()
stat_fit_tb()	P	*any with 'broom' method* tidy()

The single colon notation is based on parsing the name and is available when passing the name of the fit method as a character string. In a string such as "head:tail" the "head" gives the name of the model fit function and the "tail" gives the argument to pass it's `method` parameter. This is only a convenience, as `method.args` can be also used. In some methods, i.e., splines, the default `formula = y ~ x` needs to be overridden by the user.

Table 2 lists the correspondence of pre-defined *method names* to model fit method functions. As mentioned above, these are only a subset of the model fit methods that are expected to work. When using these names there is no need for users to attach additional packages but the packages must be available (installed).

Table 2. Available predefined method names, the model fit functions they call, the packages where the functions reside, the class of the returned fitted model object and the arguments that can be passed to their `method` parameter using single colon notation.

Predefined method names	Model fit methods	R package	Object class
"lm", "lm:qr"	lm()	'stats'	"lm"
"rlm", "rlm:M", "rlm:MM"	rlm()	'MASS'	"rlm" ("lm")
"lts", "ltsReg"	ltsReg()	'robustbase'	"lts"
"ma", "sma", "sma:SMA", "sma:MA", "sma:OLS"	sma()	'smatr'	"ma" or "sma"
"gls", "gls:REML", "gls:ML"	gls()	'nlme'	"gls"
"rq", "rq:sfn", "rq:sfnc", "rq:lasso"	rq()	'quantreg'	"rq"
"rqss", "rqss:sfn", "rqss:sfnc", "rqss:lasso"	rqss()	'quantreg'	"rqss"
"SMA", "MA", "RMA", "OLS"	lmodel2()	'lmodel2'	("list")

Which variables are available for mapping?

Computed variables and their names vary following those used in the specializations of 'broom' methods. A message is issued by default in interactive R sessions, listing the names of all variables accessible by `after_stat()` within a call to `aes()` with at least some non-missing values. This default (`"colnames"`) can be changed by setting R option `"ggpmisc.stat.vars.message"` to `"none"`.

Aesthetics

`stat_fit_tidy()` understands the following aesthetics. Required aesthetics are displayed in bold and defaults are displayed for optional aesthetics:

- x
- y
- group → inferred
- hjust → "inward"
- npcx → after_stat(npcx)
- npcy → after_stat(npcy)
- vjust → "inward"

Learn more about setting these aesthetics in `vignette("ggplot2-specs")`.

See Also

Package broom for details on how the tidying of the result of model fits is done.

Other 'broom'-based statistics for model fits: `stat_fit_augment()`, `stat_fit_glance()`, `stat_fit_tb()`

Examples

```
# Package 'broom' needs to be installed to run these examples.
# We check availability before running them to avoid errors.

broom.installed <- requireNamespace("broom", quietly = TRUE)
gginnards.installed <- requireNamespace("gginnards", quietly = TRUE)

if (broom.installed) {
  library(broom)
}

# Regression by panel example
if (broom.installed)
  ggplot(mtcars, aes(x = disp, y = mpg)) +
    stat_smooth(method = "lm", formula = y ~ x) +
    geom_point(aes(colour = factor(cyl))) +
    stat_fit_tidy(method = "lm",
                  label.x = "right",
                  method.args = list(formula = y ~ x),
                  mapping = aes(label = after_stat(
                                    sprintf("Slope = %.3g\np-value = %.3g",
                                            x_estimate, x_p.value))))

# Flipped regression by panel example
if (broom.installed)
  ggplot(mtcars, aes(x = disp, y = mpg)) +
    stat_smooth(method = "lm", formula = y ~ x) +
    geom_point(aes(colour = factor(cyl))) +
    stat_fit_tidy(method = "lm",
```

```
                  orientation = "y",
                  label.x = "right",
                  method.args = list(formula = y ~ x),
                  mapping = aes(label = after_stat(
                                  sprintf("Slope = %.3g\np-value = %.3g",
                                          x_estimate, x_p.value))))

# Regression by group example
if (broom.installed)
  ggplot(mtcars, aes(x = disp, y = mpg, colour = factor(cyl))) +
    stat_smooth(method = "lm", formula = y ~ x) +
    geom_point() +
    stat_fit_tidy(method = "lm",
                  label.x = "right",
                  method.args = list(formula = y ~ x),
                  mapping = aes(label = after_stat(
                                  sprintf("Slope = %.3g, p-value = %.3g",
                                          x_estimate, x_p.value))))

# Weighted regression example
if (broom.installed)
  ggplot(mtcars, aes(x = disp, y = mpg, weight = cyl)) +
    stat_smooth(method = "lm", formula = y ~ x) +
    geom_point(aes(colour = factor(cyl))) +
    stat_fit_tidy(method = "lm",
                  label.x = "right",
                  method.args = list(formula = y ~ x, weights = quote(weight)),
                  mapping = aes(label = after_stat(
                                  sprintf("Slope = %.3g\np-value = %.3g",
                                          x_estimate, x_p.value))))

# Inspecting the returned data using geom_debug_group()
  if (gginnards.installed) {
    library(gginnards)
}

# Regression by panel, default column names
if (broom.installed && gginnards.installed)
  ggplot(mtcars, aes(x = disp, y = mpg)) +
    stat_smooth(method = "lm", formula = y ~ x + I(x^2)) +
    geom_point(aes(colour = factor(cyl))) +
    stat_fit_tidy(method = "lm",
                  method.args = list(formula = y ~ x + I(x^2)),
                  geom = "debug_group")
```

---

stat_ma_eq *Model II prediction and annotations*

Description

Statistics `stat_ma_line()` and `stat_ma_eq()` fit model II regressions. While `stat_ma_line()` adds a prediction line and band, `stat_ma_eq()` adds textual labels to a plot.

Usage

```
stat_ma_eq(
  mapping = NULL,
  data = NULL,
  geom = "text_npc",
  position = "identity",
  ...,
  orientation = NA,
  formula = NULL,
  method = "lmodel2:MA",
  method.args = list(),
  n.min = 2L,
  range.y = NULL,
  range.x = NULL,
  nperm = 99,
  fit.seed = NA,
  eq.with.lhs = TRUE,
  eq.x.rhs = NULL,
  small.r = getOption("ggpmisc.small.r", default = FALSE),
  small.p = getOption("ggpmisc.small.p", default = FALSE),
  coef.digits = 3,
  coef.keep.zeros = TRUE,
  decreasing = getOption("ggpmisc.decreasing.poly.eq", FALSE),
  rr.digits = 2,
  theta.digits = 2,
  p.digits = max(1, ceiling(log10(nperm))),
  label.x = "left",
  label.y = "top",
  hstep = 0,
  vstep = NULL,
  output.type = NULL,
  na.rm = FALSE,
  parse = NULL,
  show.legend = FALSE,
  inherit.aes = TRUE
)

stat_ma_line(
  mapping = NULL,
  data = NULL,
  geom = "smooth",
  position = "identity",
  ...,
```

```
  orientation = NA,
  method = "lmodel2:MA",
  method.args = list(),
  n.min = 2L,
  formula = NULL,
  range.y = NULL,
  range.x = NULL,
  se = TRUE,
  fit.seed = NA,
  fm.values = FALSE,
  n = 80,
  nperm = 99,
  fullrange = FALSE,
  limit.to = NULL,
  level = 0.95,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	The aesthetic mapping, usually constructed with aes(). Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset, only needed if you want to override the plot defaults.
geom	The geometric object to use display the data
position	The position adjustment to use for overlapping points on this layer.
...	other arguments passed on to layer. This can include aesthetics whose values you want to set, not map. See layer for more details.
orientation	character Either "x" or "y" controlling the default for formula. The letter indicates the aesthetic considered the explanatory variable in the model fit.
formula	a formula object. Using aesthetic names x and y instead of original variable names.
method	function or character If character, "MA", "SMA" , "RMA" or "OLS", alternatively "lmodel2" or the name of a model fit function are accepted, possibly followed by the fit function's method argument separated by a colon (e.g. "lmodel2:MA"). If a function different to lmodel2(), it must accept arguments named formula, data, range.y, range.x and nperm and return a model fit object of class lmodel2.
method.args	named list with additional arguments. Not data or weights which are always passed through aesthetic mappings.
n.min	integer Minimum number of distinct values in the explanatory variable (on the rhs of formula) for fitting to the attempted.
range.y, range.x	character Pass "relative" or "interval" if method "RMA" is to be computed.
nperm	integer Number of permutation used to estimate significance.

fit.seed RNG seed argument passed to set.seed(). Defaults to NA, indicating that set.seed() should not be called.

eq.with.lhs If character the string is pasted to the front of the equation label before parsing or a logical (see note).

eq.x.rhs character this string will be used as replacement for "x" in the model equation when generating the label before parsing it.

small.r, small.p
logical Flags to switch use of lower case r and p for coefficient of determination and p-value.

coef.digits integer Number of significant digits to use for the fitted coefficients in the equation label.

coef.keep.zeros
logical Keep or drop trailing zeros when formatting the fitted coefficients and F-value.

decreasing logical It specifies the order of the terms in the returned character string; in increasing (default) or decreasing powers.

rr.digits, theta.digits, p.digits
integer Number of digits after the decimal point to use for R^2, theta and P-value in labels. If Inf, use exponential notation with three decimal places.

label.x, label.y
numeric with range 0..1 "normalized parent coordinates" (npc units) or character if using geom_text_npc() or geom_label_npc(). If using geom_text() or geom_label() numeric in native data units. If too short they will be recycled.

hstep, vstep numeric in npc units, the horizontal and vertical step used between labels for different groups.

output.type character One of "expression", "text", "markdown", "marquee", "latex", "latex.eqn", "latex.deqn" or "numeric".

na.rm a logical indicating whether NA values should be stripped before the computation proceeds.

parse logical Passed to the geom. If TRUE, the labels will be parsed into expressions and displayed as described in plotmath. Default is TRUE if output.type = "expression" and FALSE otherwise.

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.

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.

se logical Return confidence interval around smooth? ('TRUE' by default, see 'level' to control.)

fm.values logical Add metadata and parameter estimates extracted from the fitted model object; FALSE by default.

n Number of points at which to predict with the fitted model.

fullrange logical Should the fit prediction span the full range of the plot, or just the range of the explanatory variable?

limit.to — character or numeric If character one of "", "x", "y" or "xy". Should the fit prediction be constrained to the range of the variables mapped to x and/or y in each data group? If numeric, the *new data* values to use for the explanatory variable when computing the predicted line and confidence band. When set, limit.to silently overrides fullrange!

level — Level of confidence interval to use (only 0.95 currently).

Details

Statistics stat_ma_line() and stat_ma_eq fit major axis ("MA") and other model II regressions with function lmodel2 from package 'lmodel2'. They support linear major axis (MA), standard major axis (SMA) and ranged major axis (RMA) regression. MA and SMA regressions are supported also by stat_poly_line() and stat_poly_eq() using package 'smatr' instead of 'lmodel2'.

stat_ma_line() adds the predicted line and confidence band based on the uncertainty of the slope estimate.stat_ma_eq() adds textual annotations with the fitted model equation and other parameter estimates.

Model II regression is called for when both x and y are subject to random variation and the intention is not to predict y from x by means of the model but rather to study the relationship between two independent variables. A frequent case in biology are allometric relationships among body parts.

As the fitted line is the same whether x or y is on the rhs of the model equation, orientation even if accepted does not have an effect on the fitted line. It does, however, have an effect on the formulation of the equation displayed in the label.

The minimum number of observations with distinct values can be set through parameter n.min. The default n.min = 3L is the smallest possible value. However, model fits with very few observations are of little interest and using a larger number for n.min than the default is wise. As model fitting functions could depend on the RNG, fit.seed if different to NA is used as argument in a call to set.seed() immediately ahead of model fitting.

In lmodel2() MA, SMA and OLS fits always computed while RMA requires a numeric argument to at least one of range.y or range.x. The statistics extract estimates for one of the methods based on the argument for method.

Package 'lmodel2' implements a model fit function and fitted model object that differ from the usual approach of R. Thus, their use was implemented as a separate pair of statistics.

Value

stat_ma_eq() returns data frame with a single row and columns as described below. stat_ma_line() returns a data frame with n rows. In cases when the number of observations is less than n.min or when the model fit method returns NA or NULL, a data frame with no rows or columns is returned and rendered as an empty/invisible plot layer.

Variables returned by 'stat_ma_line()'

Some of the variables can have missing values or depend on orientation and/or method.

y or x predicted value

ymin or xmin lower pointwise confidence interval around the mean

ymax or xmax upper pointwise confidence interval around the mean

se standard error

If `fm.values = TRUE` is passed then columns based on the summary of the model fit are added, with the same value in each row within a group. This is wasteful and disabled by default, but provides a simple and robust approach to achieve effects like colouring or hiding of the model fit line based on P-values, r-squared or the number of observations.

Variables returned by 'stat_ma_eq()'

If `output.type` is `"numeric"` the returned tibble contains columns listed below. If the model fit function used does not return a value, the variable is set to `NA_real_`.

x,npcx x position

y,npcy y position

coef.ls list containing the "coefficients" matrix from the summary of the fit object

r.squared, theta, p.value, n numeric values, from the model fit object

grp.label Set according to mapping in `aes`.

b_0.constant TRUE is polynomial is forced through the origin

b_i One or two columns with the coefficient estimates

If `output.type` is different from `"numeric"` the returned tibble contains columns listed below. If the fitted model does not contain a given value, the label is set to `character(0L)`.

x,npcx x position

y,npcy y position

eq.label equation for the fitted polynomial as a character string to be parsed

rr.label R^2 of the fitted model as a character string to be parsed

p.value.label P-value if available, depends on `method`.

theta.label Angle in degrees between the two OLS lines for lines estimated from `y ~ x` and `x ~ y` linear model (`lm`) fits.

n.label Number of observations used in the fit.

grp.label Set according to mapping in `aes`.

method.label Set according `method` used.

r.squared, theta, p.value, n numeric values, from the model fit object

To explore the computed values returned for a given input we suggest the use of `geom_debug()` as shown in the last examples below.

Output types

The formatting of character strings to be displayed in plots are marked as mathematical equations. Depending on the geom used, the mark-up needs to be encoded differently, or in some cases mark-up not applied.

`"expression"` The labels are encoded as character strings to be parsed into R's plotmath expressions.

"LaTeX", "TeX", "tikz", "latex" The labels are encoded as 'LaTeX' maths equations, without the "fences" for switching in math mode.

"latex.eqn" Same as "latex" but enclosed in single $, i.e., as in-line maths.

"latex.deqn" Same as "latex" but enclosed in double $$, i.e., as display maths.

"markdown" The labels are encoded as character strings using markdown syntax, with some embedded HTML.

"marquee" The labels are encoded as character strings using markdown syntax, with 'marquee' supported spans.

"text" The labels are plain ASCII character strings.

"numeric" No labels are generated. This value is accepted by the statistics, but not by the label formatting functions.

NULL The value used depends on the argument passed to geom.

If geom = "latex" (package 'xdvir') the output type used is "latex.eqn". If geom = "richtext" (package 'ggtext') or geom = "textbox" (package 'ggtext') the output type used is "markdown". If geom = "marquee" (package 'marquee') the output type used is "marquee". For all other values of geom the default is "expression". Invalid values as argument trigger an error.

Model formula and model fitting

A ggplot statistic receives as data a data frame that is not the one passed as argument by the user, but instead a data frame with the variables mapped to aesthetics. In stat_poly_eq() the compute function is applied by group, each call "seeing" the subset of data for an individual group. As supported models are for regression lines, variables mapped to x and y should both be continuous, i.e., numeric or date time and model formulas defined using x and y as variable names.

The interpretation of the argument passed to formula is enhanced compared to stat_smooth(). Formulas with x as explanatory variable work as in stat_smooth() but formulas with y as explanatory variable are also accepted. orientation is set automatically based on which explanatory variable appears in the formula. Spline-based smoothers are only partially supported.

Model equation label

By default the equation label uses as symbols the names of the aesthetics, x and y. However, "x" and "y" can be substituted by providing a replacement character string for the right-hand-side and left-hand-side through eq.x.rhs and eq.with.lhs, respectively. For backward compatibility a logical is also accepted as argument for eq.with.lhs, with FALSE suppressing the left-hand-side.

If the model formula includes a transformation of the explanatory variable in its right-hand-side (rhs), a matching argument should be passed to parameter eq.x.rhs as its default value would result in an equation label that does not reflect the applied transformation. In most cases, a transformation should not be applied within the left hand side (lhs) of the model formula, but instead in the mapping of the response variable within aes. In this case it may be necessary to also pass a matching argument to parameter eq.with.lhs.

Parameter orientation is redundant as the orientation can be set by the formula but is included for consistency with ggplot2::stat_smooth().

Position of labels

When data are grouped by mapping a factor to an aesthetic, e.g., `colour`, `shape` and/or `linetype` the model is fitted separately to each group, and for each group a whole set of labels is generated. If the argument passed to `label.y` is a vector of length 1, this value determines the position of the equation and/or other labels for the first group, and the positions of the labels for the remaining groups are generated by adding `vspace` based on the group number. If the argument passed to `label.y` is a vector of length > 1, it is used unchanged, possibly extended by recycling, ignoring `vstep`.

If the labels are rotated by 90 degrees then the automatic stepping is best based on `hstep` with `vstep = 0`. Similarly as described above, if `label.x` is a vector of length > 1, it is used unchanged, possibly extended by recycling, ignoring `hstep`.

When using facets and with a grouping that does not repeat in each panel, the automatic positioning in most cases will not be the desired one. Manual positioning using a vector of length > 1 for `label.x` and/or `label.y` is the currently available workaround.

Range of the prediction line

The range of the prediction line is controlled by parameters `fullrange` and `limit.to`. `fullrange` is backwards compatible both with earlier versions of 'ggpmisc' and with `stat_smooth()` from 'ggplot2'; an argument passed to `limit.to` overrides `fullrange` making it possible to constrain the range to that of `x`, `y`, or both simultaneously, with `"x"`, `"y"`, or `"xy"`, respectively, as argument. `limit.to` also accepts a numeric vector of values to be used as `newdata` when computing the prediction. Limiting the range based on both aesthetics is the best approach for major axis regression (MA, SMA, RMA) but can occasionally be useful also with some other methods when slopes are very steep and error variance in the explanatory variable is large. A numeric vector can be used to predict the response at specific values of the explanatory variable. If a single or very few values are predicted, it can be necessary to override the default `geom = "smooth"` with `geom = "pointrange"`.

Model fit methods supported

Several model fit functions are supported explicitly (see tables), and some of their differences smoothed out. Compatibility is checked late, based on the class of the returned fitted model object. This makes it possible to use wrapper functions that do model selection or other adjustments to the fit procedure on a per panel or per group basis. Moreover, if the value returned as model fit object is `NULL` or `NA`, plotting is skipped on a per group within panel basis.

In the case of fitted model objects of classes not explicitly supported, an attempt is made to find the usual accessors and/or fitted object members, and if found, either complete or partial support is frequently achieved. In this case a message is issued encouraging users to check the validity of the values extracted as the structure of fitted model objects belonging to different classes and the values returned by their accessors can vary, potentially resulting in decoding errors leading to the return of wrong values for estimates.

The argument to parameter `method` can be either the name of a function object, possibly using double colon notation in case its package is not attached, or a character string matching the function name for functions in the search path. This approach makes it possible to support model fit functions that are not dependencies of 'ggpmisc'. Either by attaching the package where the function is defined and passing it by name or as string, or using double colon notation when passing the name of the function.

User-defined functions can be passed as argument to parameter `method` as long as they have parameters `formula`, `data` `subset` and possibly `weights`. Additional arguments can be passed to any method as a named list through parameter `method.args`. As in `stat_smooth()` prior `weights` are passed to the model fit functions' `weights` (plural!) parameter by mapping a numeric variable to plot aesthetic `weight` (singular!).

Tables 1 lists natively supported model fit functions, with the caveat that only some 'broom' methods' specializations have been actually tested with statistics from 'ggpmisc'. In addition, the statistics based on 'broom' methods require the user to tailor their behaviour by passing additional arguments in the call and occasionally some detective work to find out the names of variables in the returned data frame as these names are set by methods from 'broom'.

Table 1. Model fit methods supported by the different statistics available in package 'ggpmisc'. Column *f* indicates whether computations are done by group (G) or by plot panel (P).

Statistic	*f*	Supported model fit methods
`stat_poly_line()`	G	"lm", "rlm", "lts", "sma", "ma", "gls", "nls", "onls", *others with methods* `predict()` or `fitt`
`stat_poly_eq()`	G	"lm", "rlm", "lts", "sma", "ma", "gls", "nls", "onls", *others with needed accesors*
`stat_quant_line()`	G	"rq", "rqss"
`stat_quant_band()`	G	"rq", "rqss"
`stat_quant_eq()`	G	"rq", "rqss"
`stat_ma_line()`	G	"SMA", "MA", "RMA", "OLS"
`stat_ma_eq()`	G	"SMA", "MA", "RMA", "OLS"
`stat_fit_residuals()`	G	"lm", "rlm", "lts", "sma", "ma", "gls", "rq", "rqss", "nls", "onls", *others with method* `residu`
`stat_fit_fitted()`	G	"lm", "rlm", "lts", "gls", "rq", "rqss", "nls", "onls", *others with method* `fitted()`
`stat_fit_deviations()`	G	"lm", "rlm", "lts", "gls", "rq", "rqss", "nls", "onls", *others with methods* `fitted()` and `weigh`
`stat_fit_augment()`	G	*any with 'broom' method* `augment()`
`stat_fit_glance()`	G	*any with 'broom' method* `glance()`
`stat_fit_tidy()`	G	*any with 'broom' method* `tidy()`
`stat_fit_tb()`	P	*any with 'broom' method* `tidy()`

The single colon notation is based on parsing the name and is available when passing the name of the fit method as a character string. In a string such as "head:tail" the "head" gives the name of the model fit function and the "tail" gives the argument to pass it's `method` parameter. This is only a convenience, as `method.args` can be also used. In some methods, i.e., splines, the default `formula = y ~ x` needs to be overridden by the user.

Table 2 lists the correspondence of pre-defined *method names* to model fit method functions. As mentioned above, these are only a subset of the model fit methods that are expected to work. When using these names there is no need for users to attach additional packages but the packages must be available (installed).

Table 2. Available predefined method names, the model fit functions they call, the packages where the functions reside, the class of the returned fitted model object and the arguments that can be passed to their `method` parameter using single colon notation.

Predefined method names	Model fit methods	R package	Object class
"lm", "lm:qr"	`lm()`	'stats'	"lm"
"rlm", "rlm:M", "rlm:MM"	`rlm()`	'MASS'	"rlm" ("lm")
"lts", "ltsReg"	`ltsReg()`	'robustbase'	"lts"

"ma", "sma", "sma:SMA", "sma:MA", "sma:OLS"	`sma()`	'smatr'	"ma" or "sma"
"gls", "gls:REML", "gls:ML"	`gls()`	'nlme'	"gls"
"rq", "rq:sfn", "rq:sfnc", "rq:lasso"	`rq()`	'quantreg'	"rq"
"rqss", "rqss:sfn", "rqss:sfnc", "rqss:lasso"	`rqss()`	'quantreg'	"rqss"
"SMA", "MA", "RMA", "OLS"	`lmodel2()`	'lmodel2'	("list")

Several model fit functions are supported explicitly (see tables), and some of their differences smoothed out. Compatibility is checked late, based on the class of the returned fitted model object. This makes it possible to use wrapper functions that do model selection or other adjustments to the fit procedure on a per panel or per group basis. Moreover, if the value returned as model fit object is `NULL` or `NA`, plotting is skipped on a per group within panel basis.

In the case of fitted model objects of classes not explicitly supported, an attempt is made to find the usual accessors and/or fitted object members, and if found, either complete or partial support is frequently achieved. In this case a message is issued encouraging users to check the validity of the values extracted as the structure of fitted model objects belonging to different classes and the values returned by their accessors can vary, potentially resulting in decoding errors leading to the return of wrong values for estimates.

The argument to parameter `method` can be either the name of a function object, possibly using double colon notation in case its package is not attached, or a character string matching the function name for functions in the search path. This approach makes it possible to support model fit functions that are not dependencies of 'ggpmisc'. Either by attaching the package where the function is defined and passing it by name or as string, or using double colon notation when passing the name of the function.

User-defined functions can be passed as argument to parameter `method` as long as they have parameters `formula`, `data` `subset` and possibly `weights`. Additional arguments can be passed to any method as a named list through parameter `method.args`. As in `stat_smooth()` prior `weights` are passed to the model fit functions' `weights` (plural!) parameter by mapping a numeric variable to plot aesthetic `weight` (singular!).

Tables 1 lists natively supported model fit functions, with the caveat that only some 'broom' methods' specializations have been actually tested with statistics from 'ggpmisc'. In addition, the statistics based on 'broom' methods require the user to tailor their behaviour by passing additional arguments in the call and occasionally some detective work to find out the names of variables in the returned data frame as these names are set by methods from 'broom'.

Table 1. Model fit methods supported by the different statistics available in package 'ggpmisc'. Column f indicates whether computations are done by group (G) or by plot panel (P).

Statistic	f	**Supported model fit methods**
`stat_poly_line()`	G	"lm", "rlm", "lts", "sma", "ma", "gls", "nls", "onls", *others with methods* `predict()` or `fitt`
`stat_poly_eq()`	G	"lm", "rlm", "lts", "sma", "ma", "gls", "nls", "onls", *others with needed accesors*
`stat_quant_line()`	G	"rq", "rqss"
`stat_quant_band()`	G	"rq", "rqss"
`stat_quant_eq()`	G	"rq", "rqss"
`stat_ma_line()`	G	"SMA", "MA", "RMA", "OLS"
`stat_ma_eq()`	G	"SMA", "MA", "RMA", "OLS"
`stat_fit_residuals()`	G	"lm", "rlm", "lts", "sma", "ma", "gls", "rq", "rqss", "nls", "onls", *others with method* `residu`

stat_fit_fitted()	G	"lm", "rlm", "lts", "gls", "rq", "rqss", "nls", "onls", *others with method* fitted()
stat_fit_deviations()	G	"lm", "rlm", "lts", "gls", "rq", "rqss", "nls", "onls", *others with methods* fitted() and weigh
stat_fit_augment()	G	*any with 'broom' method* augment()
stat_fit_glance()	G	*any with 'broom' method* glance()
stat_fit_tidy()	G	*any with 'broom' method* tidy()
stat_fit_tb()	P	*any with 'broom' method* tidy()

The single colon notation is based on parsing the name and is available when passing the name of the fit method as a character string. In a string such as "head:tail" the "head" gives the name of the model fit function and the "tail" gives the argument to pass it's method parameter. This is only a convenience, as method.args can be also used. In some methods, i.e., splines, the default formula = y ~ x needs to be overridden by the user.

Table 2 lists the correspondence of pre-defined *method names* to model fit method functions. As mentioned above, these are only a subset of the model fit methods that are expected to work. When using these names there is no need for users to attach additional packages but the packages must be available (installed).

Table 2. Available predefined method names, the model fit functions they call, the packages where the functions reside, the class of the returned fitted model object and the arguments that can be passed to their method parameter using single colon notation.

Predefined method names	Model fit methods	R package	Object class
"lm", "lm:qr"	lm()	'stats'	"lm"
"rlm", "rlm:M", "rlm:MM"	rlm()	'MASS'	"rlm" ("lm")
"lts", "ltsReg"	ltsReg()	'robustbase'	"lts"
"ma", "sma", "sma:SMA", "sma:MA", "sma:OLS"	sma()	'smatr'	"ma" or "sma"
"gls", "gls:REML", "gls:ML"	gls()	'nlme'	"gls"
"rq", "rq:sfn", "rq:sfnc", "rq:lasso"	rq()	'quantreg'	"rq"
"rqss", "rqss:sfn", "rqss:sfnc", "rqss:lasso"	rqss()	'quantreg'	"rqss"
"SMA", "MA", "RMA", "OLS"	lmodel2()	'lmodel2'	("list")

Which variables are available for mapping?

Computed variables and their names can vary depending on the method used to fit a model or the output.type in use. They can also depend for a given method on other arguments passed when fitting a model or extracting estimates and other computed values. In many cases, when values are not available, the variables are filled with NA values.

In the statistics returning formatted strings for use as annotations, a message is issued by default in interactive R sessions, listing the short names for available formatted labels as recognized by functions use_label() and f_use_label(), except when output.type = "numeric" is passed, in which case the names of all variables accessible by after_stat() within a call to aes() are listed. This default ("nicknames") can be changed by setting R option "ggpmisc.stat.vars.message" to one of "names", "colnames" or "none".

In the statistics that plot a prediction or more generally mainly return numeric variables, a message is issued by default in interactive R sessions, listing the names of all variables accessible by after_stat() within a call to aes() with at least some non-missing values. This default ("colnames") can be changed by setting R option "ggpmisc.stat.vars.message" to "none".

Aesthetics

`stat_ma_line()` understands the following aesthetics. Required aesthetics are displayed in bold and defaults are displayed for optional aesthetics:

- x
- y
- group → inferred

`stat_ma_eq()` understands the following aesthetics. Required aesthetics are displayed in bold and defaults are displayed for optional aesthetics:

- x
- y
- group → inferred
- grp.label
- hjust → "inward"
- label → after_stat(rr.label)
- npcx → after_stat(npcx)
- npcy → after_stat(npcy)
- vjust → "inward"

Learn more about setting these aesthetics in `vignette("ggplot2-specs")`.

See Also

The major axis regression model is fitted with function `lmodel2()`, please consult its documentation.

Please, see the articles in online-only documentation for additional use examples and guidance.

statistics from 'ggpmisc' for model fit annotations: `stat_poly_eq()`, `stat_quant_eq()`, `stat_ma_eq()` and `stat_distrmix_eq()`, and for model fit predictions: `stat_poly_line()`, `stat_quant_line()`, `stat_quant_band()`, `stat_ma_line()` and `stat_distrmix_line()`.

Examples

```
# generate artificial data
set.seed(98723)
my.data <- data.frame(x = rnorm(100) + (0:99) / 10 - 5,
                      y = rnorm(100) + (0:99) / 10 - 5,
                      group = c("A", "B"))

# using defaults (major axis regression)
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_ma_line() +
  stat_ma_eq()
```

```
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_ma_line() +
  stat_ma_eq(mapping = use_label("eq"))

ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_ma_line() +
  stat_ma_eq(mapping = use_label("eq"), decreasing = TRUE)

# use_label() can assemble and map a combined label
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_ma_line(method = "MA") +
  stat_ma_eq(mapping = use_label("eq", "R2", "P"))

ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_ma_line(method = "MA") +
  stat_ma_eq(mapping = use_label("R2", "P", "theta", "method"))

# using ranged major axis regression
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_ma_line(method = "RMA",
               range.y = "interval",
               range.x = "interval") +
  stat_ma_eq(mapping = use_label("eq", "R2", "P"),
             method = "RMA",
             range.y = "interval",
             range.x = "interval")

# No permutation-based test
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_ma_line(method = "MA") +
  stat_ma_eq(mapping = use_label("eq", "R2"),
             method = "MA",
             nperm = 0)

# explicit formula "x explained by y"
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_ma_line(formula = x ~ y) +
  stat_ma_eq(formula = x ~ y,
             mapping = use_label("eq", "R2", "P"))

# grouping
ggplot(my.data, aes(x, y, color = group)) +
  geom_point() +
  stat_ma_line() +
  stat_ma_eq()
```

```
# labelling equations
ggplot(my.data,
       aes(x, y,  shape = group, linetype = group, grp.label = group)) +
  geom_point() +
  stat_ma_line(color = "black") +
  stat_ma_eq(mapping = use_label("grp", "eq", "R2")) +
  theme_classic()
```

---

stat_multcomp *Labels for pairwise multiple comparisons*

Description

stat_multcomp fits a linear model by default with stats::lm() but alternatively using other model fit functions. The model is passed to function glht() from package 'multcomp' to fit Tukey, Dunnet or other **pairwise** contrasts and generates labels based on adjusted *P*-values.

Usage

```
stat_multcomp(
  mapping = NULL,
  data = NULL,
  geom = NULL,
  position = "identity",
  ...,
  orientation = NA,
  formula = y ~ factor(x),
  method = "lm",
  method.args = list(),
  contrasts = "Tukey",
  p.adjust.method = NULL,
  fit.seed = NA,
  fm.cutoff.p.value = 1,
  mc.cutoff.p.value = 1,
  mc.critical.p.value = 0.05,
  small.p = getOption("ggpmisc.small.p", default = FALSE),
  adj.method.tag = 4,
  p.digits = NULL,
  label.type = "bars",
  bjust = 0.5,
  label.y = NULL,
  vstep = NULL,
  output.type = NULL,
  na.rm = FALSE,
  parse = NULL,
  show.legend = FALSE,
  inherit.aes = TRUE
)
```

Arguments

mapping	The aesthetic mapping, usually constructed with aes(). Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset, only needed if you want to override the plot defaults.
geom	The geometric object to use display the data
position	The position adjustment to use for overlapping points on this layer.
...	other arguments passed on to layer. This can include aesthetics whose values you want to set, not map. See layer for more details.
orientation	character 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".
formula	a formula object. Using aesthetic names x and y instead of original variable names. The rhs must include a call to factor() even if the variable mapped to the x aesthetic is a factor! *In both flipped and not flipped plots,* x *should be in the rhs*.
method	function or character If character, "lm", "rlm", "lmrob", "lts", "gls", "ma", "sma", "segreg", "rq" or the name of a model fit function are accepted, possibly followed by the fit function's method argument separated by a colon (e.g. "rlm:M"). If a function is different to lm(), rlm(), ltsReg(), gls(), ma, sma, it must have formal parameters named formula, data, and weights. See Details.
method.args	named list with additional arguments. Not data or weights which are always passed through aesthetic mappings.
contrasts	character vector of length one or a numeric matrix. If character, one of "Tukey" or "Dunnet". If a matrix, one column per level of the factor mapped to x and one row per **pairwise** contrast.
p.adjust.method	character As the argument for parameter type of function adjusted() passed as argument to parameter test of summary.glht. Accepted values are "single-step", "Shaffer", "Westfall", "free", "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr", "none".
fit.seed	RNG seed argument passed to set.seed(). Defaults to NA, indicating that set.seed() should not be called.
fm.cutoff.p.value	numeric [0..1] The *P*-value for the main effect of factor x in the ANOVA test for the fitted model above which no pairwise comparisons are computed or labels generated. Be aware that recent literature tends to recommend to consider which testing approach is relevant to the problem at hand instead of requiring the significance of the main effect before applying multiple comparisons' tests. The default value is 1, imposing no restrictions.
mc.cutoff.p.value	numeric [0..1] The *P*-value for the individual contrasts above which no labelled bars are generated. Default is 1, labelling all pairwise contrasts tested.
mc.critical.p.value	numeric The critical *P*-value used for tests when encoded as letters.

small.p	logical Flag to switch use of lower case p for *p*-value.
adj.method.tag	numeric, character or function If numeric, the length in characters of the abbreviation of the method used to adjust *p*-values. A value of zero, adds no label and a negative value uses as starting point for the abbreviation the word "adjusted". If character its value is used as subscript. If a function, the value used is the value returned by the function when passed p.adjust.method as its only argument.
p.digits	integer Number of digits after the decimal point to use for *P*-value in labels. If Inf, use exponential notation with three decimal places.
label.type	character One of "bars", "letters" or "LETTERS", selects how the results of the multiple comparisons are displayed. Only "bars" can be used together with contrasts = "Dunnet".
bjust	numeric usually with range 0..1, the justification of the labels relative to the bars.
label.y	numeric with range 0..1 "normalized parent coordinates" (npc units) or character if using geom_text_npc() or geom_label_npc(). If using geom_text() or geom_label() numeric in native data units. *In flipped plots, it refers to* x *rather than* y *position*.
vstep	numeric in npc units, the vertical displacement step-size used between labels for different contrasts when label.type = "bars". *In flipped plots, it increases the* x *rather than* y *position*.
output.type	character One of "expression", "text", "markdown", "marquee", "latex", "latex.eqn", "latex.deqn" or "numeric".
na.rm	a logical indicating whether NA values should be stripped before the computation proceeds.
parse	logical Passed to the geom. If TRUE, the labels will be parsed into expressions and displayed as described in plotmath. Default is TRUE if output.type = "expression" and FALSE otherwise.
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.
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.

Details

This statistic can be used to automatically annotate a plot with *P*-values for **pairwise** multiple comparison tests, based on Tukey contrasts (all pairwise), Dunnet contrasts (other levels against the first one) or a subset of all possible pairwise contrasts. See Meier (2022, Chapter 3) for an accessible explanation of multiple comparisons and contrasts with package 'multcomp', of which stat_multcomp() is mostly a wrapper. It supports most model fit methods supported by function glht(). The requirement is that the necessary query functions for the class of the fitted model object are available. If not supported, an error will be triggered by glht(). The computations are done in two stages. First a model is fitted to all the data in a plot panel, followed by the pairwise contrasts. A threshold P-value can the set for the main effect of "treatments" as a condition to apply the pairwise tests.

One variable mapped to either *x* or *y* aesthetic must be a factor, and the other variable a continuous one. Mapping a factor creates the required grouping and determines the default orientation, i.e., similarly as in `stat_boxplot()`.

Two ways of displaying the outcomes are implemented, and are selected by ‘"bars"‘, ‘"letters"‘ or ‘"LETTERS"‘ as argument to parameter ‘label.type‘. ‘"letters"‘ and ‘"LETTERS"‘ can be used only with Tukey contrasts, as otherwise the encoding is ambiguous. As too many bars clutter a plot, the maximum number of factor levels supported for ‘"bars"‘ together with Tukey contrasts is five, while together with Dunnet contrasts or contrasts defined by a numeric matrix, no limit is imposed. `label.y` determines the location of the *letters* labels or the position of the first *pairwise label*. If `label.y` is numeric and has length > 1, values are taken as positions for successive bar labels, and recycled if the vector is too short, ignoring `vstep`.

`stat_multcomp()` by default generates character labels ready to be parsed as R expressions but LaTeX (use TikZ device), markdown (use package 'ggtext') and plain text are also supported, as well as numeric values for user-generated text labels. The value of `parse` is set automatically based on `output.type`, but if you assemble labels that need parsing from `numeric` output, the default needs to be overridden. This statistic only generates annotation labels and segments connecting the compared factor levels, or letter labels that discriminate significantly different groups.

Currently, contrasts that involve more than two levels of a factor, such as the average of two treatment levels against a control level are not supported. Similarly, flipping is not yet functional with bar labels because of the lack of a working geom.

Value

A data frame with one row per comparison for `label.type = "bars"`, or a data frame with one row per factor level for `label.type = "letters"` and `label.type = "LETTERS"`. Variables (= columns) as described under **Computed variables**.

Computed variables

Computed variables and their names vary depending on the `method` used to fit a model or the `output.type` in use. They can also depend for a given `method` on the `contrasts`.

If `output.type = "numeric"` and `label.type = "bars"` the returned tibble contains columns listed below. In all cases if the model fit function used does not return a value, the label is set to `character(0L)` and the numeric value to `NA`.

x,xmin,xmax x position, numeric.

y,ymin,ymax y position, numeric.

coefficients Delta estimate from pairwise contrasts, numeric.

contrasts Contrasts as two levels' ordinal "numbers" separated by a dash, character.

tstat *t*-statistic estimates for the pairwise contrasts, numeric.

p.value *P*-value for the pairwise contrasts.

p.signif logical for the pairwise contrasts significance.

fm.method Set according `method` used.

fm.class Most derived class of the fitted model object.

fm.formula Formula extracted from the fitted model object if available, or the formula argument.

fm.formula.chr Formula extracted from the fitted model object if available, or the formula argument, formatted as character.

mc.adjusted The method used to adjust the *P*-values.

mc.contrast The type of contrast used for multiple comparisons.

n The total number of observations or rows in data.

default.label text label, always included, but possibly NA.

If output.type is not `"numeric"` the returned data frame includes in addition the following labels:

stars.label *P*-value for the pairwise contrasts encoded as "starts", character.

p.value.label *P*-value for the pairwise contrasts, character.

delta.label The coefficient or estimate for the difference between compared pairs of levels.

t.value.label *t*-statistic estimates for the pairwise contrasts, character.

If `output.type = "numeric"` and `label.type = "letters"` or `label.type = "LETTERS"` the returned tibble contains columns listed below.

x,xmin,xmax x position, numeric.

y,ymin,ymax y position, numeric.

critical.p.value *P*-value used in pairwise tests, numeric.

fm.method Set according `method` used.

fm.class Most derived class of the fitted model object.

fm.formula Formula extracted from the fitted model object if available, or the formula argument.

fm.formula.chr Formula extracted from the fitted model object if available, or the formula argument, formatted as character.

mc.adjusted The method used to adjust the *P*-values.

mc.contrast The type of contrast used for multiple comparisons.

n The total number of observations or rows in data.

default.label text label, always included, but possibly NA.

If output.type is not `"numeric"` the returned data frame includes in addition the following labels:

letters.label Letters that distinguish levels based on significance from multiple comparisons test.

Alternatives

`stat_signif()` in package 'ggsignif' is an earlier and independent implementation of pairwise tests.

Output types

The formatting of character strings to be displayed in plots are marked as mathematical equations. Depending on the geom used, the mark-up needs to be encoded differently, or in some cases mark-up not applied.

`"expression"` The labels are encoded as character strings to be parsed into R's plotmath expressions.

`"LaTeX"`, `"TeX"`, `"tikz"`, `"latex"` The labels are encoded as 'LaTeX' maths equations, without the "fences" for switching in math mode.

`"latex.eqn"` Same as `"latex"` but enclosed in single `$`, i.e., as in-line maths.

`"latex.deqn"` Same as `"latex"` but enclosed in double `$$`, i.e., as display maths.

`"markdown"` The labels are encoded as character strings using markdown syntax, with some embedded HTML.

`"marquee"` The labels are encoded as character strings using markdown syntax, with 'marquee' supported spans.

`"text"` The labels are plain ASCII character strings.

`"numeric"` No labels are generated. This value is accepted by the statistics, but not by the label formatting functions.

`NULL` The value used depends on the argument passed to `geom`.

If `geom = "latex"` (package 'xdvir') the output type used is `"latex.eqn"`. If `geom = "richtext"` (package 'ggtext') or `geom = "textbox"` (package 'ggtext') the output type used is `"markdown"`. If `geom = "marquee"` (package 'marquee') the output type used is `"marquee"`. For all other values of `geom` the default is `"expression"`. Invalid values as argument trigger an error.

Which variables are available for mapping?

Computed variables and their names can vary depending on the `method` used to fit a model or the `output.type` in use. They can also depend for a given `method` on other arguments passed when fitting a model or extracting estimates and other computed values. In many cases, when values are not available, the variables are filled with `NA` values.

In the statistics returning formatted strings for use as annotations, a message is issued by default in interactive R sessions, listing the short names for available formatted labels as recognized by functions `use_label()` and `f_use_label()`, except when `output.type = "numeric"` is passed, in which case the names of all variables accessible by `after_stat()` within a call to `aes()` are listed. This default (`"nicknames"`) can be changed by setting R option `"ggpmisc.stat.vars.message"` to one of `"names"`, `"colnames"` or `"none"`.

In the statistics that plot a prediction or more generally mainly return numeric variables, a message is issued by default in interactive R sessions, listing the names of all variables accessible by `after_stat()` within a call to `aes()` with at least some non-missing values. This default (`"colnames"`) can be changed by setting R option `"ggpmisc.stat.vars.message"` to `"none"`.

Aesthetics

`stat_multcomp()` understands the following aesthetics. Required aesthetics are displayed in bold and defaults are displayed for optional aesthetics:

- x
- y
- group → inferred
- hjust → after_stat(just)
- label → after_stat(default.label)
- size → 2.5
- weight → 1
- xmax → after_stat(ifelse(flipped_aes, NA_real_, xmax))
- xmin → after_stat(ifelse(flipped_aes, NA_real_, xmin))
- ymax → after_stat(ifelse(flipped_aes, ymax, NA_real_))
- ymin → after_stat(ifelse(flipped_aes, ymin, NA_real_))

Learn more about setting these aesthetics in vignette("ggplot2-specs").

Note

R option OutDec is obeyed based on its value at the time the plot is rendered, i.e., displayed or printed. Set options(OutDec = ",") for languages like Spanish or French.

References

Meier, Lukas (2022) *ANOVA and Mixed Models: A Short Introduction Using R*. Chapter 3 Contrasts and Multiple Testing. The R Series. Boca Raton: Chapman and Hall/CRC. ISBN: 9780367704209, doi:10.1201/9781003146216.

See Also

This statistic uses the implementation of Tests of General Linear Hypotheses in function glht. See summary.glht and p.adjust for the supported tests and the references therein for the theory behind them.

Please, see the articles at online-only documentation for additional use examples and guidance.

Examples

```
p1 <- ggplot(mpg, aes(factor(cyl), hwy)) +
  geom_boxplot(width = 0.33)

## labeleld bars

p1 +
  stat_multcomp()

p1 +
  stat_multcomp(adj.method.tag = FALSE)

# test against a control, with first level being the control
# change order of factor levels in data to set the control group
p1 +
  stat_multcomp(contrasts = "Dunnet")
```

```
# arbitrary pairwise contrasts, in arbitrary order
p1 +
  stat_multcomp(contrasts = rbind(c(0, 0, -1, 1),
                                  c(0, -1, 1, 0),
                                  c(-1, 1, 0, 0)))

# different methods to adjust the contrasts
p1 +
  stat_multcomp(p.adjust.method = "holm")

p1 +
  stat_multcomp(p.adjust.method = "fdr")

# position of contrasts' bars (based on scale limits)
p1 +
  stat_multcomp(label.y = "bottom")

p1 +
  stat_multcomp(label.y = 11)

# use different labels: difference and P-value from hypothesis tests
p1 +
  stat_multcomp(use_label("Delta", "P"),
                size = 2.75)

# control smallest P-value displayed and number of digits
p1 +
  stat_multcomp(p.digits = 4)

# highlight significant differences
p1 +
  stat_multcomp(aes(alpha = after_stat(p.signif)),
                alpha.target = c("text", "segment", "box.line")) +
  scale_alpha_manual(values =  c(0.33, 1))

# label only significant differences
# but test and correct for all pairwise contrasts!
p1 +
  stat_multcomp(mc.cutoff.p.value = 0.01)

## letters as labels for test results

p1 +
  stat_multcomp(label.type = "letters")

# stricter critical p-value than default used for test
p1 +
  stat_multcomp(label.type = "letters",
                mc.critical.p.value = 0.01)

# justification of label in bar, and text anchor point
# for a compact display
```

```
p1 +
  stat_multcomp(geom = "text_pairwise",
                arrow = grid::arrow(ends = "both",
                                    length = unit(1.5, "mm")),
                bjust = 0,
                hjust = 1.05,
                vjust = 0.5,
                vstep = 0.05,
                label.y = 14,
                adj.method.tag = 0) +
  expand_limits(y = 0)

## Flipping is supported
p2 <- ggplot(mpg, aes(hwy, factor(cyl))) +
  geom_boxplot(width = 0.33)

p2 +
  stat_multcomp(label.type = "letters") +
  scale_y_discrete(expand = expansion(add = c(1.2, 0.5)))

p2 +
  stat_multcomp(label.type = "bars",
                geom = "text_pairwise",
                size = 3)
```

`stat_peaks` *Local maxima (peaks) or minima (valleys)*

Description

`stat_peaks()` tags or extracts rows in `data` containing local or global maxima of `y`. `stat_valleys` tags or extracts rows in `data` containing local or global minima of `y`. They make it easy to highlight and label peaks and valleys based on their `x` and/or `y` coordinates. Orientations flipping as well as dates and times are supported.

Usage

```
stat_peaks(
  mapping = NULL,
  data = NULL,
  geom = "point",
  position = "identity",
  ...,
  orientation = "x",
  span = 5,
  global.threshold = 0,
  local.threshold = 0,
```

```
  local.reference = "median",
  strict = FALSE,
  label.fmt = NULL,
  x.label.fmt = label.fmt,
  y.label.fmt = NULL,
  extract.peaks = NULL,
  na.rm = FALSE,
  show.legend = FALSE,
  inherit.aes = TRUE
)

stat_valleys(
  mapping = NULL,
  data = NULL,
  geom = "point",
  position = "identity",
  ...,
  orientation = "x",
  span = 5,
  global.threshold = 0.01,
  local.threshold = NULL,
  local.reference = "median",
  strict = FALSE,
  label.fmt = NULL,
  x.label.fmt = label.fmt,
  y.label.fmt = NULL,
  extract.valleys = NULL,
  na.rm = FALSE,
  show.legend = FALSE,
  inherit.aes = TRUE
)
```

Arguments

mapping	The aesthetic mapping, usually constructed with aes or aes_. Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
geom	The geometric object to use display the data
position	The position adjustment to use for overlapping points on this layer
...	other arguments passed on to layer. This can include aesthetics whose values you want to set, not map. See layer for more details.
orientation	character The orientation of the layer can be set to either "x", the default, or "y".
span	odd positive integer A peak is defined as an element in a sequence which is greater than all other elements within a moving window of width span centred at that element. The default value is 5, meaning that a peak is taller than its four nearest neighbours. span = NULL extends the span to the whole length of x.

`global.threshold`
: numeric A value belonging to class `"AsIs"` is interpreted as an absolute minimum height or depth expressed in data units. A bare `numeric` value (normally between 0.0 and 1.0), is interpreted as relative to `threshold.range`. In both cases it sets a *global* height (depth) threshold below which peaks (valleys) are ignored. A bare negative `numeric` value indicates the *global* height (depth) threshold below which peaks (valleys) are be ignored. If `global.threshold = NULL`, no threshold is applied and all peaks returned.

`local.threshold`
: numeric A value belonging to class `"AsIs"` is interpreted as an absolute minimum height (depth) expressed in data units relative to a within-window computed reference value. A bare `numeric` value (normally between 0.0 and 1.0), is interpreted as expressed in units relative to `threshold.range`. In both cases `local.threshold` sets a *local* height (depth) threshold below which peaks (valleys) are ignored. If `local.threshold = NULL` or if `span` spans the whole of `x`, no threshold is applied.

`local.reference`
: character One of `"median"`, `"median.log"`, `"median.sqrt"`, `"farthest"`, `"farthest.log"` or `"farthest.sqrt"`. The reference used to assess the height of the peak, is either the minimum/maximum value within the window or the median of all values in the window.

`strict`
: logical flag: if `TRUE`, an element must be strictly greater than all other values in its window to be considered a peak. Default: `FALSE` (since version 0.13.1).

`label.fmt`, `x.label.fmt`, `y.label.fmt`
: character strings giving a format definition for construction of character strings labels with function `sprintf` from `x` and/or `y` values.

`extract.peaks`, `extract.valleys`
: If `TRUE` only the rows containing peaks or valleys are returned. If `FALSE` the whole of `data` is returned but with labels set to `""` in rows not containing peaks or valleys. If `NULL`, the default, `TRUE`, is used unless the argument passed to `geom` is `"text_repel"`, `"label_repel"` or `"marquee_repel"`.

`na.rm`
: a logical value indicating whether NA values should be stripped before the computation proceeds.

`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.

`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`.

Details

As `find_valleys`, `stat_peaks` and `stat_valleys` call `find_peaks` to search for peaks or valleys, this description applies to all four functions.

Function `find_peaks` is a wrapper built onto function `peaks` from **splus2R**, adds support for peak height thresholds and handles `span = NULL` and non-finite (including NA) values differently than `splus2R::peaks`. Instead of giving an error when `na.rm = FALSE` and `x` contains `NA` values, `NA` values are replaced with the smallest finite value in `x`. `span = NULL` is treated as a special case and

selects max(x). Passing 'strict = TRUE' ensures that multiple global and within window maxima are ignored, and can result in no peaks being returned.#'

Two tests make it possible to ignore irrelevant peaks. One test (global.threshold) is based on the absolute height of the peaks and can be used in all cases to ignore globally low peaks. A second test (local.threshold) is available when the window defined by 'span' does not include all observations and can be used to ignore peaks that are not locally prominent. In this second approach the height of each peak is compared to a summary computed from other values within the window of width equal to span where it was found. In this second case, the reference value used within each window containing a peak is given by local.reference. Parameter threshold.range determines how the bare numeric values passed as argument to global.threshold and local.threshold are scaled. The default, NULL uses the range of x. Thresholds for ignoring too small peaks are applied after peaks are searched for, and threshold values can in some cases result in no peaks being found. If either threshold is not available (NA) the returned value is a NA vector of the same length as x.

The local.threshold argument is used *as is* when local.reference is "median" or "farthest", i.e., the same distance between peak and reference is used as cut-off irrespective of the value of the reference. In cases when the prominence of peaks is positively correlated with the baseline, a local.threshold that increases together with increasing computed within window median or farthest value applies apply a less stringent height requirement in regions with overall low height. In this case, natural logarithm or square root weighting can be requested with 'local.reference' arguments '"median.log"', '"farthest.log"', '"median.sqrt"', and '"farthest.sqrt"' as arguments for local.reference.

Value

A data frame with one row for each peak (or valley) found in the data extracted from the input data or all rows in data. Added columns contain the labels.

Computed and copied variables in the returned data frame

x x-value at the peak (or valley) as numeric.

y y-value at the peak (or valley) as numeric.

x.label x-value at the peak (or valley) formatted as character.

y.label y-value at the peak (or valley) formatted as character.

is.peak/is.valley logical vector, TRUE at peaks or valleys.

Label positioning and formatting

stat_peaks(), stat_valleys() and stat_spikes() work nicely together with geoms geom_text_repel(), geom_label_repel(), and geom_marquee_repel() from package ggrepel to solve the problem of overlapping labels by displacing them. If using geom_text(), discard overlapping labels using check_overlap = TRUE.

By default the labels are character values ready to be ploted as plain text, but with a suitable label.fmt argument, labels formatted as plotmath expressions, markdown or LaTeX can be created (e.g., containing Greek letters or super or subscripts, maths or colour) can be generated for use with geoms from packages 'marquee', 'ggtext' and 'xdvir'.

The default is geom = "point" it is likely to work well in almost any situation. The default aesthetics mappings set by these stats allow their direct use with geom_text(), geom_label(), geom_line(), geom_rug(), geom_hline() and geom_vline() by just passing an argument to geom.

Aesthetics

`stat_peaks()` understands the following aesthetics. Required aesthetics are displayed in bold and defaults are displayed for optional aesthetics:

- x
- y
- group → inferred
- label → `after_stat(x.label)`
- xintercept → `after_stat(x)`
- yintercept → `after_stat(y)`

`stat_valleys()` understands the following aesthetics. Required aesthetics are displayed in bold and defaults are displayed for optional aesthetics:

- x
- y
- group → inferred
- label → `after_stat(x.label)`
- xintercept → `after_stat(x)`
- yintercept → `after_stat(y)`

Learn more about setting these aesthetics in `vignette("ggplot2-specs")`.

See Also

`find_spikes()` for a function suitable for very narrow and steep peaks and valleys.

Examples

```
# lynx and Nile are time.series objects recognized by
# ggpp::ggplot.ts() and converted on-the-fly with a default mapping

# numeric, date times and dates are supported with data frames

# using defaults
ggplot(Nile) +
  geom_line() +
  stat_peaks(colour = "red") +
  stat_valleys(colour = "blue")

# using wider window
ggplot(Nile) +
  geom_line() +
  stat_peaks(colour = "red", span = 11) +
  stat_valleys(colour = "blue", span = 11)

# global threshold for peak height
ggplot(Nile) +
```

```
  geom_line() +
  stat_peaks(colour = "red",
             global.threshold = 0.5) # half of data range

ggplot(Nile) +
  geom_line() +
  stat_peaks(colour = "red",
             global.threshold = I(1100)) + # data unit
             expand_limits(y = c(0, 1500))

# local (within window) threshold for peak height
# narrow peaks at the tip and locally tall

ggplot(Nile) +
  geom_line() +
  stat_peaks(colour = "red",
             span = 9,
             local.threshold = 0.3,
             local.reference = "farthest")

# with narrower window
ggplot(Nile) +
  geom_line() +
  stat_peaks(colour = "red",
             span = 5,
             local.threshold = 0.25,
             local.reference = "farthest")

ggplot(lynx) +
  geom_line() +
  stat_peaks(colour = "red",
             local.threshold = 1/5,
             local.reference = "median")

ggplot(Nile) +
  geom_line() +
  stat_valleys(colour = "blue",
               global.threshold = I(700))

# orientation is supported
ggplot(lynx, aes(lynx, time)) +
  geom_line(orientation = "y") +
  stat_peaks(colour = "red", orientation = "y") +
  stat_valleys(colour = "blue", orientation = "y")

# default aesthetic mapping supports additional geoms
ggplot(lynx) +
  geom_line() +
  stat_peaks(colour = "red") +
  stat_peaks(colour = "red", geom = "rug")

ggplot(lynx) +
  geom_line() +
```

```
  stat_peaks(colour = "red") +
  stat_peaks(colour = "red", geom = "text", hjust = -0.1, angle = 33)

ggplot(lynx, aes(lynx, time)) +
  geom_line(orientation = "y") +
  stat_peaks(colour = "red", orientation = "y") +
  stat_peaks(colour = "red", orientation = "y",
             geom = "text", hjust = -0.1)

# Force conversion of time series time into POSIXct date time
ggplot(lynx, as.numeric = FALSE) +
  geom_line() +
  stat_peaks(colour = "red") +
  stat_peaks(colour = "red",
             geom = "text",
             hjust = -0.1,
             x.label.fmt = "%Y",
             angle = 33)

ggplot(Nile, as.numeric = FALSE) +
  geom_line() +
  stat_peaks(colour = "red") +
  stat_peaks(colour = "red",
             geom = "text_s",
             position = position_nudge_keep(x = 0, y = 60),
             hjust = -0.1,
             x.label.fmt = "%Y",
             angle = 90) +
  expand_limits(y = 2000)

ggplot(lynx, as.numeric = FALSE) +
  geom_line() +
  stat_peaks(colour = "red",
             geom = "text_s",
             position = position_nudge_to(y = 7600),
             arrow = arrow(length = grid::unit(1.5, "mm")),
             point.padding = 0.7,
             x.label.fmt = "%Y",
             angle = 90) +
  expand_limits(y = 9000)
```

---

stat_poly_eq *Fitted model prediction and annotations*

Description

Statistics stat_poly_line and stat_poly_eq() fit a model, by default with stats::lm(), but alternatively using other model fit functions. While stat_poly_line adds a prediction line and band, stat_poly_eq() adds textual labels to a plot.

Usage

```
stat_poly_eq(
  mapping = NULL,
  data = NULL,
  geom = "text_npc",
  position = "identity",
  ...,
  orientation = NA,
  formula = NULL,
  method = "lm",
  method.args = list(),
  n.min = 2L,
  fit.seed = NA,
  eq.with.lhs = TRUE,
  eq.x.rhs = NULL,
  small.r = getOption("ggpmisc.small.r", default = FALSE),
  small.p = getOption("ggpmisc.small.p", default = FALSE),
  CI.brackets = c("[", "]"),
  rsquared.conf.level = 0.95,
  coef.digits = 3,
  coef.keep.zeros = TRUE,
  decreasing = getOption("ggpmisc.decreasing.poly.eq", FALSE),
  rr.digits = 2,
  f.digits = 3,
  p.digits = 3,
  label.x = "left",
  label.y = "top",
  hstep = 0,
  vstep = NULL,
  output.type = NULL,
  na.rm = FALSE,
  parse = NULL,
  show.legend = FALSE,
  inherit.aes = TRUE
)

stat_poly_line(
  mapping = NULL,
  data = NULL,
  geom = "smooth",
  position = "identity",
  ...,
  orientation = NA,
  method = "lm",
  formula = NULL,
  se = NULL,
  fit.seed = NA,
  fm.values = FALSE,
```

```
  n = 80,
  fullrange = FALSE,
  limit.to = NULL,
  level = 0.95,
  method.args = list(),
  n.min = 2L,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	The aesthetic mapping, usually constructed with aes(). Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset, only needed if you want to override the plot defaults.
geom	The geometric object to use display the data
position	The position adjustment to use for overlapping points on this layer.
...	other arguments passed on to layer. This can include aesthetics whose values you want to set, not map. See layer for more details.
orientation	character Either "x" or "y" controlling the default for formula. The letter indicates the aesthetic considered the explanatory variable in the model fit.
formula	a formula object. Using aesthetic names x and y instead of original variable names.
method	function or character If character, "lm", "rlm", "lmrob", "lts", "gls", "ma", "sma", "segreg", "rq" or the name of a model fit function are accepted, possibly followed by the fit function's method argument separated by a colon (e.g. "rlm:M"). If a function is different to lm(), rlm(), ltsReg(), gls(), ma, sma, it must have formal parameters named formula, data, and weights. See Details.
method.args	named list with additional arguments. Not data or weights which are always passed through aesthetic mappings.
n.min	integer Minimum number of distinct values in the explanatory variable (on the rhs of formula) for fitting to the attempted.
fit.seed	RNG seed argument passed to set.seed(). Defaults to NA, indicating that set.seed() should not be called.
eq.with.lhs	If character the string is pasted to the front of the equation label before parsing or a logical (see note).
eq.x.rhs	character this string will be used as replacement for "x" in the model equation when generating the label before parsing it.
small.r, small.p	logical Flags to switch use of lower case r and p for coefficient of determination and p-value.
CI.brackets	character vector of length 2. The opening and closing brackets used for the CI label.

`rsquared.conf.level`
: numeric Confidence level for the returned confidence interval. Set to NA to skip CI computation.

`coef.digits`, `f.digits`
: integer Number of significant digits to use for the fitted coefficients and F-value.

`coef.keep.zeros`
: logical Keep or drop trailing zeros when formatting the fitted coefficients and F-value.

`decreasing`
: logical It specifies the order of the terms in the returned character string; in increasing (default) or decreasing powers.

`rr.digits`, `p.digits`
: integer Number of digits after the decimal point to use for R^2 and P-value in labels. If `Inf`, use exponential notation with three decimal places.

`label.x`, `label.y`
: `numeric` with range 0..1 "normalized parent coordinates" (npc units) or character if using `geom_text_npc()` or `geom_label_npc()`. If using `geom_text()` or `geom_label()` numeric in native data units. If too short they will be recycled.

`hstep`, `vstep`
: numeric in npc units, the horizontal and vertical step used between labels for different groups.

`output.type`
: character One of "expression", "text", "markdown", "marquee", "latex", "latex.eqn", "latex.deqn" or "numeric".

`na.rm`
: a logical indicating whether NA values should be stripped before the computation proceeds.

`parse`
: logical Passed to the geom. If `TRUE`, the labels will be parsed into expressions and displayed as described in `plotmath`. Default is `TRUE` if `output.type = "expression"` and `FALSE` otherwise.

`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.

`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`.

`se`
: Display confidence interval around smooth? ('TRUE' by default only for fits with `lm()` and `rlm()`, see 'level' to control.)

`fm.values`
: logical Add metadata and parameter estimates extracted from the fitted model object; `FALSE` by default.

`n`
: Number of points at which to predict with the fitted model.

`fullrange`
: logical Should the fit prediction span the full range of the plot, or just the range of the explanatory variable?

`limit.to`
: character or numeric If character one of `""`, `"x"`, `"y"` or `"xy"`. Should the fit prediction be constrained to the range of the variables mapped to `x` and/or `y` in each data group? If numeric, the *new data* values to use for the explanatory variable when computing the predicted line and confidence band. When set, `limit.to` silently overrides `fullrange`!

`level`
: Level of confidence interval to use (0.95 by default).

Details

Statistics `stat_poly_line()` and `stat_poly_eq()` fit a model consistently, but return different values. `stat_poly_line()` plots a prediction line and band, similarly to `stat_smooth()` but has different defaults and supports a different set of model fit functions. `stat_poly_eq()` adds textual labels for R^2, adjusted R^2, the fitted model equation, P, and other parameters from a fitted model to a plot.

Lack of methods or explicit support for extraction of individual parameters results in the affected estimates and corresponding labels being set to `NA`. Similarly, confidence bands for the prediction line are not plotted in some cases, while in the case of MA and SMA models, the band only displays the uncertainty of the slope rather than for both slope plus intercept. While strings for R^2, adjusted R^2, F, and P annotations are returned for all valid linear models and many other types of fitted models, an automatically constructed character string for the fitted model equation is returned only for polynomials (see below). However, when not generated automatically, the equation can still be assembled by the user within the call to `aes()`. A label for the confidence interval of R^2, based on values computed with function `ci_rsquared()` from package 'confintr' is returned when possible.

When possible, i.e., nearly always, the `formula` used to build the equation label is extracted from the returned fitted model object. Most fitted model objects follow the example of `lm()` and include the formula for the model that has been fitted. Thus, this model formula can safely differ from the argument passed to parameter `formula` in the call to `stat_poly_eq()`.

The stats are designed to support user-defined methods that implement any or all of `method` *selection, model* `formula` *selection, dynamically adjusted* `method.args` *and conditional skipping of labelling on a by group basis.*

The minimum number of observations with distinct values in the explanatory variable can be set through parameter `n.min`. The default `n.min = 2L` is the smallest suitable for method `"lm"` but too small for method `"rlm"` for which `n.min = 3L` is needed. Anyway, model fits with very few observations are of little interest and using larger values of `n.min` than the default is wise.

As some model fitting approaches depend on the RNG (pseudo-Random Number Generator), when `fit.seed` is not `NA` it is used as argument in a call to `set.seed()` immediately ahead of model fitting, i.e., once for each group of observations.

Singularity, convergence, etc., are handled by the model fit functions. With method `"lm"`, singularity results in terms being dropped with a message if more numerous than can be fitted with a singular (exact) fit. In this case and if the model results in a perfect fit due to low number of observation, estimates for various parameters are `NaN` or `NA`. With methods other than `"lm"`, the model fit functions simply fail in case of singularity, e.g., singular fits are not implemented in `"rlm"`.

Value

`stat_poly_eq()` returns a data frame, with a single row per group and columns as described below. `stat_poly_line()` returns a data frame, with `n` rows per group and columns as described below. In cases when the number of observations is less than `n.min` or when the model fit function returns a single `NA` or `NULL`, a data frame with no rows or columns (built by `data.frame()`) is returned, and silently rendered as an empty/invisible plot layer.

When a `predict()` method is not available for the fitted model class, the value returned by calling `fitted()`, if available, is replaces it and the returned data frame with as many rows as observations, instead of `n` rows, is returned with a message.

Model formula and model fitting

A ggplot statistic receives as `data` a data frame that is not the one passed as argument by the user, but instead a data frame with the variables mapped to aesthetics. In `stat_poly_eq()` the compute function is applied by group, each call "seeing" the subset of `data` for an individual group. As supported models are for regression lines, variables mapped to `x` and `y` should both be continuous, i.e., numeric or date time and model formulas defined using `x` and `y` as variable names.

The interpretation of the argument passed to `formula` is enhanced compared to `stat_smooth()`. Formulas with `x` as explanatory variable work as in `stat_smooth()` but formulas with `y` as explanatory variable are also accepted. `orientation` is set automatically based on which explanatory variable appears in the formula. Spline-based smoothers are only partially supported.

Model equation label

By default the equation label uses as symbols the names of the aesthetics, `x` and `y`. However, `"x"` and `"y"` can be substituted by providing a replacement character string for the right-hand-side and left-hand-side through `eq.x.rhs` and `eq.with.lhs`, respectively. For backward compatibility a logical is also accepted as argument for `eq.with.lhs`, with `FALSE` suppressing the left-hand-side.

If the model `formula` includes a transformation of the explanatory variable in its right-hand-side (rhs), a matching argument should be passed to parameter `eq.x.rhs` as its default value would result in an equation label that does not reflect the applied transformation. In most cases, a transformation should not be applied within the left hand side (lhs) of the model formula, but instead in the mapping of the response variable within `aes`. In this case it may be necessary to also pass a matching argument to parameter `eq.with.lhs`.

Parameter `orientation` is redundant as the orientation can be set by the `formula` but is included for consistency with `ggplot2::stat_smooth()`.

Position of labels

When data are grouped by mapping a factor to an aesthetic, e.g., `colour`, `shape` and/or `linetype` the model is fitted separately to each group, and for each group a whole set of labels is generated. If the argument passed to `label.y` is a vector of length 1, this value determines the position of the equation and/or other labels for the first group, and the positions of the labels for the remaining groups are generated by adding `vspace` based on the group number. If the argument passed to `label.y` is a vector of length > 1, it is used unchanged, possibly extended by recycling, ignoring `vstep`.

If the labels are rotated by 90 degrees then the automatic stepping is best based on `hstep` with `vstep = 0`. Similarly as described above, if `label.x` is a vector of length > 1, it is used unchanged, possibly extended by recycling, ignoring `hstep`.

When using facets and with a grouping that does not repeat in each panel, the automatic positioning in most cases will not be the desired one. Manual positioning using a vector of length > 1 for `label.x` and/or `label.y` is the currently available workaround.

Range of the prediction line

The range of the prediction line is controlled by parameters `fullrange` and `limit.to`. `fullrange` is backwards compatible both with earlier versions of 'ggpmisc' and with `stat_smooth()` from 'ggplot2'; an argument passed to `limit.to` overrides `fullrange` making it possible to constrain

the range to that of x, y, or both simultaneously, with "x", "y", or "xy", respectively, as argument. limit.to also accepts a numeric vector of values to be used as newdata when computing the prediction. Limiting the range based on both aesthetics is the best approach for major axis regression (MA, SMA, RMA) but can occasionally be useful also with some other methods when slopes are very steep and error variance in the explanatory variable is large. A numeric vector can be used to predict the response at specific values of the explanatory variable. If a single or very few values are predicted, it can be necessary to override the default geom = "smooth" with geom = "pointrange".

Model fit methods supported

Several model fit functions are supported explicitly (see tables), and some of their differences smoothed out. Compatibility is checked late, based on the class of the returned fitted model object. This makes it possible to use wrapper functions that do model selection or other adjustments to the fit procedure on a per panel or per group basis. Moreover, if the value returned as model fit object is NULL or NA, plotting is skipped on a per group within panel basis.

In the case of fitted model objects of classes not explicitly supported, an attempt is made to find the usual accessors and/or fitted object members, and if found, either complete or partial support is frequently achieved. In this case a message is issued encouraging users to check the validity of the values extracted as the structure of fitted model objects belonging to different classes and the values returned by their accessors can vary, potentially resulting in decoding errors leading to the return of wrong values for estimates.

The argument to parameter method can be either the name of a function object, possibly using double colon notation in case its package is not attached, or a character string matching the function name for functions in the search path. This approach makes it possible to support model fit functions that are not dependencies of 'ggpmisc'. Either by attaching the package where the function is defined and passing it by name or as string, or using double colon notation when passing the name of the function.

User-defined functions can be passed as argument to parameter method as long as they have parameters formula, data subset and possibly weights. Additional arguments can be passed to any method as a named list through parameter method.args. As in stat_smooth() prior weights are passed to the model fit functions' weights (plural!) parameter by mapping a numeric variable to plot aesthetic weight (singular!).

Tables 1 lists natively supported model fit functions, with the caveat that only some 'broom' methods' specializations have been actually tested with statistics from 'ggpmisc'. In addition, the statistics based on 'broom' methods require the user to tailor their behaviour by passing additional arguments in the call and occasionally some detective work to find out the names of variables in the returned data frame as these names are set by methods from 'broom'.

Table 1. Model fit methods supported by the different statistics available in package 'ggpmisc'. Column *f* indicates whether computations are done by group (G) or by plot panel (P).

Statistic	*f*	Supported model fit methods
stat_poly_line()	G	"lm", "rlm", "lts", "sma", "ma", "gls", "nls", "onls", *others with methods* predict() or fitt
stat_poly_eq()	G	"lm", "rlm", "lts", "sma", "ma", "gls", "nls", "onls", *others with needed accesors*
stat_quant_line()	G	"rq", "rqss"
stat_quant_band()	G	"rq", "rqss"
stat_quant_eq()	G	"rq", "rqss"
stat_ma_line()	G	"SMA", "MA", "RMA", "OLS"

stat_ma_eq()	G	"SMA", "MA", "RMA", "OLS"
stat_fit_residuals()	G	"lm", "rlm", "lts", "sma", "ma", "gls", "rq", "rqss", "nls", "onls", *others with method* residu
stat_fit_fitted()	G	"lm", "rlm", "lts", "gls", "rq", "rqss", "nls", "onls", *others with method* fitted()
stat_fit_deviations()	G	"lm", "rlm", "lts", "gls", "rq", "rqss", "nls", "onls", *others with methods* fitted() and weigh
stat_fit_augment()	G	*any with 'broom' method* augment()
stat_fit_glance()	G	*any with 'broom' method* glance()
stat_fit_tidy()	G	*any with 'broom' method* tidy()
stat_fit_tb()	P	*any with 'broom' method* tidy()

The single colon notation is based on parsing the name and is available when passing the name of the fit method as a character string. In a string such as "head:tail" the "head" gives the name of the model fit function and the "tail" gives the argument to pass it's `method` parameter. This is only a convenience, as `method.args` can be also used. In some methods, i.e., splines, the default `formula = y ~ x` needs to be overridden by the user.

Table 2 lists the correspondence of pre-defined *method names* to model fit method functions. As mentioned above, these are only a subset of the model fit methods that are expected to work. When using these names there is no need for users to attach additional packages but the packages must be available (installed).

Table 2. Available predefined method names, the model fit functions they call, the packages where the functions reside, the class of the returned fitted model object and the arguments that can be passed to their `method` parameter using single colon notation.

Predefined method names	Model fit methods	R package	Object class
"lm", "lm:qr"	lm()	'stats'	"lm"
"rlm", "rlm:M", "rlm:MM"	rlm()	'MASS'	"rlm" ("lm")
"lts", "ltsReg"	ltsReg()	'robustbase'	"lts"
"ma", "sma", "sma:SMA", "sma:MA", "sma:OLS"	sma()	'smatr'	"ma" or "sma"
"gls", "gls:REML", "gls:ML"	gls()	'nlme'	"gls"
"rq", "rq:sfn", "rq:sfnc", "rq:lasso"	rq()	'quantreg'	"rq"
"rqss", "rqss:sfn", "rqss:sfnc", "rqss:lasso"	rqss()	'quantreg'	"rqss"
"SMA", "MA", "RMA", "OLS"	lmodel2()	'lmodel2'	("list")

Which variables are available for mapping?

Computed variables and their names can vary depending on the `method` used to fit a model or the `output.type` in use. They can also depend for a given `method` on other arguments passed when fitting a model or extracting estimates and other computed values. In many cases, when values are not available, the variables are filled with `NA` values.

In the statistics returning formatted strings for use as annotations, a message is issued by default in interactive R sessions, listing the short names for available formatted labels as recognized by functions use_label() and f_use_label(), except when `output.type = "numeric"` is passed, in which case the names of all variables accessible by `after_stat()` within a call to `aes()` are listed. This default (`"nicknames"`) can be changed by setting R option `"ggpmisc.stat.vars.message"` to one of `"names"`, `"colnames"` or `"none"`.

In the statistics that plot a prediction or more generally mainly return numeric variables, a message is issued by default in interactive R sessions, listing the names of all variables accessible

by `after_stat()` within a call to `aes()` with at least some non-missing values. This default (`"colnames"`) can be changed by setting R option `"ggpmisc.stat.vars.message"` to `"none"`.

Variables returned by `stat_poly_line()`

y or x predicted value

ymin or xmin lower confidence limit around the fitted line

ymax or xmax upper confidence limit around the fitted line

se standard error

If `fm.values = TRUE` is passed then columns based on the summary of the model fit are added, with the same value in each row within a group. This is wasteful and disabled by default, but provides a simple and robust approach to achieve effects like colouring or hiding of the model fit line based on P, R^2, R^2_{adj} or the number of observations in a fit.

Variables returned by `stat_poly_eq()`

For all `output.type` arguments the following values are returned.

x,npcx x position

y,npcy y position

coefs fitted coefficients, named numeric vector as a list member

r.squared, rr.confint.level, rr.confint.low, rr.confint.high, adj.r.squared, f.value, f.df1, f.df2, p.value, AIC, BIC, n, knot numeric values, from the model fit object

grp.label Set according to mapping in `aes`.

knots list containing a numeric vector of knot or "psi" x-value for linear splines

fm.method name of method used, character

fm.class most derived class or the fitted model object, character

fm.formula.chr formatted model formula, character

If `output.type` is not `"numeric"` the returned tibble contains in addition to those above the columns listed below, each containing a single character string. The markup used depends on the value of `output.type`.

eq.label equation for the fitted polynomial as a character string to be parsed or `NA`

rr.label R^2 of the fitted model as a character string to be parsed

adj.rr.label Adjusted R^2 of the fitted model as a character string to be parsed

rr.confint.label Confidence interval for R^2 of the fitted model as a character string to be parsed

f.value.label F value and degrees of freedom for the fitted model as a whole.

p.value.label P-value for the F-value above.

AIC.label AIC for the fitted model.

BIC.label BIC for the fitted model.

n.label Number of observations used in the fit.

knots.label The knots or change points in segmented regression.

grp.label Set according to mapping in `aes`.

method.label Set according `method` used.

If output.type is `"numeric"` the returned tibble contains columns listed below in addition to the base ones. If the model fit function used does not return a value, the variable is set to `NA_real_`.

coef.ls list containing the "coefficients" matrix from the summary of the fit object

b_0.constant TRUE is polynomial is forced through the origin

b_i One or more columns with the coefficient estimates

To explore the computed values returned for a given input we suggest the use of `geom_debug()` as shown in the last examples below.

Output types

The formatting of character strings to be displayed in plots are marked as mathematical equations. Depending on the geom used, the mark-up needs to be encoded differently, or in some cases mark-up not applied.

`"expression"` The labels are encoded as character strings to be parsed into R's plotmath expressions.

`"LaTeX"`, `"TeX"`, `"tikz"`, `"latex"` The labels are encoded as 'LaTeX' maths equations, without the "fences" for switching in math mode.

`"latex.eqn"` Same as `"latex"` but enclosed in single `$`, i.e., as in-line maths.

`"latex.deqn"` Same as `"latex"` but enclosed in double `$$`, i.e., as display maths.

`"markdown"` The labels are encoded as character strings using markdown syntax, with some embedded HTML.

`"marquee"` The labels are encoded as character strings using markdown syntax, with 'marquee' supported spans.

`"text"` The labels are plain ASCII character strings.

`"numeric"` No labels are generated. This value is accepted by the statistics, but not by the label formatting functions.

`NULL` The value used depends on the argument passed to `geom`.

If `geom = "latex"` (package 'xdvir') the output type used is `"latex.eqn"`. If `geom = "richtext"` (package 'ggtext') or `geom = "textbox"` (package 'ggtext') the output type used is `"markdown"`. If `geom = "marquee"` (package 'marquee') the output type used is `"marquee"`. For all other values of `geom` the default is `"expression"`. Invalid values as argument trigger an error.

Aesthetics

`stat_poly_eq()` understands the following aesthetics. Required aesthetics are displayed in bold and defaults are displayed for optional aesthetics:

- `x`
- `y`
- `group` → inferred

- grp.label
- hjust → "inward"
- label → after_stat(rr.label)
- npcx → after_stat(npcx)
- npcy → after_stat(npcy)
- vjust → "inward"
- weight → 1

stat_poly_line() understands the following aesthetics. Required aesthetics are displayed in bold and defaults are displayed for optional aesthetics:

- x
- y
- group → inferred

Learn more about setting these aesthetics in vignette("ggplot2-specs").

References

Originally written as an answer to question 7549694 at Stackoverflow but enhanced based on suggestions from several users and my own needs.

See Also

Consult the documentation of the model fit functions used for the details and additional arguments that can be passed to them by name through parameter method.args.

Please, see the articles at online-only documentation for additional use examples and guidance.

statistics from 'ggpmisc' for model fit annotations: stat_poly_eq(), stat_quant_eq(), stat_ma_eq() and stat_distrmix_eq(), and for model fit predictions: stat_poly_line(), stat_quant_line(), stat_quant_band(), stat_ma_line() and stat_distrmix_line().

Examples

```
# generate artificial data
set.seed(4321)
x <- 1:100
y <- (x + x^2 + x^3) + rnorm(length(x), mean = 0, sd = mean(x^3) / 4)
y <- y / max(y)
my.data <- data.frame(x = x, y = y,
                      group = c("A", "B"),
                      y2 = y * c(1, 2) + c(0, 0.1),
                      w = sqrt(x))

# give a name to a formula
formula <- y ~ poly(x, 3, raw = TRUE)

# using defaults
```

```
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_poly_line() +
  stat_poly_eq()

# no weights
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_poly_line(formula = formula) +
  stat_poly_eq(formula = formula)

# other labels
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_poly_line(formula = formula) +
  stat_poly_eq(use_label("eq"), formula = formula)

# other labels
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_poly_line(formula = formula) +
  stat_poly_eq(use_label("eq"), formula = formula, decreasing = TRUE)

ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_poly_line(formula = formula) +
  stat_poly_eq(use_label("eq", "R2"), formula = formula)

ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_poly_line(formula = formula) +
  stat_poly_eq(use_label("R2", "R2.CI", "P", "method"), formula = formula)

ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_poly_line(formula = formula) +
  stat_poly_eq(use_label("R2", "F", "P", "n", sep = "*\"; \"*"),
               formula = formula)

# grouping
ggplot(my.data, aes(x, y2, color = group)) +
  geom_point() +
  stat_poly_line(formula = formula) +
  stat_poly_eq(formula = formula)

# rotation
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_poly_line(formula = formula) +
  stat_poly_eq(formula = formula, angle = 90)

# label location
ggplot(my.data, aes(x, y)) +
```

```
  geom_point() +
  stat_poly_line(formula = formula) +
  stat_poly_eq(formula = formula, label.y = "bottom", label.x = "right")

ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_poly_line(formula = formula) +
  stat_poly_eq(formula = formula, label.y = 0.1, label.x = 0.9)

# modifying the explanatory variable within the model formula
# modifying the response variable within aes()
# eq.x.rhs and eq.with.lhs defaults must be overridden!!
formula.trans <- y ~ I(x^2)
ggplot(my.data, aes(x, y + 1)) +
  geom_point() +
  stat_poly_line(formula = formula.trans) +
  stat_poly_eq(use_label("eq"),
               formula = formula.trans,
               eq.x.rhs = "~x^2",
               eq.with.lhs = "y + 1~~`=`~~")

# using weights
ggplot(my.data, aes(x, y, weight = w)) +
  geom_point() +
  stat_poly_line(formula = formula) +
  stat_poly_eq(formula = formula)

# no weights, 4 digits for R square
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_poly_line(formula = formula) +
  stat_poly_eq(formula = formula, rr.digits = 4)

# manually assemble and map a specific label using paste() and aes()
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_poly_line(formula = formula) +
  stat_poly_eq(aes(label =
                 after_stat(paste(rr.label, n.label, sep = "*\", \"*"))),
               formula = formula)

# manually assemble and map a specific label using sprintf() and aes()
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_poly_line(formula = formula) +
  stat_poly_eq(aes(label =
                 after_stat(
                   sprintf("%s*\" with \"*%s*\" and \"*%s",
                           rr.label, f.value.label, p.value.label))),
               formula = formula)

# x on y regression
ggplot(my.data, aes(x, y)) +
```

```
  geom_point() +
  stat_poly_line(formula = formula, orientation = "y") +
  stat_poly_eq(use_label("eq", "adj.R2"),
               formula = x ~ poly(y, 3, raw = TRUE))

# conditional user specified label
ggplot(my.data, aes(x, y2, color = group)) +
  geom_point() +
  stat_poly_line(formula = formula) +
  stat_poly_eq(aes(label =
                     after_stat(ifelse(adj.r.squared > 0.96,
                                       paste(adj.rr.label, eq.label,
                                             sep = "*\", \"*"),
                                       adj.rr.label))),
               rr.digits = 3,
               formula = formula)

# geom = "text"
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_poly_line(formula = formula) +
  stat_poly_eq(geom = "text", label.x = 100, label.y = 0, hjust = 1,
               formula = formula)
```

---

stat_quant_band *Quantile regression predictions and annotations*

Description

Statistics `stat_quant_line()`, `stat_quant_band()` and `stat_quant_eq()` fit models by quantile regression. While `stat_quant_line()` and `stat_quant_band()` add prediction lines and bands, `stat_quant_eq()` adds textual labels to a plot.

Usage

```
stat_quant_band(
  mapping = NULL,
  data = NULL,
  geom = "smooth",
  position = "identity",
  ...,
  orientation = NA,
  quantiles = c(0.25, 0.5, 0.75),
  formula = NULL,
  fit.seed = NA,
  fm.values = FALSE,
  n = 80,
  fullrange = FALSE,
```

```
  limit.to = NULL,
  method = "rq",
  method.args = list(),
  n.min = 3L,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

stat_quant_eq(
  mapping = NULL,
  data = NULL,
  geom = "text_npc",
  position = "identity",
  ...,
  orientation = NA,
  formula = NULL,
  quantiles = c(0.25, 0.5, 0.75),
  method = "rq:br",
  method.args = list(),
  n.min = 10L,
  fit.seed = NA,
  eq.with.lhs = TRUE,
  eq.x.rhs = NULL,
  coef.digits = 3,
  coef.keep.zeros = TRUE,
  decreasing = getOption("ggpmisc.decreasing.poly.eq", FALSE),
  rho.digits = 4,
  label.x = "left",
  label.y = "top",
  hstep = 0,
  vstep = NULL,
  output.type = NULL,
  na.rm = FALSE,
  parse = NULL,
  show.legend = FALSE,
  inherit.aes = TRUE
)

stat_quant_line(
  mapping = NULL,
  data = NULL,
  geom = "smooth",
  position = "identity",
  ...,
  orientation = NA,
  quantiles = c(0.25, 0.5, 0.75),
  formula = NULL,
```

```
  se = length(quantiles) == 1L,
  fit.seed = NA,
  fm.values = FALSE,
  n = 80,
  fullrange = FALSE,
  limit.to = NULL,
  method = "rq",
  method.args = list(),
  n.min = 3L,
  level = 0.95,
  type = "direct",
  interval = "confidence",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	The aesthetic mapping, usually constructed with aes(). Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset, only needed if you want to override the plot defaults.
geom	The geometric object to use display the data
position	The position adjustment to use for overlapping points on this layer.
...	other arguments passed on to layer. This can include aesthetics whose values you want to set, not map. See layer for more details.
orientation	character Either "x" or "y" controlling the default for formula. The letter indicates the aesthetic considered the explanatory variable in the model fit.
quantiles	numeric vector Values in 0..1 indicating the quantiles.
formula	a formula object. Using aesthetic names x and y instead of original variable names.
fit.seed	RNG seed argument passed to set.seed(). Defaults to NA, indicating that set.seed() should not be called.
fm.values	logical Add metadata and parameter estimates extracted from the fitted model object; FALSE by default.
n	Number of points at which to predict with the fitted model.
fullrange	logical Should the fit prediction span the full range of the plot, or just the range of the explanatory variable?
limit.to	character or numeric If character one of "", "x", "y" or "xy". Should the fit prediction be constrained to the range of the variables mapped to x and/or y in each data group? If numeric, the *new data* values to use for the explanatory variable when computing the predicted line and confidence band. When set, limit.to silently overrides fullrange!

method	function or character If character, "rq", "rqss" or the name of a model fit function are accepted, possibly followed by the fit function's method argument separated by a colon (e.g. "rq:br"). If a function different to rq(), it must accept arguments named formula, data, weights, tau and method and return a model fit object of class rq, rqs or rqss.
method.args	named list with additional arguments passed to rq(), rqss() or to another function passed as argument to method.
n.min	integer Minimum number of distinct values in the explanatory variable (on the rhs of formula) for fitting to the attempted.
na.rm	a logical indicating whether NA values should be stripped before the computation proceeds.
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.
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.
eq.with.lhs	If character the string is pasted to the front of the equation label before parsing or a logical (see note).
eq.x.rhs	character this string will be used as replacement for "x" in the model equation when generating the label before parsing it.
coef.digits, rho.digits	integer Number of significant digits to use for the fitted coefficients and rho in labels.
coef.keep.zeros	logical Keep or drop trailing zeros when formatting the fitted coefficients and F-value.
decreasing	logical It specifies the order of the terms in the returned character string; in increasing (default) or decreasing powers.
label.x, label.y	numeric with range 0..1 "normalized parent coordinates" (npc units) or character if using geom_text_npc() or geom_label_npc(). If using geom_text() or geom_label() numeric in native data units. If too short they will be recycled.
hstep, vstep	numeric in npc units, the horizontal and vertical step used between labels for different groups.
output.type	character One of "expression", "text", "markdown", "marquee", "latex", "latex.eqn", "latex.deqn" or "numeric".
parse	logical Passed to the geom. If TRUE, the labels will be parsed into expressions and displayed as described in plotmath. Default is TRUE if output.type = "expression" and FALSE otherwise.
se	logical Passed to quantreg::predict.rq().
level	numeric in range [0..1] Passed to quantreg::predict.rq().
type	character Passed to quantreg::predict.rq().
interval	character Passed to quantreg::predict.rq().

Details

While stat_poly_line() and stat_poly_eq() fit a single model per plot layer, stat_quant_line(), stat_quant_band() and stat_quant_eq() can fit multiple models sharing the same method and formula but differing in their probability. These probabilities are passed a vector argument to parameter quantiles.

stat_quant_line fits one or more quantile regressions and obtains predictions similarly to stat_quantile() from 'ggplot2', but in addition it computes confidence regions for the prediction lines. By default each quantile is plotted as a line, with a confidence band when se = TRUE.

stat_quant_band() fits quantile regressions and obtains predictions identically to stat_quant_line(). stat_quant_band() fits 2 or 3 quantiles in the same plot layer and displays the area between the predicted regression lines for the extreme quantiles as a band.

stat_quant_eq() fits quantile regressions and generates a set of labels for each regression line fitted. By default the labels are formatted as R's plotmath expressions, LaTeX and markdown are also supported.

stat_quant_eq(), stat_quant_line() and stat_quant_band() support both "rq" and "rqss" as method. In the case of "rqss" the model formula makes normally use of qss() to formulate the spline and its constraints. User defined functions are supported as method as long as they accept arguments named formula, data, weights, tau and method and return a model fit object of class rq, rqs or rqss. Such user-defined functions can implement model selection and/or method selection, or conditionally skip model fitting on a per data group basis.

The minimum number of observations with distinct values in the explanatory variable can be set through parameter n.min. The default n.min = 10L is a bare minimum for quantile regression. Model fits with such a small number of observations are of little interest and using larger values of n.min than the default is wise.

There are interesting uses for *double quantile regression*, i.e., a pair of quantile regressions on x and y on the same data. For example, when two variables are subject to mutual constrains, it is useful to consider both of them as explanatory and interpret the relationship based on them considered as limiting. 'ggpmisc' (>= 0.4.1) supports orientation making it easy implement the approach described by Cardoso (2019) under the name of "Double quantile regression".

Value

stat_quant_eq() returns a data frame, with one row per quantile and columns as described below, while stat_quant_line() and stat_quant_band() return a data frame, with n rows per quantile and columns as described below. If the number of observations is less than n.min or if the model fit method returns NA or NULL, a data frame with no rows or columns is returned, resulting in an empty/invisible plot layer.

Variables returned by stat_quant_eq()

If output.type is "numeric" the returned tibble contains columns in addition to a modified version of the original group:

x,npcx x position

y,npcy y position

coef.ls list containing the "coefficients" matrix from the summary of the fit object

rho, AIC, n numeric values extracted or computed from fit object

rq.method character, method used.

hjust, vjust Set to "inward" to override the default of the "text" geom.

quantile Indicating the quantile used for the fit

quantile.f Factor with a level for each quantile

b_0.constant TRUE is polynomial is forced through the origin

b_i One or columns with the coefficient estimates

If output.type different from `"numeric"` the returned tibble contains columns below in addition to a modified version of the original `group`:

x,npcx x position

y,npcy y position

eq.label equation for the fitted polynomial as a character string to be parsed

r.label, and one of cor.label, rho.label, or tau.label rho of the fitted model as a character string to be parsed

AIC.label AIC for the fitted model.

n.label Number of observations used in the fit.

method.label Set according `method` used.

rq.method character, method used.

rho, n numeric values extracted or computed from fit object.

hjust, vjust Set to "inward" to override the default of the "text" geom.

quantile Numeric value of the quantile used for the fit

quantile.f Factor with a level for each quantile

To explore the computed values returned for a given input we suggest the use of `geom_debug` as shown in the example below.

Variables returned by `stat_quant_line()`

y or x predicted value

ymin or xmin lower confidence limit around the fitted line

ymax or xmax upper confidence limit around the fitted line

If `fm.values = TRUE` is passed then one column with the number of observations `n` used for each fit is also included, with the same value in each row within a group. This is wasteful and disabled by default, but provides a simple and robust approach to achieve effects like colouring or hiding of the model fit line based on the number of observations.

Variables returned by `stat_quant_band()`

y or x Regression prediction for the middle quantile, if three quantiles are passed as argument

ymin or xmin Regression prediction for the smallest quantile

ymax or xmax Regression prediction for the largest quantile

If `fm.values = TRUE` is passed then one column with the number of observations `n` used for each fit is also included, with the same value in each row within a group. This is wasteful and disabled by default, but provides a simple and robust approach to achieve effects like colouring or hiding of the model fit line based on the number of observations.

Output types

The formatting of character strings to be displayed in plots are marked as mathematical equations. Depending on the geom used, the mark-up needs to be encoded differently, or in some cases mark-up not applied.

`"expression"` The labels are encoded as character strings to be parsed into R's plotmath expressions.

`"LaTeX"`, `"TeX"`, `"tikz"`, `"latex"` The labels are encoded as 'LaTeX' maths equations, without the "fences" for switching in math mode.

`"latex.eqn"` Same as `"latex"` but enclosed in single `$`, i.e., as in-line maths.

`"latex.deqn"` Same as `"latex"` but enclosed in double `$$`, i.e., as display maths.

`"markdown"` The labels are encoded as character strings using markdown syntax, with some embedded HTML.

`"marquee"` The labels are encoded as character strings using markdown syntax, with 'marquee' supported spans.

`"text"` The labels are plain ASCII character strings.

`"numeric"` No labels are generated. This value is accepted by the statistics, but not by the label formatting functions.

`NULL` The value used depends on the argument passed to `geom`.

If `geom = "latex"` (package 'xdvir') the output type used is `"latex.eqn"`. If `geom = "richtext"` (package 'ggtext') or `geom = "textbox"` (package 'ggtext') the output type used is `"markdown"`. If `geom = "marquee"` (package 'marquee') the output type used is `"marquee"`. For all other values of `geom` the default is `"expression"`. Invalid values as argument trigger an error.

Model equation label

By default the equation label uses as symbols the names of the aesthetics, `x` and `y`. However, `"x"` and `"y"` can be substituted by providing a replacement character string for the right-hand-side and left-hand-side through `eq.x.rhs` and `eq.with.lhs`, respectively. For backward compatibility a logical is also accepted as argument for `eq.with.lhs`, with `FALSE` suppressing the left-hand-side.

If the model `formula` includes a transformation of the explanatory variable in its right-hand-side (rhs), a matching argument should be passed to parameter `eq.x.rhs` as its default value would result in an equation label that does not reflect the applied transformation. In most cases, a transformation should not be applied within the left hand side (lhs) of the model formula, but instead in the mapping

of the response variable within aes. In this case it may be necessary to also pass a matching argument to parameter eq.with.lhs.

Parameter orientation is redundant as the orientation can be set by the formula but is included for consistency with ggplot2::stat_smooth().

Position of labels

When data are grouped by mapping a factor to an aesthetic, e.g., colour, shape and/or linetype the model is fitted separately to each group, and for each group a whole set of labels is generated. If the argument passed to label.y is a vector of length 1, this value determines the position of the equation and/or other labels for the first group, and the positions of the labels for the remaining groups are generated by adding vspace based on the group number. If the argument passed to label.y is a vector of length > 1, it is used unchanged, possibly extended by recycling, ignoring vstep.

If the labels are rotated by 90 degrees then the automatic stepping is best based on hstep with vstep = 0. Similarly as described above, if label.x is a vector of length > 1, it is used unchanged, possibly extended by recycling, ignoring hstep.

When using facets and with a grouping that does not repeat in each panel, the automatic positioning in most cases will not be the desired one. Manual positioning using a vector of length > 1 for label.x and/or label.y is the currently available workaround.

Model formula and model fitting

A ggplot statistic receives as data a data frame that is not the one passed as argument by the user, but instead a data frame with the variables mapped to aesthetics. In stat_poly_eq() the compute function is applied by group, each call "seeing" the subset of data for an individual group. As supported models are for regression lines, variables mapped to x and y should both be continuous, i.e., numeric or date time and model formulas defined using x and y as variable names.

The interpretation of the argument passed to formula is enhanced compared to stat_smooth(). Formulas with x as explanatory variable work as in stat_smooth() but formulas with y as explanatory variable are also accepted. orientation is set automatically based on which explanatory variable appears in the formula. Spline-based smoothers are only partially supported.

Range of the prediction line

The range of the prediction line is controlled by parameters fullrange and limit.to. fullrange is backwards compatible both with earlier versions of 'ggpmisc' and with stat_smooth() from 'ggplot2'; an argument passed to limit.to overrides fullrange making it possible to constrain the range to that of x, y, or both simultaneously, with "x", "y", or "xy", respectively, as argument. limit.to also accepts a numeric vector of values to be used as newdata when computing the prediction. Limiting the range based on both aesthetics is the best approach for major axis regression (MA, SMA, RMA) but can occasionally be useful also with some other methods when slopes are very steep and error variance in the explanatory variable is large. A numeric vector can be used to predict the response at specific values of the explanatory variable. If a single or very few values are predicted, it can be necessary to override the default geom = "smooth" with geom = "pointrange".

Model fit methods supported

Several model fit functions are supported explicitly (see tables), and some of their differences smoothed out. Compatibility is checked late, based on the class of the returned fitted model object. This makes it possible to use wrapper functions that do model selection or other adjustments to the fit procedure on a per panel or per group basis. Moreover, if the value returned as model fit object is `NULL` or `NA`, plotting is skipped on a per group within panel basis.

In the case of fitted model objects of classes not explicitly supported, an attempt is made to find the usual accessors and/or fitted object members, and if found, either complete or partial support is frequently achieved. In this case a message is issued encouraging users to check the validity of the values extracted as the structure of fitted model objects belonging to different classes and the values returned by their accessors can vary, potentially resulting in decoding errors leading to the return of wrong values for estimates.

The argument to parameter `method` can be either the name of a function object, possibly using double colon notation in case its package is not attached, or a character string matching the function name for functions in the search path. This approach makes it possible to support model fit functions that are not dependencies of 'ggpmisc'. Either by attaching the package where the function is defined and passing it by name or as string, or using double colon notation when passing the name of the function.

User-defined functions can be passed as argument to parameter `method` as long as they have parameters `formula`, `data` `subset` and possibly `weights`. Additional arguments can be passed to any method as a named list through parameter `method.args`. As in `stat_smooth()` prior `weights` are passed to the model fit functions' `weights` (plural!) parameter by mapping a numeric variable to plot aesthetic `weight` (singular!).

Tables 1 lists natively supported model fit functions, with the caveat that only some 'broom' methods' specializations have been actually tested with statistics from 'ggpmisc'. In addition, the statistics based on 'broom' methods require the user to tailor their behaviour by passing additional arguments in the call and occasionally some detective work to find out the names of variables in the returned data frame as these names are set by methods from 'broom'.

Table 1. Model fit methods supported by the different statistics available in package 'ggpmisc'. Column *f* indicates whether computations are done by group (G) or by plot panel (P).

Statistic	*f*	Supported model fit methods
`stat_poly_line()`	G	"lm", "rlm", "lts", "sma", "ma", "gls", "nls", "onls", *others with methods* `predict()` or `fitt`
`stat_poly_eq()`	G	"lm", "rlm", "lts", "sma", "ma", "gls", "nls", "onls", *others with needed accesors*
`stat_quant_line()`	G	"rq", "rqss"
`stat_quant_band()`	G	"rq", "rqss"
`stat_quant_eq()`	G	"rq", "rqss"
`stat_ma_line()`	G	"SMA", "MA", "RMA", "OLS"
`stat_ma_eq()`	G	"SMA", "MA", "RMA", "OLS"
`stat_fit_residuals()`	G	"lm", "rlm", "lts", "sma", "ma", "gls", "rq", "rqss", "nls", "onls", *others with method* `residu`
`stat_fit_fitted()`	G	"lm", "rlm", "lts", "gls", "rq", "rqss", "nls", "onls", *others with method* `fitted()`
`stat_fit_deviations()`	G	"lm", "rlm", "lts", "gls", "rq", "rqss", "nls", "onls", *others with methods* `fitted()` and `weigh`
`stat_fit_augment()`	G	*any with 'broom' method* `augment()`
`stat_fit_glance()`	G	*any with 'broom' method* `glance()`
`stat_fit_tidy()`	G	*any with 'broom' method* `tidy()`
`stat_fit_tb()`	P	*any with 'broom' method* `tidy()`

The single colon notation is based on parsing the name and is available when passing the name of the fit method as a character string. In a string such as "head:tail" the "head" gives the name of the model fit function and the "tail" gives the argument to pass it's `method` parameter. This is only a convenience, as `method.args` can be also used. In some methods, i.e., splines, the default `formula = y ~ x` needs to be overridden by the user.

Table 2 lists the correspondence of pre-defined *method names* to model fit method functions. As mentioned above, these are only a subset of the model fit methods that are expected to work. When using these names there is no need for users to attach additional packages but the packages must be available (installed).

Table 2. Available predefined method names, the model fit functions they call, the packages where the functions reside, the class of the returned fitted model object and the arguments that can be passed to their `method` parameter using single colon notation.

Predefined method names	Model fit methods	R package	Object class
"lm", "lm:qr"	`lm()`	'stats'	"lm"
"rlm", "rlm:M", "rlm:MM"	`rlm()`	'MASS'	"rlm" ("lm")
"lts", "ltsReg"	`ltsReg()`	'robustbase'	"lts"
"ma", "sma", "sma:SMA", "sma:MA", "sma:OLS"	`sma()`	'smatr'	"ma" or "sma"
"gls", "gls:REML", "gls:ML"	`gls()`	'nlme'	"gls"
"rq", "rq:sfn", "rq:sfnc", "rq:lasso"	`rq()`	'quantreg'	"rq"
"rqss", "rqss:sfn", "rqss:sfnc", "rqss:lasso"	`rqss()`	'quantreg'	"rqss"
"SMA", "MA", "RMA", "OLS"	`lmodel2()`	'lmodel2'	("list")

Which variables are available for mapping?

Computed variables and their names can vary depending on the `method` used to fit a model or the `output.type` in use. They can also depend for a given `method` on other arguments passed when fitting a model or extracting estimates and other computed values. In many cases, when values are not available, the variables are filled with `NA` values.

In the statistics returning formatted strings for use as annotations, a message is issued by default in interactive R sessions, listing the short names for available formatted labels as recognized by functions `use_label()` and `f_use_label()`, except when `output.type = "numeric"` is passed, in which case the names of all variables accessible by `after_stat()` within a call to `aes()` are listed. This default (`"nicknames"`) can be changed by setting R option `"ggpmisc.stat.vars.message"` to one of `"names"`, `"colnames"` or `"none"`.

In the statistics that plot a prediction or more generally mainly return numeric variables, a message is issued by default in interactive R sessions, listing the names of all variables accessible by `after_stat()` within a call to `aes()` with at least some non-missing values. This default (`"colnames"`) can be changed by setting R option `"ggpmisc.stat.vars.message"` to `"none"`.

Aesthetics

`stat_quant_eq()` understands the following aesthetics. Required aesthetics are displayed in bold and defaults are displayed for optional aesthetics:

- x
- y
- group → inferred
- grp.label
- hjust → "inward"
- label → after_stat(eq.label)
- npcx → after_stat(npcx)
- npcy → after_stat(npcy)
- vjust → "inward"

stat_quant_line() understands the following aesthetics. Required aesthetics are displayed in bold and defaults are displayed for optional aesthetics:

- x
- y
- group → after_stat(group)
- weight → 1

stat_quant_band() understands the following aesthetics. Required aesthetics are displayed in bold and defaults are displayed for optional aesthetics:

- x
- y
- group → inferred

Learn more about setting these aesthetics in vignette("ggplot2-specs").

References

Cardoso, G. C. (2019) Double quantile regression accurately assesses distance to boundary trade-off. Methods in ecology and evolution, 10(8), 1322-1331.

See Also

rq, rqss and qss.

Please, see the articles at online-only documentation for additional use examples and guidance.

statistics from 'ggpmisc' for model fit annotations: stat_poly_eq(), stat_quant_eq(), stat_ma_eq() and stat_distrmix_eq(), and for model fit predictions: stat_poly_line(), stat_quant_line(), stat_quant_band(), stat_ma_line() and stat_distrmix_line().

Examples

```
# generate artificial data
set.seed(4321)
x <- 1:100
```

```
y <- (x + x^2 + x^3) + rnorm(length(x), mean = 0, sd = mean(x^3) / 4)
y <- y / max(y)
my.data <- data.frame(x = x, y = y,
                      group = c("A", "B"),
                      y2 = y * c(1, 2) + max(y) * c(0, 0.1),
                      w = sqrt(x))

# Predictions as lines
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_quant_line()

ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_quant_line(quantiles = 0.5, se = TRUE)

# Predictions as band
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_quant_band()

# y as explanatory variable (orientation = y)
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_quant_band(formula = x ~ y)

# Using splines
library(quantreg)

ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_quant_line(method = "rqss",
                  formula = y ~ qss(x, constraint = "D"),
                  quantiles = 0.5, se = FALSE)

# Adding annotations
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_quant_line() +
  stat_quant_eq()

ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_quant_line() +
  stat_quant_eq(mapping = use_label("eq"))

ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_quant_line() +
  stat_quant_eq(mapping = use_label("eq"), decreasing = TRUE)

ggplot(my.data, aes(x, y)) +
  geom_point() +
```

```
  stat_quant_line() +
  stat_quant_eq(mapping = use_label("eq", "method"))

# same formula as default
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_quant_line(formula = y ~ x) +
  stat_quant_eq(formula = y ~ x)

# explicit formula "x explained by y"
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_quant_line(formula = x ~ y) +
  stat_quant_eq(formula = x ~ y)

# using color
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_quant_line(mapping = aes(color = after_stat(quantile.f))) +
  stat_quant_eq(mapping = aes(color = after_stat(quantile.f))) +
  labs(color = "Quantiles")

# location and colour
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_quant_line(mapping = aes(color = after_stat(quantile.f))) +
  stat_quant_eq(mapping = aes(color = after_stat(quantile.f)),
                label.y = "bottom", label.x = "right") +
  labs(color = "Quantiles")

# give a name to a formula
formula <- y ~ poly(x, 3, raw = TRUE)

ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_quant_line(formula = formula, linewidth = 0.5) +
  stat_quant_eq(formula = formula)

# angle
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_quant_line(formula = formula, linewidth = 0.5) +
  stat_quant_eq(formula = formula, angle = 90, hstep = 0.04, vstep = 0,
                label.y = 0.02, hjust = 0, size = 3) +
  expand_limits(x = -15) # make space for equations

# user set quantiles
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_quant_line(formula = formula, quantiles = 0.5) +
  stat_quant_eq(formula = formula, quantiles = 0.5)

ggplot(my.data, aes(x, y)) +
```

```
  geom_point() +
  stat_quant_band(formula = formula,
                  quantiles = c(0.1, 0.5, 0.9)) +
  stat_quant_eq(formula = formula, parse = TRUE,
                quantiles = c(0.1, 0.5, 0.9))

# grouping
ggplot(my.data, aes(x, y2, color = group)) +
  geom_point() +
  stat_quant_line(formula = formula, linewidth = 0.5) +
  stat_quant_eq(formula = formula)

ggplot(my.data, aes(x, y2, color = group)) +
  geom_point() +
  stat_quant_band(formula = formula, linewidth = 0.75) +
  stat_quant_eq(formula = formula) +
  theme_bw()

# labelling equations
ggplot(my.data, aes(x, y2,  shape = group, linetype = group,
       grp.label = group)) +
  geom_point() +
  stat_quant_band(formula = formula, color = "black", linewidth = 0.75) +
  stat_quant_eq(mapping = use_label("grp", "eq", sep = "*\": \"*"),
                formula = formula) +
  expand_limits(y = 3) +
  theme_classic()

# modifying the explanatory variable within the model formula
# modifying the response variable within aes()
formula.trans <- y ~ I(x^2)
ggplot(my.data, aes(x, y + 1)) +
  geom_point() +
  stat_quant_line(formula = formula.trans) +
  stat_quant_eq(mapping = use_label("eq"),
                formula = formula.trans,
                eq.x.rhs = "~x^2",
                eq.with.lhs = "y + 1~~`=`~~")

# using weights
ggplot(my.data, aes(x, y, weight = w)) +
  geom_point() +
  stat_quant_line(formula = formula, linewidth = 0.5) +
  stat_quant_eq(formula = formula)

# no weights, quantile set to upper boundary
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_quant_line(formula = formula, quantiles = 0.95) +
  stat_quant_eq(formula = formula, quantiles = 0.95)

# manually assemble and map a specific label using paste() and aes()
ggplot(my.data, aes(x, y2, color = group, grp.label = group)) +
```

```
  geom_point() +
  stat_quant_line(method = "rq", formula = formula,
                  quantiles = c(0.05, 0.5, 0.95),
                  linewidth = 0.5) +
  stat_quant_eq(mapping = aes(label =
                              after_stat(
                                paste(grp.label, "*\": \"*", eq.label, sep = ""))),
                quantiles = c(0.05, 0.5, 0.95),
                formula = formula, size = 3)

# manually assemble and map a specific label using sprintf() and aes()
ggplot(my.data, aes(x, y2, color = group, grp.label = group)) +
  geom_point() +
  stat_quant_band(method = "rq", formula = formula,
                  quantiles = c(0.05, 0.5, 0.95)) +
  stat_quant_eq(mapping = aes(label =
                              after_stat(
                                sprintf("%s*\": \"*%s", grp.label, eq.label))),
                quantiles = c(0.05, 0.5, 0.95),
                formula = formula, size = 3)

# geom = "text"
ggplot(my.data, aes(x, y)) +
  geom_point() +
  stat_quant_line(formula = formula, quantiles = 0.5) +
  stat_quant_eq(label.x = "left", label.y = "top",
                formula = formula,
                quantiles = 0.5)
```

`stat_spikes` *Local narrow maxima or minima (spikes)*

Description

`stat_spikes()` tags or extracts rows in `data` containing local `y` narrow maxima and/or minima with very steep shoulders. It makes it possible to highlight and label spikes based on their `x` and/or `y` coordinates. Orientations flipping as well as dates and times are supported.

Usage

```
stat_spikes(
  mapping = NULL,
  data = NULL,
  geom = "point",
  position = "identity",
  ...,
  orientation = "x",
  height.threshold = 20,
```

```
  z.threshold = 7,
  k = 20,
  spike.direction = "both",
  label.fmt = NULL,
  x.label.fmt = label.fmt,
  y.label.fmt = NULL,
  extract.spikes = NULL,
  na.rm = FALSE,
  show.legend = FALSE,
  inherit.aes = TRUE
)
```

Arguments

mapping	The aesthetic mapping, usually constructed with aes or aes_. Only needs to be set at the layer level if you are overriding the plot defaults.
data	A layer specific dataset - only needed if you want to override the plot defaults.
geom	The geometric object to use display the data
position	The position adjustment to use for overlapping points on this layer
...	other arguments passed on to layer. This can include aesthetics whose values you want to set, not map. See layer for more details.
orientation	character The orientation of the layer can be set to either "x", the default, or "y".
height.threshold	numeric The minimum height of spikes expressed relative to the median amplitude of the baseline local variation of x.
z.threshold	numeric Modified local Z values larger than z.threshold are detected as boundaries of spikes.
k	integer width of median window used for smoothing; must be odd
spike.direction	character One of "up", "down", "both" or "skip", indicating which spikes are to be returned, if any.
label.fmt, x.label.fmt, y.label.fmt	character strings giving a format definition for construction of character strings labels with function sprintf from x and/or y values.
extract.spikes	If TRUE only the rows containing spikes are returned. If FALSE the whole of data is returned but with labels set to "" in rows not containing spikes. If NULL, the default, TRUE, is used unless the argument passed to geom is "text_repel", "label_repel" or "marquee_repel".
na.rm	a logical value indicating whether NA values should be stripped before the computation proceeds.
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.
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.

Details

Spikes are detected based on a modified Z score calculated from the differenced spectrum. The Z threshold used should be adjusted to the characteristics of the input and desired sensitivity. The lower the threshold the more stringent the test becomes, with shorter spikes being detected.

The algorithm uses running differences to detect abrupt changes in value, compared to an estimate of the baseline variation of the differences, approximating a baseline Z from MAD and a baseline value from the median differences. Currently, a single estimate of MAD is used but running medians, when posisble, as baseline. This comparison detects running differences that are unusually large, in most cases signalling a transition between values near the baseline and far from it, in both directions.

Transitions into- and out of spikes are distinguished based on the median of the non-differenced values, as a descriptor of the data baseline. As for the median of the differences, a running median is used when possible.

This function thus detects the start and end of each spike, and distinguishes upward and downward spikes.

`k` is the width in number of observations of the window used for running median smoothing to extract the baseline. A value several times the width of the broader spike but narrow enough to track broader peaks needs to be manually set in most cases.

With `na.rm = TRUE`, `NA` values are omitted before searching for spikes and set to `0L` in the returned vector.

If all spikes are guaranteed to be one observation-wide and either going up or down from the baseline, it is possible to detect them based purely on the `z.threshold` by passing `height.threshold = NA` and either `spike.direction = "up"` or `spike.direction = "down"`, which ensures very fast computation.

Value

A data frame with one row for each spike found in the data extracted from the input `data` or all rows in data. Added columns contain the labels.

Computed and copied variables in the returned data frame

x x-values at the spikes as numeric.

y y-values at the spikes as numeric.

x.label x-values at the spikes formatted as character.

y.label y-values at the spikes formatted as character.

is.spike integer vector of `0`, `1` or `-1`.

Label positioning and formatting

`stat_peaks()`, `stat_valleys()` and `stat_spikes()` work nicely together with geoms `geom_text_repel()`, `geom_label_repel()`, and `geom_marquee_repel()` from package ggrepel to solve the problem of overlapping labels by displacing them. If using `geom_text()`, discard overlapping labels using `check_overlap = TRUE`.

By default the labels are character values ready to be ploted as plain text, but with a suitable `label.fmt` argument, labels formatted as `plotmath` expressions, markdown or LaTeX can be created (e.g., containing Greek letters or super or subscripts, maths or colour) can be generated for use with geoms from packages 'marquee', 'ggtext' and 'xdvir'.

The default is `geom = "point"` it is likely to work well in almost any situation. The default aesthetics mappings set by these stats allow their direct use with `geom_text()`, `geom_label()`, `geom_line()`, `geom_rug()`, `geom_hline()` and `geom_vline()` by just passing an argument to `geom`.

Aesthetics

`stat_spikes()` understands the following aesthetics. Required aesthetics are displayed in bold and defaults are displayed for optional aesthetics:

- `x`
- `y`
- `group` → inferred
- `label` → `after_stat(x.label)`
- `xintercept` → `after_stat(x)`
- `yintercept` → `after_stat(y)`

Learn more about setting these aesthetics in `vignette("ggplot2-specs")`.

References

Whitaker, D. A.; Hayes, K. (2018) A simple algorithm for despiking Raman spectra. Chemometrics and Intelligent Laboratory Systems, 179, 82-84. doi:10.1016/j.chemolab.2018.06.009.

See Also

`find_peaks()` for functions suitable for broader peaks and valleys.

Examples

```
# lynx and Nile are time.series objects recognized by
# ggpp::ggplot.ts() and converted on-the-fly with a default mapping

n = 500
set.seed(45678)
my.data <- data.frame(x = 1:n,
                      y = rep(sin((0:19)/20 * 2 * pi), n / 20) +
                          stats::rnorm(n, sd = 0.5))
selector <- sample(seq_len(n), 5)
my.data$y[selector] <- my.data$y[selector] + 10

ggplot(my.data, aes(x, y)) +
  geom_line() +
  stat_spikes(colour = "orange")

ggplot(my.data, aes(x, -y)) +
```

```
  geom_line() +
  stat_spikes(colour = "orange")

ggplot(my.data, aes(x, y)) +
  geom_line() +
  stat_spikes(geom = "text", vjust = -0.5) +
  stat_spikes(geom = "rug", colour = "red")

ggplot(my.data, aes(x, y)) +
  geom_line() +
  stat_spikes(colour = "red", spike.direction = "up") +
  stat_spikes(colour = "blue", spike.direction = "down")

ggplot(my.data, aes(x, y)) +
  geom_line() +
  stat_spikes(colour = "red", spike.direction = "up")

ggplot(my.data, aes(x, y)) +
  geom_line() +
  stat_spikes(colour = "blue", spike.direction = "down")

ggplot(my.data, aes(x, y)) +
  geom_line() +
  stat_spikes(z.threshold = 2, colour = "orange")

ggplot(my.data, aes(x, y)) +
  geom_line() +
  stat_spikes(z.threshold = 20, colour = "orange")

ggplot(my.data, aes(x, y)) +
  geom_line() +
  stat_spikes(colour = "red",
              spike.direction = "up",
              height.threshold = NA)

# Inspecting the returned data using geom_debug_group()
# This provides a quick way of finding out the names of the variables that
# are available for mapping to aesthetics with after_stat().

gginnards.installed <- requireNamespace("gginnards", quietly = TRUE)

if (gginnards.installed)
  library(gginnards)

if (gginnards.installed)
ggplot(my.data, aes(x, y)) +
  geom_line() +
  stat_spikes(geom = "debug_group")

if (gginnards.installed)
ggplot(my.data, aes(x, y)) +
  geom_line() +
  stat_spikes(geom = "debug_group", extract.spikes = FALSE)
```

swap_xy *Swap x and y in a formula*

Description

By default a formula of x on y is converted into a formula of y on x, while the reverse swap is done only if backward = TRUE.

Usage

```
swap_xy(f, backwards = FALSE)
```

Arguments

f	formula An R model formula
backwards	logical If NULL the swap is done irrespective of the variable in the lhs.

Details

If backwards = TRUE, a formula with x in the lhs is always, returned. If backwards = FALSE, a formula with y in the lhs is always, returned. If backwards = NULL x and y are always swapped.

This function is meant to be used only as a helper within 'ggplot2' statistics. Normally together with geometries supporting orientation when we want to automate the change in orientation based on a user-supplied formula. Only x and y are exchanged, and in other respects the formula is rebuilt copying the environment from f.

Value

A copy of f with x and y swapped by each other in the lhs and rhs.

symmetric_limits *Expand a range to make it symmetric*

Description

Expand scale limits to make them symmetric around zero. Can be passed as argument to parameter limits of continuous scales from packages 'ggplot2' or 'scales'. Can be also used to obtain an enclosing symmetric range for numeric vectors.

Usage

```
symmetric_limits(x)
```

Arguments

x	numeric The automatic limits when used as argument to a scale's `limits` formal parameter. Otherwise a numeric vector, possibly a range, for which to compute a symmetric enclosing range.

Value

A numeric vector of length two with the new limits, which are always such that the absolute value of upper and lower limits is the same.

Examples

```
symmetric_limits(c(-1, 1.8))
symmetric_limits(c(-10, 1.8))
symmetric_limits(-5:20)
```

---

typeset_numbers *Typeset/format numbers preserving trailing zeros*

Description

Typeset/format numbers preserving trailing zeros

Usage

```
typeset_numbers(eq.char, output.type)
```

Arguments

eq.char	character A polynomial model equation as a character string.
output.type	character One of "expression", "latex", "tex", "text", "tikz", "markdown", "marquee".

Value

A `character` string.

Note

exponential number notation to typeset equivalent: Protecting trailing zeros in negative numbers is more involved than I would like. Not only we need to enclose numbers in quotations marks but we also need to replace dashes with the minus character. I am not sure we can do the replacement portably, but that recent R supports UTF gives some hope.

use_label *Assemble label and map it*

Description

Assemble model-fit-derived text or expressions and map them to the `label` aesthetic.

Usage

```
use_label(..., labels = NULL, other.mapping = NULL, sep = "*\", \"*")

f_use_label(..., labels = NULL, other.mapping = NULL, format = NULL)
```

Arguments

`...`	character Strings giving the names of at most six label components in the order they will be included in the combined label.
`labels`	character A vector with the name of at most six label components. If provided, values passed through `...` are ignored.
`other.mapping`	An unevaluated expression constructed with function `aes()` to be included in the returned value.
`sep`	character A string used as separator when pasting the label components together.
`format`	character A string with as many labels being mapped. **The format string must have mark-up matching the** `output.type` **used!** With the default geometries, `output.type = "expression"` requiring that the character string can

Details

Statistics `stat_poly_eq()`, `stat_ma_eq()`, `stat_quant_eq()`, `stat_distrmix_eq()` and `stat_correlation()` return multiple text strings to be used individually or assembled into longer character strings depending on the labels actually desired. Assembling and mapping them requires verbose R code and familiarity with the syntax of the markup language used, by default R *plotmath* expression syntax. Function `use_label()` automates these two tasks and accepts abbreviated familiar names for the parameters in addition to the name of the columns in the data object returned by the statistics. The default separator is suitable for `plotmath` expressions.

These four statistics return several `character` variables with names ending in `.label`. This ending can be omitted, as well as `.value` for `f.value.label`, `t.value.label`, `z.value.label`, `S.value.label` and `p.value.label`. `R2` can be used in place of `rr`. Furthermore, case is ignored. Thus, `use_label("eq", "R2")` is equivalent to `aes(label = paste(after_stat(eq.label), after_stat(rr.label), sep = "*\", \"*"))`.

Function `use_label()` calls `aes()` to create a mapping for the `label` aesthetic to a text string assembled by calling `paste()`. Function `f_use_label()` uses `sprintf()` instead of `paste()` and expects a format string with the correct number of place-holders for the strings. Both functions can, when needed, combine this mapping to the `label` aesthetic with other mappings created with `aes()`.

When the number of labels does not match the number of fields in the format, or the label names do not match the names of variables in `data`, errors are triggered when the mapping is constructed, i.e., at the time the plot is rendered.

Value

A mapping to the `label` aesthetic and optionally additional mappings as an unevaluated R expression, built using function `aes()`, ready to be passed as argument to the `mapping` parameter of the supported statistics.

See Also

Functions `use_label()` and `f_use_label()` are wrappers on function `aes()` and can be used to create mappings to the `label` aesthetic. These functions are matched to *statistics* for annotations from package 'ggpmisc'. Please, see their documentation for the formatted character labels they generate.

Please, see the articles at online-only documentation for additional use examples and guidance.

Examples

```
# generate artificial data
set.seed(4321)
x <- 1:100
y <- (x + x^2 + x^3) + rnorm(length(x), mean = 0, sd = mean(x^3) / 4)
my.data <- data.frame(x = x,
                      y = y * 1e-5,
                      group = c("A", "B"),
                      y2 = y * 1e-5 + c(2, 0))

# give a name to a formula
formula <- y ~ poly(x, 3, raw = TRUE)

# default label constructed by use_label()
ggplot(data = my.data,
       mapping = aes(x = x, y = y2, colour = group)) +
  geom_point() +
  stat_poly_line(formula = formula) +
  stat_poly_eq(mapping = use_label(),
               formula = formula)

# user specified label components and default for sep
ggplot(data = my.data,
       mapping = aes(x = x, y = y2, colour = group)) +
  geom_point() +
  stat_poly_line(formula = formula) +
  stat_poly_eq(mapping = use_label("eq", "F"),
               formula = formula)

# user specified label components and separator
ggplot(data = my.data,
       mapping = aes(x = x, y = y2, colour = group)) +
```

```
  geom_point() +
  stat_poly_line(formula = formula) +
  stat_poly_eq(mapping = use_label("R2", "F", sep = "*\" with \"*"),
               formula = formula)

# user specified label components and format string
ggplot(data = my.data,
       mapping = aes(x = x, y = y2, colour = group)) +
  geom_point() +
  stat_poly_line(formula = formula) +
  stat_poly_eq(mapping =
                 f_use_label("R2", "F",
                             format = "\"Estimates: \"*%s*\" with \"*%s*\" works!\""),
               formula = formula)

# combine the mapping to the label aesthetic with other mappings
ggplot(data = my.data,
       mapping = aes(x = x, y = y2)) +
  geom_point(mapping = aes(colour = group)) +
  stat_poly_line(mapping = aes(colour = group), formula = formula) +
  stat_poly_eq(mapping = use_label("grp", "eq", "F",
                                   aes(grp.label = group)),
               formula = formula)

# combine other mappings with default labels
ggplot(data = my.data,
       mapping = aes(x = x, y = y2)) +
  geom_point(mapping = aes(colour = group)) +
  stat_poly_line(mapping = aes(colour = group), formula = formula) +
  stat_poly_eq(mapping = use_label(aes(colour = group)),
               formula = formula)

# example with other available components
ggplot(data = my.data,
       mapping = aes(x = x, y = y2, colour = group)) +
  geom_point() +
  stat_poly_line(formula = formula) +
  stat_poly_eq(mapping = use_label("eq", "adj.R2", "n"),
               formula = formula)

# multiple labels
ggplot(data = my.data,
       mapping = aes(x, y2, colour = group)) +
  geom_point() +
  stat_poly_line(formula = formula) +
  stat_poly_eq(mapping = use_label("R2", "F", "P", "AIC", "BIC"),
               formula = formula) +
  stat_poly_eq(mapping = use_label(c("eq", "n")),
               formula = formula,
               label.y = "bottom",
               label.x = "right")

# quantile regression
```

```
ggplot(data = my.data,
       mapping = aes(x, y)) +
  stat_quant_band(formula = formula) +
  stat_quant_eq(mapping = use_label("eq", "n"),
                formula = formula) +
  geom_point()

# major axis regression
ggplot(data = my.data, aes(x = x, y = y)) +
  stat_ma_line() +
  stat_ma_eq(mapping = use_label("eq", "n")) +
  geom_point()

# correlation
ggplot(data = my.data,
       mapping = aes(x = x, y = y)) +
  stat_correlation(mapping = use_label("r", "t", "p")) +
  geom_point()
```

`xy_outcomes2factor` *Convert two numeric ternary outcomes into a factor*

Description

Convert two numeric ternary outcomes into a factor

Usage

```
xy_outcomes2factor(x, y)

xy_thresholds2factor(x, y, x_threshold = 0, y_threshold = 0)
```

Arguments

`x, y`	numeric vectors of -1, 0, and +1 values, indicating down regulation, uncertain response or up-regulation, or numeric vectors that can be converted into such values using a pair of thresholds.
`x_threshold, y_threshold`	numeric vector Ranges enclosing the values to be considered uncertain for each of the two vectors..

Details

This function converts the numerically encoded values into a factor with the four levels `"xy"`, `"x"`, `"y"` and `"none"`. The factor created can be used for faceting or can be mapped to aesthetics.

Note

This is an utility function that only saves some typing. The same result can be achieved by a direct call to `factor`. This function aims at making it easier to draw quadrant plots with facets based on the combined outcomes.

See Also

Other scales for omics data: `FC_format()`, `outcome2factor()`, `scale_colour_logFC()`, `scale_colour_outcome()`, `scale_shape_outcome()`, `scale_x_logFC()`, `scale_y_Pvalue()`

Examples

```
xy_outcomes2factor(c(-1, 0, 0, 1, -1), c(0, 1, 0, 1, -1))
xy_thresholds2factor(c(-1, 0, 0, 1, -1), c(0, 1, 0, 1, -1))
xy_thresholds2factor(c(-1, 0, 0, 0.1, -5), c(0, 2, 0, 1, -1))
```

Index

∗ **'broom'-based statistics for model fits**
  stat_fit_augment, 57
  stat_fit_glance, 75
  stat_fit_tb, 82
  stat_fit_tidy, 91
∗ **peaks and valleys functions**
  find_peaks, 12
  find_spikes, 16
∗ **scales for omics data**
  outcome2factor, 19
  scale_colour_logFC, 28
  scale_colour_outcome, 31
  scale_shape_outcome, 34
  scale_x_logFC, 35
  scale_y_Pvalue, 38
  xy_outcomes2factor, 161
∗ **statistics for correlation-test annotation**
  stat_correlation, 41
∗ **statistics for display of model fit residuals**
  stat_fit_deviations, 64
∗ **statistics for multiple comparison annotations**
  stat_multcomp, 110

adj_rr_label *(*plain_label*)*, 20
aes, *15*, *42*, *51*, *58*, *65*, *76*, *83*, *92*, *99*, *111*, *119*, *126*, *128*, *139*, *152*, *158*, *159*
aes_, *119*, *152*
as.factor, *15*
augment, *59*, *61*, *68*, *79*, *86*, *95*, *105*, *107*, *131*, *145*

bold_label *(*plain_label*)*, 20
borders, *43*, *52*, *59*, *66*, *76*, *84*, *93*, *100*, *112*, *120*, *127*, *140*, *152*
broom, *62*, *80*, *87*, *96*

check_output_type, 5
check_poly_formula, 7
ci_rsquared, *128*
coef.lmodel2, 9
coefs2poly_eq, 10
confint.lmodel2, 11
cor.test, *46*

dev.capabilities, *15*

f_use_label, *46*, *55*, *107*, *115*, *131*, *146*
f_use_label *(*use_label*)*, 158
f_value_label *(*plain_label*)*, 20
factor, *20*, *162*
FC_format(), *20*, *30*, *33*, *34*, *37*, *40*, *162*
find_peaks, 12, *154*
find_peaks(), *18*
find_quantiles, 15
find_spikes, 16, *122*
find_spikes(), *14*
find_valleys *(*find_peaks*)*, 12
fitdistr, *52*, *56*
fitted, *60*, *61*, *68*, *78*, *79*, *86*, *95*, *105–107*, *130*, *131*, *145*

geom_area, *15*
geom_debug, *44*, *59*, *71*, *77*, *84*, *93*, *102*, *133*, *142*
geom_debug_group, *66*
geom_table, *87*
ggpmisc *(*ggpmisc-package*)*, 3
ggpmisc-package, 3
ggrepel, *121*, *153*
glance, *61*, *68*, *77*, *79*, *86*, *95*, *105*, *107*, *131*, *145*
glht, *112*, *116*
gls, *61*, *69*, *79*, *86*, *95*, *106*, *107*, *131*, *146*
group, *46*, *55*, *56*, *62*, *72*, *79*, *87*, *96*, *108*, *116*, *122*, *133*, *134*, *147*, *154*

italic_label *(*plain_label*)*, 20

keep_augment *(*keep_tidy*)*, 18
keep_glance *(*keep_tidy*)*, 18

keep_tidy, 18

layer, *42*, *51*, *58*, *65*, *76*, *83*, *92*, *99*, *111*, *119*, *126*, *139*, *152*
lm, *61*, *69*, *79*, *86*, *95*, *105*, *107*, *131*, *146*
lmodel2, *10*, *12*, *28*, *61*, *69*, *79*, *86*, *95*, *101*, *106–108*, *131*, *146*
ltsReg, *61*, *69*, *79*, *86*, *95*, *105*, *107*, *131*, *146*

mean_value_label *(*plain_label*)*, 20

normalmixEM, *52*, *56*

outcome2factor, 19
outcome2factor(), *30*, *33, 34*, *37*, *40*, *162*

p.adjust, *116*
p_value_label *(*plain_label*)*, 20
package ggpp, *4*
paste, *158*
peaks, *13, 14*, *120*
plain_label, 20
plotmath, *43*, *52*, *84*, *100*, *112*, *121*, *127*, *140, 141*, *154*, *158*
poly2character, 26
predict, *60*, *68*, *78*, *86*, *95*, *105, 106*, *130*, *145*
predict.lmodel2, 27

qss, *147*

r_ci_label *(*plain_label*)*, 20
r_label *(*plain_label*)*, 20
residuals, *61*, *66*, *68*, *73*, *78*, *86*, *95*, *105, 106*, *131*, *145*
rlm, *61*, *69*, *79*, *86*, *95*, *105*, *107*, *131*, *146*
rq, *61*, *69*, *79*, *86*, *95*, *106, 107*, *131*, *146, 147*
rqss, *61*, *69*, *79*, *86*, *95*, *106, 107*, *131*, *146, 147*
rr_ci_label *(*plain_label*)*, 20
rr_label *(*plain_label*)*, 20

S_value_label *(*plain_label*)*, 20
scale_color_logFC *(*scale_colour_logFC*)*, 28
scale_color_outcome *(*scale_colour_outcome*)*, 31
scale_colour_logFC, 28
scale_colour_logFC(), *20*, *33, 34*, *37*, *40*, *162*
scale_colour_outcome, *19*, 31
scale_colour_outcome(), *20*, *30*, *34*, *37*, *40*, *162*
scale_continuous, *30*, *36*, *40*
scale_fill_logFC *(*scale_colour_logFC*)*, 28
scale_fill_outcome, *19*
scale_fill_outcome *(*scale_colour_outcome*)*, 31
scale_manual, *33, 34*
scale_shape_outcome, *19*, 34
scale_shape_outcome(), *20*, *30*, *33*, *37*, *40*, *162*
scale_x_FDR *(*scale_y_Pvalue*)*, 38
scale_x_logFC, 35
scale_x_logFC(), *20*, *30*, *33, 34*, *40*, *162*
scale_x_Pvalue *(*scale_y_Pvalue*)*, 38
scale_y_FDR *(*scale_y_Pvalue*)*, 38
scale_y_logFC *(*scale_x_logFC*)*, 35
scale_y_Pvalue, 38
scale_y_Pvalue(), *20*, *30*, *33, 34*, *37*, *162*
sd_value_label *(*plain_label*)*, 20
se_value_label *(*plain_label*)*, 20
set.seed, *43*, *51*, *58*, *66*, *76*, *83*, *92*, *100, 101*, *111*, *126*, *128*, *139*
size, *116*
sma, *61*, *69*, *79*, *86*, *95*, *106, 107*, *131*, *146*
split, *15*
sprintf, *40, 41*, *120*, *152*, *158*
sprintf_dm, *24*, 40
stat_boxplot, *113*
stat_correlation, 41, *77*, *93*, *158*
stat_density, *15*, *52*
stat_distrmix_area, *15*
stat_distrmix_area *(*stat_distrmix_eq*)*, 49
stat_distrmix_eq, 49, *56*, *108*, *134*, *147*, *158*
stat_distrmix_line, *15*, *56*, *108*, *134*, *147*
stat_distrmix_line *(*stat_distrmix_eq*)*, 49
stat_fit_augment, 57, *61*, *68*, *77*, *79*, *86*, *93*, *95*, *105*, *107*, *131*, *145*
stat_fit_augment(), *80*, *87*, *96*
stat_fit_deviations, *61*, 64, *68*, *79*, *86*, *95*, *105*, *107*, *131*, *145*
stat_fit_fitted, *59*, *61*, *68*, *78*, *86*, *95*, *105*, *107*, *131*, *145*
stat_fit_fitted *(*stat_fit_deviations*)*, 64

stat_fit_glance, *59*, *61*, *68*, 75, *79*, *86*, *93*, *95*, *105*, *107*, *131*, *145*
stat_fit_glance(), *62*, *87*, *96*
stat_fit_residuals, *59*, *61*, *68*, *78*, *86*, *95*, *105*, *106*, *131*, *145*
stat_fit_residuals (stat_fit_deviations), 64
stat_fit_tb, *61*, *68*, *79*, 82, *86*, *95*, *105*, *107*, *131*, *145*
stat_fit_tb(), *62*, *80*, *96*
stat_fit_tidy, *59*, *61*, *68*, *77*, *79*, *86*, 91, *95*, *105*, *107*, *131*, *145*
stat_fit_tidy(), *62*, *80*, *87*
stat_ma_eq, *56*, *61*, *68*, *77*, *78*, *86*, *93*, *95*, 97, *105*, *106*, *108*, *131*, *134*, *145*, *147*, *158*
stat_ma_line, *56*, *59*, *61*, *68*, *78*, *86*, *95*, *105*, *106*, *108*, *130*, *134*, *145*, *147*
stat_ma_line (stat_ma_eq), 97
stat_multcomp, 110
stat_peaks, 118
stat_poly_eq, *56*, *59*, *60*, *68*, *77*, *78*, *86*, *93*, *95*, *105*, *106*, *108*, *124*, 124, *128*, *130*, *134*, *145*, *147*, *158*
stat_poly_line, *56*, *59*, *60*, *68*, *78*, *86*, *95*, *105*, *106*, *108*, *130*, *134*, *145*, *147*
stat_poly_line (stat_poly_eq), 124
stat_quant_band, *56*, *59*, *61*, *68*, *78*, *86*, *95*, *105*, *106*, *108*, *130*, *134*, 137, *145*, *147*
stat_quant_eq, *56*, *61*, *68*, *77*, *78*, *86*, *93*, *95*, *105*, *106*, *108*, *130*, *134*, *145*, *147*, *158*
stat_quant_eq (stat_quant_band), 137
stat_quant_line, *56*, *59*, *61*, *68*, *78*, *86*, *95*, *105*, *106*, *108*, *130*, *134*, *145*, *147*
stat_quant_line (stat_quant_band), 137
stat_quantile, *141*
stat_smooth, *60*, *68*, *78*, *85*, *94*, *105*, *106*, *128*, *130*, *145*
stat_spikes, 151
stat_valleys (stat_peaks), 118
summary.glht, *111*, *116*
swap_xy, 156
symmetric_limits, 156

t_value_label (plain_label), 20
threshold2factor (outcome2factor), 19
tidy, *61*, *68*, *79*, *86*, *93*, *95*, *105*, *107*, *131*, *145*
ttheme_gtdefault, *87*
typeset_numbers, 157

use_label, *46*, *55*, *107*, *115*, *131*, *146*, 158

value2char (sprintf_dm), 40
var_value_label (plain_label), 20

weighted.residuals, *66*
weights, *61*, *68*, *73*, *79*, *86*, *95*, *105*, *107*, *131*, *145*

x, *46*, *55*, *56*, *62*, *72*, *79*, *87*, *96*, *108*, *116*, *122*, *133*, *134*, *147*, *154*
xend, *72*
xmax, *116*
xmin, *116*
xy_outcomes2factor, 161
xy_outcomes2factor(), *20*, *30*, *33*, *34*, *37*, *40*
xy_thresholds2factor (xy_outcomes2factor), 161

y, *46*, *55*, *56*, *62*, *72*, *79*, *87*, *96*, *108*, *116*, *122*, *133*, *134*, *147*, *154*
yend, *72*
ymax, *62*, *116*
ymin, *62*, *116*

z_value_label (plain_label), 20