Package 'erplots'

October 23, 2026

Title Model-Agnostic Exposure-Response Plots

Version 0.2.1

Description Provides a fluent mini-language for building exposure-response plots (model curves/ribbons, quantile-binned summaries, data strips, and grouped distribution panels) from observed data and a fitted exposure-response model. Designed to be model-agnostic: any model object that implements the er_predict() generic (and, optionally, er_simulate() and er_summary()) can be visualised.

License MIT + file LICENSE

Encoding UTF-8

VignetteBuilder knitr

Imports dplyr (>= 1.1.0), ggplot2 (>= 4.0.0), patchwork (>= 1.3.2), purrr, rlang, scales, stats, survival, tibble, tidyselect, withr

Suggests emaxnls (>= 0.1.1.9000), erglm, ertte, hexbin, knitr, MASS, mvtnorm, rmarkdown, spelling, testthat (>= 3.0.0)

URL https://github.com/djnavarro/erplots, https://erplots.djnavarro.net/

BugReports https://github.com/djnavarro/erplots/issues

Depends R (>= 4.1.0)

LazyData true

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

Config/Needs/website djnavarro/waeponwifestre, rmarkdown

Collate 'data.R' 'er-generics.R' 'er-plot-add.R' 'er-plot-api.R' 'er-plot-build.R' 'er-plot-compose.R' 'er-plot-layer.R' 'er-plot-style.R' 'er-style-registry.R' 'er-plot-style-data.R' 'er-plot-style-group.R' 'er-plot-style-model.R' 'er-plot-style-quantile.R' 'er-plot-style-summary.R' 'er-plot-theme.R' 'er-tte-add.R' 'er-tte-api.R'

'er-tte-layer.R' 'er-tte-style-censor.R' 'er-tte-style-curve.R'
'er-tte-style-model.R' 'er-tte-style-risktable.R'
'er-tte-style-summary.R' 'er-tte-style.R' 'er-tte-theme.R'
'er-vpc-add.R' 'er-vpc-api.R' 'er-vpc-build.R' 'er-vpc-layer.R'
'er-vpc-style-observed.R' 'er-vpc-style-simulated.R'
'er-vpc-style.R' 'er-vpc-theme.R' 'utils-helpers.R'

Language en-GB

NeedsCompilation no

Author Danielle Navarro [aut, cre] (ORCID:
<https://orcid.org/0000-0001-7648-6578>)

Maintainer Danielle Navarro <djnavarro@protonmail.com>

Repository CRAN

Date/Publication 2026-10-23 11:00:08 UTC

Contents

ci_clopper_pearson 3
ci_poisson 4
ci_quantile 5
ci_t 6
cut_quantile 6
erplots_data 8
er_model_interface 10
er_plot 13
er_plot_add_data 14
er_plot_add_groups 17
er_plot_add_model 19
er_plot_add_quantiles 22
er_plot_add_summary 25
er_plot_build 26
er_plot_theme 27
er_style 30
er_style_data 33
er_style_group 36
er_style_labels 41
er_style_model 42
er_style_quantile 45
er_style_summary 50
er_style_tag 53
er_style_tte 58
er_style_tte_censor 59
er_style_tte_curve 61
er_style_tte_model 62
er_style_tte_risktable 64
er_style_tte_summary 65
er_style_vpc 69

er_style_vpc_observed 71
er_style_vpc_simulated 75
er_tte 79
er_tte_add_censor 81
er_tte_add_curve 82
er_tte_add_model 83
er_tte_add_risktable 85
er_tte_add_summary 86
er_tte_build 88
er_tte_theme 89
er_vpc 91
er_vpc_add_observed 93
er_vpc_add_simulated 94
er_vpc_build 96
er_vpc_theme 96

Index **99**

`ci_clopper_pearson` *Clopper-Pearson confidence interval for binary data*

Description

Computes an exact binomial confidence interval for a proportion.

Usage

```
ci_clopper_pearson(x, n, conf_level = 0.95)
```

Arguments

`x`	Number of successes
`n`	Total number of trials
`conf_level`	Confidence level

Details

Used by the quantile-binned summary layer (see `er_plot_add_quantiles()`) to compute empirical response-rate confidence intervals. This assumes a binary (0/1) response.

Value

Named numeric vector, with confidence level stored as an attribute

Examples

```
ci_clopper_pearson(1, 10)
```

`ci_poisson` *Exact Poisson confidence interval for a count rate*

Description

Computes an exact Poisson confidence interval for a count rate.

Usage

```
ci_poisson(x, n, conf_level = 0.95)
```

Arguments

`x`	Vector (or sum) of observed counts, e.g. all counts falling in one exposure bin
`n`	Number of units the counts were accumulated over (e.g. the number of observations in the bin); the rate being estimated is `sum(x) / n`
`conf_level`	Confidence level

Details

The count-response analogue of `ci_clopper_pearson()`, used by the quantile-binned summary layer (see `er_plot_add_quantiles()`) and `er_vpc_add_observed()`/`er_vpc_add_simulated()` when `response_type = "count"` is explicitly declared. Unlike `ci_t()` (the default, opt-in-required approximation used when a count response auto-detects or is declared `"continuous"`), this interval is exact and never produces a negative lower bound. Uses the standard exact ("Garwood") Poisson interval, derived from the chi-squared/gamma relationship; if the total count is 0, the lower bound is 0.

Value

Named numeric vector (`lower`, `upper`) for the rate `sum(x) / n`, with confidence level stored as an attribute.

Examples

```
ci_poisson(3, 10)
```

ci_quantile *Distribution-free confidence interval for a sample quantile*

Description

Computes a nonparametric confidence interval for a sample quantile using the order-statistic method: the interval endpoints are order statistics of `x`, chosen via the binomial distribution of ranks so that no assumption is made about the shape of `x`'s distribution.

Usage

```
ci_quantile(x, prob = 0.5, conf_level = 0.95)
```

Arguments

`x`	Numeric vector of observations
`prob`	Quantile probability (e.g. `0.1` for the tenth percentile)
`conf_level`	Confidence level

Details

Used by `er_vpc_add_observed()` to compute a confidence interval for each requested percentile of the observed response within an exposure bin (the observed-side analogue of the across-replicate percentile interval `er_vpc_add_simulated()` gets from simulated data, powering `er_style_vpc_observed_quantile_err` Like `ci_clopper_pearson()`, this interval is exact for its target coverage but conservative – the discreteness of the binomial rank distribution means the achieved coverage can exceed the nominal `conf_level`, especially for a small bin or an extreme `prob`. The candidate rank indices are clipped to `[1, length(x)]`, so a very small or extreme-`prob` bin returns a (still valid, but wider-than-nominal) interval built from the most extreme order statistics available rather than `NA`.

Value

Named numeric vector (`lower`, `upper`), with confidence level stored as an attribute. Returns `c(lower = NA, upper = NA)` if fewer than 2 non-missing values are supplied.

References

Conover, W. J. (1999). *Practical Nonparametric Statistics* (Third edition). New York: John Wiley & Sons. ISBN 0-471-16068-7.

Examples

```
ci_quantile(rnorm(100), prob = 0.1)
```

ci_t *t-interval confidence interval for the mean of continuous data*

Description

Computes a t-distribution confidence interval for a sample mean.

Usage

```
ci_t(x, conf_level = 0.95)
```

Arguments

x	Numeric vector of observations
conf_level	Confidence level

Details

Used by the quantile-binned summary layer (see `er_plot_add_quantiles()`) and `er_vpc_add_observed()`/`er_vpc_add_simulated()` to compute a confidence interval for the mean response within an exposure bin, for continuous (and, as an approximation, count) responses. This is the continuous-response analogue of `ci_clopper_pearson()`. NAs in `x` are dropped before computing the interval.

Value

Named numeric vector (`lower`, `upper`), with confidence level stored as an attribute. Returns `c(lower = NA, upper = NA)` if fewer than 2 non-missing values are supplied.

Examples

```
ci_t(rnorm(20))
```

cut_quantile *Cut a continuous variable into quantiles*

Description

`cut_quantile()` bins a numeric vector into `n_bins` quantile groups. `cut_exposure_quantile()` does the same for an exposure variable, additionally keeping placebo (`0`) observations in their own bin.

Usage

```
cut_exposure_quantile(
  x,
  n_bins = 4,
  is_placebo = NULL,
  ties = c("upward", "downward", "split-even"),
  seed = NULL,
  quantile_type = 7,
  labeller = NULL
)

cut_quantile(
  x,
  n_bins = 4,
  ties = c("upward", "downward", "split-even"),
  seed = NULL,
  quantile_type = 7,
  labeller = NULL
)
```

Arguments

x	Numeric vector
n_bins	Number of bins
is_placebo	Logical vector indicating placebo samples
ties	Rule for assigning a value that sits exactly on an interior break point, where the bin membership would otherwise be ambiguous. "upward" (the default, matching prior behaviour) is equivalent to cut() with right = TRUE; "downward" is equivalent to right = FALSE; "split-even" randomly divides each tied group between its two candidate bins so that final bin sizes are as equal as possible, rather than sending every tied value the same direction.
seed	Optional single number used to seed the random tie-break used by ties = "split-even" (ignored for "upward"/"downward", which involve no randomness). NULL (the default) draws from the ambient RNG stream and so is not reproducible across calls; pass a seed for reproducible bin assignment.
quantile_type	Integer between 1 and 9, passed straight through as stats::quantile()'s own type argument to compute the quantile break points. Defaults to 7, matching stats::quantile()'s own default.
labeller	Controls the labels used for the n_bins quantile bins (cut_exposure_quantile()'s separate "Placebo" level is always used as-is, regardless of labeller). NULL (the default) labels bins "Q1", "Q2", etc. A function is called as labeller(n_bins, breaks) (the actual bin count and the n_bins + 1 quantile cutpoints, after any resolution-driven fallback – see @details below) and must return a character vector of length n_bins; this is the hook for, e.g., range-style labels built from breaks. A character vector is used directly as the n_bins labels.

Details

Both functions error if `x` has fewer than 2 distinct non-missing values, since quantile bins aren't well-defined in that case. If `x` doesn't have enough resolution to distinguish all `n_bins` requested bins (e.g. many repeated values clustered at one end), both functions warn and fall back to using as many bins as the data supports, rather than erroring or silently showing fewer bins with no explanation. `cut_exposure_quantile()`'s `"breaks"` attribute is read back out by quantile-layer builders that draw bin-boundary separators (e.g. `er_style_quantile_errorbar_vlines()`) via `attr(exposure_bins, "breaks")`. Because that fallback can lower `n_bins` below what was originally requested, a character-vector `labeller` is length-checked against the *actual* bin count, not the requested one, and errors informatively on a mismatch.

Value

A factor with `"ties"` and `"quantile_type"` attributes recording those two arguments. `cut_exposure_quantile()`'s result additionally carries a `"breaks"` attribute holding the `n_bins + 1` quantile cutpoints used to form the bins.

Examples

```
x <- rnorm(100)
cut_quantile(x)
cut_exposure_quantile(abs(x))
cut_quantile(x, ties = "split-even", seed = 8213)
cut_quantile(x, quantile_type = 1)
cut_quantile(x, labeller = function(n_bins, breaks) paste0("Group ", 1:n_bins))
cut_quantile(x, labeller = c("Low", "Mid-low", "Mid-high", "High"))
```

---

erplots_data *Simulated exposure-response data*

Description

A simulated dataset with multiple exposure columns and response columns spanning all three response types, designed to demonstrate every layer of the erplots mini-language without depending on any companion model-fitting package for the data itself.

Usage

```
erplots_data
```

Format

A tibble with 4,000 rows (one per simulated subject) and 15 columns:

subject_id Integer subject identifier, `1:4000`.

dose_mg Numeric assigned dose in mg: one of `0`, `10`, `30`, `100`, `300`.

dose_group Ordered factor version of dose_mg ("Placebo" < "10 mg" < "30 mg" < "100 mg" < "300 mg"), about 800 subjects per level. A natural stratify_by/grouping column.

study_id Factor, "Study 1"-"Study 4" (400/800/1200/1600 subjects respectively). A purely administrative label, independent of dose/exposure/response by construction – see Details.

bodyweight_kg Numeric bodyweight covariate.

age_years Numeric age covariate.

sex Factor covariate, "F"/"M".

renal_function Factor covariate, "Normal"/"Mild"/"Moderate".

auc_ss Numeric exposure: steady-state AUC (cumulative exposure). 0 for placebo subjects.

cmax_ss Numeric exposure: steady-state peak concentration.

cmin_ss Numeric exposure: steady-state trough concentration.

biomarker_change Continuous response, Emax-shaped in auc_ss.

responder Binary (0/1) response, Emax-shaped on the logit scale in cmax_ss.

adverse_event Binary (0/1) response, log-linear (plain logistic regression, no saturation) in auc_ss.

symptom_score Continuous response, linear in cmin_ss.

n_events Integer count response, log-linear Poisson rate in auc_ss.

Details

The three exposure columns (auc_ss, cmax_ss, cmin_ss) come from a simplified, internally-consistent PK-flavoured simulation (individual clearance driven by bodyweight_kg/renal_function, with between-subject variability) rather than a literal pharmacokinetic model – good enough to produce a plausible, correlated exposure triple, not a validated PK simulator.

Each response column is paired with the exposure column and mechanism that makes it a natural fit for one modelling scenario:

Response	Exposure	Scenario
biomarker_change	auc_ss	Emax (continuous)
responder	cmax_ss	Emax (binary), e.g. emaxnls::emax_logistic()
adverse_event	auc_ss	logistic regression
symptom_score	cmin_ss	linear regression
n_events	auc_ss	Poisson regression

At 4,000 rows, a raw-point data-layer overlay (er_style_data_overlay()) visibly overplots – see the relevant example below, which uses er_style_data_hex() instead.

study_id ("Study 1"-"Study 4", unevenly sized: 400/800/1200/1600 subjects) is included purely as a convenient filtering column: it's independent of dose, exposure, and response by construction, so subsetting to a single study (e.g. dplyr::filter(erplots_data, study_id == "Study 1")) gives a much smaller sample that still spans the full dose range – useful for illustrating how the same plot looks with less data (e.g. whether a raw-point overlay is legible again once N drops, or whether er_style_data_hex()'s bins become too sparse to be useful).

Source

Simulated; see `data-raw/erplots_data.R` for the full generating code.

Examples

```
erplots_data

# Logistic regression: adverse_event ~ auc_ss
if (requireNamespace("erglm", quietly = TRUE)) {
 mod <- erglm::erglm_model(adverse_event ~ auc_ss, data = erplots_data, family = binomial())
  erplots_data |>
    er_plot(auc_ss, adverse_event) |>
    er_plot_add_model(mod) |>
    er_plot_add_summary(model = mod) |>
    plot()
}

# Linear regression: symptom_score ~ cmin_ss
if (requireNamespace("erglm", quietly = TRUE)) {
 mod <- erglm::erglm_model(symptom_score ~ cmin_ss, data = erplots_data, family = gaussian())
  erplots_data |>
    er_plot(cmin_ss, symptom_score) |>
    er_plot_add_model(mod) |>
    plot()
}

# Linear regression: symptom_score ~ cmin_ss, with a hex-binned data layer
if (requireNamespace("erglm", quietly = TRUE) && requireNamespace("hexbin", quietly = TRUE)) {
 mod <- erglm::erglm_model(symptom_score ~ cmin_ss, data = erplots_data, family = gaussian())
  erplots_data |>
    er_plot(cmin_ss, symptom_score) |>
    er_plot_add_model(mod) |>
    er_plot_add_data(style = er_style_data_hex) |>
    plot()
}

# Filtering to one study (n = 400) for a smaller-sample illustration: a
# Poisson regression n_events ~ auc_ss with scatter plot data layer
if (requireNamespace("erglm", quietly = TRUE)) {
  small_data <- erplots_data[erplots_data$study_id == "Study 1", ]
  mod <- erglm::erglm_model(n_events ~ auc_ss, data = small_data, family = poisson())
  small_data |>
    er_plot(auc_ss, n_events, response_type = "count") |>
    er_plot_add_model(mod) |>
    er_plot_add_data() |>
    plot()
}
```

`er_model_interface` *Model interface for exposure-response plots*

Description

erplots draws exposure-response plots from fitted models that implement this small interface, rather than assuming a particular model class:

- Implement er_predict() for basic plotting support.
- Implement er_simulate() for simulation-based visualisations.
- Implement er_summary() for summary annotations.
- Implement er_predict_survival() for a parametric survival-curve overlay in the er_tte() grammar (er_tte_add_model()).

Usage

```
er_predict(model, newdata, conf_level = 0.95, ...)

er_simulate(model, newdata, nsim = 100, seed = NULL, ...)

er_summary(model, ...)

er_predict_survival(model, newdata, time_grid, conf_level = 0.95, ...)
```

Arguments

model	A fitted exposure-response model object.
newdata	A data frame of covariate values at which to predict. For er_predict_survival(), one row per covariate profile (e.g. one per stratum level) – it does *not* include a time column; times come from time_grid instead (see "Details").
conf_level	Confidence level for the prediction interval.
...	Passed to methods.
nsim	Number of simulation replicates.
seed	Optional RNG seed.
time_grid	Numeric vector of times at which to predict S(t).

Details

er_plot_add_model() does not verify that model was fit on the same exposure/response variables as the plot; that compatibility is the caller's responsibility. When er_plot_add_model() builds newdata, it always includes the exposure and, if stratified, strata variables, plus reference values for any other covariates in the model's original fitting data.

Strata membership for er_predict_survival() is carried on newdata the same way: as an ordinary column (named after the er_tte() object's own stratify_by variable), never implicit in model itself. er_tte_add_model() builds one newdata row per stratum level (or a single row, unstratified), filling any other covariate the model references with a reference value exactly as er_plot_add_model() already does (see its "Details").

A method may rely on caller-supplied extra arguments being forwarded through ...: er_plot_add_model()'s predict_args, er_plot_add_summary()'s summary_args, and er_vpc_add_simulated()'s simulate_args

are each spliced into the corresponding generic call (`er_predict()`/`er_summary()`/`er_simulate()` respectively), so a model-specific argument beyond the fixed contract below (e.g. a landmark time for a time-to-event model) has a documented path to reach the method. These are deliberately kept separate from each `er_plot_add_*()`/`er_vpc_add_*()` function's own `...`, which is reserved for the `style` builder instead (see `er_style()`'s "Passing extra arguments to a builder" section) – a method should not assume it receives anything passed via that `...`.

Value

- `er_predict()` returns `newdata` with three additional columns: `fit_resp` (point prediction), `ci_lower`, and `ci_upper`.
- `er_simulate()` returns a data frame containing `nsim` replicates of `newdata`, with a `sim_id` column identifying each replicate, and a `fit_resp` column giving the simulated prediction for that replicate (reflecting parameter uncertainty). Models that cannot support simulation-based visualisation should not implement a method; the default method returns `NULL`; callers should treat a `NULL` result as "not available" rather than an error.

  A method may *additionally* return a `sim_resp` column: a full response-scale draw for that replicate/observation, reflecting both parameter uncertainty (as `fit_resp` already does) and observation-level sampling/residual noise (e.g. a 0/1 draw for a binary response, an integer draw for a count response, a draw including residual variance for a continuous response) – not just the fitted mean/probability. This is what `er_vpc_add_simulated()`'s `model` argument requires: a visual predictive check needs simulated observations comparable to the actually observed data, not points on the mean curve, which is a genuinely different question from the one `fit_resp` (used by `er_style_model_spaghetti()`) answers. `sim_resp` is independently optional – a method can supply `fit_resp` alone (as every implementation did before `sim_resp` existed, and as remains sufficient for spaghetti plots), or both columns from the same call. `er_vpc_add_simulated()` treats a `sim_resp`-less result the same way it treats an outright `NULL`: "predictive simulation not available for this model."
- `er_summary()` returns `NULL` (nothing available – the default method's behaviour), or a named list with any of the following independently optional keys. Unrecognised keys are permitted and ignored by built-in builders, giving a model package room to stash extra fields for its own custom builders.
  - `p_value`: a single headline p-value (or `NULL`) for "the" exposure effect, when the model has one unambiguous candidate (e.g. a GLM's exposure coefficient). A model with no single privileged parameter (e.g. a multi-parameter nonlinear Emax model, with separate `E0`/`Emax`/`EC50`/`Hill` terms and no obviously "the" effect) should return `NULL` here rather than picking an arbitrary term – `er_style_summary_pvalue()` already treats `NULL` as "nothing to show".
  - `coefficients`: a tibble/data frame with one row per model parameter, for builders (e.g. `er_style_summary_coefficients()`) that display more than a single p-value. Columns follow this package's snake_case convention rather than `broom::tidy()`'s dotted names: `term` (required), `label` (optional display name, falls back to `term`), `estimate` (required), and optional `std_error`, `statistic`, `p_value`, `conf_low`, `conf_high` (each `NA` if not computed/meaningful). `NULL` if not available.
  - `glance`: a single-row tibble/data frame of model-level goodness-of-fit, `broom::glance()`-style: optional `n`, `df_residual`, `logLik`, `aic`, `bic`, `deviance`, `r_squared` (`NA` where not meaningful, e.g. non-Gaussian models), `converged`. Reserved for future builders; no built-in builder currently consumes it. `NULL` if not available.

This is purely additive: a method that only ever returns `list(p_value = ...)` (as above) continues to work unchanged.

- `er_predict_survival()` returns `newdata`, cross-joined with `time_grid` (one row per `newdata` row x `time_grid` value), with three additional columns: `time`, `fit_survival` (the point estimate of `S(time | newdata row)`), `ci_lower`, and `ci_upper`. Unlike `er_predict()`, `newdata` here never includes a time/exposure column itself – `time_grid` is a separate argument, so a method can build the full time grid in one call per covariate profile rather than being handed a pre-crossed data frame. There is no default method that returns `NULL`; a model with no survival-curve support should simply not implement this generic, and `er_tte_add_model()` errors informatively (via the default method above) if called with one.

Examples

```
# a bare-bones er_predict() method for a plain `lm` fit
toy_fit <- lm(biomarker_change ~ auc_ss, data = erplots_data)
class(toy_fit) <- c("toy_lm", class(toy_fit))

er_predict.toy_lm <- function(model, newdata, conf_level = 0.95, ...) {
  z <- -qnorm((1 - conf_level) / 2)
  pred <- predict(model, newdata = newdata, se.fit = TRUE)
  newdata$fit_resp <- pred$fit
  newdata$ci_lower <- pred$fit - z * pred$se.fit
  newdata$ci_upper <- pred$fit + z * pred$se.fit
  newdata
}

er_predict(toy_fit, newdata = data.frame(auc_ss = c(100, 500, 900)))
```

---

er_plot *The exposure-response plotting mini-language*

Description

Create an `er_plot` specification for exposure-response visualization. Build the plot by adding layers (model, summary, quantiles, data, groups) and render with `plot()`/`print()` or `er_plot_build()`.

Usage

```
er_plot(data, exposure, response, stratify_by = NULL, response_type = "auto")
```

Arguments

`data`	Data frame or tibble containing the observed data.
`exposure`	Exposure variable (one variable, unquoted).
`response`	Response variable (one variable, unquoted).
`stratify_by`	Stratification variable used for colour and fill (one variable, unquoted).
`response_type`	One of `"auto"`, `"binary"`, `"continuous"`, or `"count"`.

Details

Layers are either singleton or additive: model, summary, quantile, and data layers are singleton (a second call replaces the previous); groups are additive (each call adds a panel).

`stratify_by` declares a discrete variable used for colour/fill across layers; each layer's `keep_strata` controls whether it uses stratification. Rows with `NA` in the stratification variable are kept as their own level. A numeric `stratify_by` errors – bin it yourself first with `cut_quantile()`/`cut_exposure_quantile()` and pass the resulting factor.

`response_type` governs response-scale defaults and which interval method the quantile and VPC layers use; see `response_type` below and `er_plot_add_quantiles()` for details.

Value

An (empty) plot object of class `er_plot`.

See Also

`er_plot_add_model()`, `er_plot_add_summary()`, `er_plot_add_quantiles()`, `er_plot_add_data()`, `er_plot_add_groups()`, `er_plot_build()`, `er_plot_theme()`, er_model_interface

Examples

```
if (requireNamespace("erglm", quietly = TRUE)) {
library(erglm)
mod <- erglm_model(ae1 ~ aucss, erglm_data, family = binomial())

erglm_data |>
  er_plot(aucss, ae1) |>
  er_plot_add_model(mod) |>
  er_plot_add_quantiles() |>
  er_plot_add_groups(aucss) |>
  plot()
}
```

---

er_plot_add_data *Add a raw-data layer*

Description

Adds the data layer: individual observations. By default, points are drawn as an overlay showing the exposure and response values in the main panel of the plot, but other possibilities are available.

Usage

```
er_plot_add_data(object, style = NULL, keep_strata = NULL, panel = "both", ...)
```

Arguments

`object`	Partially constructed plot (has S3 class `er_plot`).
`style`	Style used to draw the data layer. Can either be a string corresponding to one of the registered style labels (e.g., `"overlay"`, the default), or a builder function used to compute the relevant plot object (see "Styles" below).
`keep_strata`	Logical; whether this layer should use stratification. Defaults to `TRUE` when a stratification variable has been specified, and `FALSE` otherwise.
`panel`	Character string: `"upper"`, `"lower"`, or `"both"` (the default). Only meaningful for `er_style_data_boxjitter()` on a binary response; see "Details" for when `"both"` is required.
`...`	Additional named arguments forwarded to the `style` builder function when the plot is built.

Value

The input `object`, with the data layer added.

Styles

The following pre-defined styles are available for this layer. Please see the documentation for the corresponding builder function to see what customisation options are available:

Label	Builder	Description
`"overlay"`	`er_style_data_overlay()`	Raw points (jittered for a binary response) drawn on the main panel (the de
`"hex"`	`er_style_data_hex()`	2D hexbin density of the raw points on the main panel.
`"boxjitter"`	`er_style_data_boxjitter()`	Boxplot + jittered points in a stacked panel, split by response (binary respo

See `er_style()` for details on how style builder functions are defined for the exposure-response mini-grammar, should a custom style be required.

Default builders

The default builder for the data layer is `er_style_data_overlay()`, which creates a plain scatter plot for continuous/count responses, or a scatter with a small vertical jitter for a binary response (whose y-values are exactly 0/1 and would otherwise overplot into two solid lines). This works uniformly across all three response types, with no response-type dispatch on which builder to use. `er_style_data_boxjitter()` instead uses a panel-based design, and is binary-response-only: responders (`response == 1`) get a boxplot + jittered points in an upper panel and non-responders (`response == 0`) get the same in a lower panel, so the panel shows the exposure *distribution* conditional on response, not just raw points. There is no built-in "panel"-layout builder for a continuous/count response; `panel` must be `"both"` (the default) for these response types regardless of builder, since there's no upper/lower partition to select from.

Structural families

Every data-layer builder declares which of these two *structural* families it belongs to via `er_style_tag()` – `"overlay"` (a single call merged into the main panel) or `"panel"` (one-or-more panels stacked below the base plot) – which `er_plot_add_data()` reads off `style` to decide how to assemble the layer, rather than taking a separate argument for it. This makes the pairing structural rather than incidental: `er_style_data_overlay()` can never be routed into upper/lower panels, and `er_style_data_boxjitter()` can never be merged into the main panel. See `er_style_tag()` and `er_style()` for how to tag a custom builder the same way. If `style` is tagged with a `layer` other than `"plot_data"`, `er_plot_add_data()` errors informatively; an untagged builder is never checked (only `layout` is a hard requirement).

Effect of `keep_strata`

`keep_strata`'s effect also depends on a builder's structural family: for an "overlay"-layout builder it always means a shared colour aesthetic, for any response type; for a "panel"-layout builder on a continuous/count response it instead produces one panel per stratum level rather than a shared colour aesthetic. `panel` must be `"both"` for an "overlay"-layout builder (there's no upper/lower partition to select from) and for a continuous/count response under a "panel"-layout builder (same reason).

See Also

`er_plot()`, `er_plot_add_model()`, `er_plot_add_summary()`, `er_plot_add_quantiles()`, `er_plot_add_groups()`, `er_style()`

Examples

```
if (requireNamespace("erglm", quietly = TRUE)) {
library(erglm)
mod2 <- erglm_model(ae2 ~ aucss + sex, erglm_data, family = binomial())
erglm_data |>
  er_plot(aucss, ae2, stratify_by = sex) |>
  er_plot_add_model(mod2) |>
  er_plot_add_quantiles() |>
  er_plot_add_data() |>
  plot()

# continuous response: overlay works the same way, with no
# response-type-specific styling needed
mod3 <- erglm_model(biomarker_change ~ aucss, erglm_data, family = gaussian())
erglm_data |>
  er_plot(aucss, biomarker_change) |>
  er_plot_add_model(mod3) |>
  er_plot_add_data() |>
  plot()

# panel-based design, binary-response only: a boxplot + jittered
# points per panel (responders above, non-responders below), instead
# of an overlay in the main panel
erglm_data |>
  er_plot(aucss, ae2, stratify_by = sex) |>
```

```
  er_plot_add_model(mod2) |>
  er_plot_add_data(style = er_style_data_boxjitter) |>
  plot()

# plug in a 2D density in the main panel instead of a scatter; tagging
# it "overlay" via `er_style_tag()` keeps it in the single main-panel
# layout -- see `?er_style`
build_data_density <- er_style_tag(
  function(data, config, stratify, exposure, response, strata, theme, ...) {
    ggplot2::geom_density_2d(
      data = data,
      mapping = ggplot2::aes(x = .data[[exposure$name]], y = .data[[response$name]])
    )
  },
  layout = "overlay"
)
erglm_data |>
  er_plot(aucss, biomarker_change) |>
  er_plot_add_model(mod3) |>
  er_plot_add_data(style = build_data_density) |>
  plot()
}
```

`er_plot_add_groups` *Add a grouped exposure-distribution panel*

Description

Adds a group layer: a boxplot/violin panel showing the exposure distribution, split by one or more grouping variables (continuous grouping variables are binned into quantiles first).

Usage

```
er_plot_add_groups(
  object,
  group_by,
  style = NULL,
  keep_strata = NULL,
  n_bins = NULL,
  ties = "upward",
  quantile_type = 7,
  labeller = NULL,
  ...
)
```

Arguments

`object`	Partially constructed plot (has S3 class `er_plot`).
`group_by`	Grouping variables to define groups for distribution plots (a tidyselection of variables).
`style`	Style used to draw the group layer. Can either be a string corresponding to one of the registered style labels (e.g., `"boxplot"`, the default), or a builder function used to compute the relevant plot object (see "Styles" below).
`keep_strata`	Logical; whether this layer should use stratification. Defaults to `TRUE` when a stratification variable has been specified, and `FALSE` otherwise.
`n_bins`	Number of quantile bins used for continuous grouping variables (`NULL`, the default, uses `cut_quantile()`'s own default). Applied identically to every grouping variable added by this call.
`ties, quantile_type, labeller`	Passed straight through to `cut_quantile()`/`cut_exposure_quantile()` to control how a continuous grouping variable is split into bins – see their documentation for what each controls. Applied identically to every grouping variable added by this call.
`...`	Additional named arguments forwarded to the `style` builder function when the plot is built.

Details

Unlike the other four layers, the groups layer is **additive**: each call adds another panel alongside any already added by a previous call, rather than replacing it.

`er_style_group_violin()` and `er_style_group_histogram()` are the other built-in `style` options; any function matching the standard `(data, config, stratify, exposure, response, strata, theme, ...)` signature can be supplied instead. If `style` is tagged with a `layer` (via `er_style_tag()`) other than `"plot_group"`, this errors informatively; an untagged builder is never checked.

`keep_strata = TRUE` errors if `group_by` is itself the plot's stratification variable, since that would mean grouping and stratifying by the same column at once; pass `keep_strata = FALSE` for that grouping variable instead.

`n_bins`/`ties`/`quantile_type`/`labeller` are local to this call – different grouping variables (including across separate `er_plot_add_groups()` calls) aren't required to agree, and generally shouldn't: they're usually different variables with no reason to share a binning scheme. The one exception is grouping by the plot's own exposure variable, which risks silently disagreeing with `er_plot_add_quantiles()`'s own exposure-binning; `er_plot_build()` warns (doesn't error) if the two disagree in that specific case.

Value

The input `object`, with a group panel added.

Styles

The following pre-defined styles are available for this layer. Please see the documentation for the corresponding builder function to see what customisation options are available:

Label	Builder	Description
"boxplot"	er_style_group_boxplot()	Boxplot per group level, group levels on the y-axis (the default).
"violin"	er_style_group_violin()	Violin per group level, group levels on the y-axis.
"histogram"	er_style_group_histogram()	Histogram per group level, group levels on facet strips, y-axis free
"linerange"	er_style_group_linerange()	Median dot with inner/outer-range lines per group level, group leve
"boxjitter"	er_style_group_boxjitter()	"boxplot" with jittered raw exposure values overlaid.
"violinjitter"	er_style_group_violinjitter()	"violin" with jittered raw exposure values overlaid.

See er_style() for details on how style builder functions are defined for the exposure-response mini-grammar, should a custom style be required.

See Also

er_plot(), er_plot_add_model(), er_plot_add_summary(), er_plot_add_quantiles(), er_plot_add_data(), er_style()

Examples

```
if (requireNamespace("erglm", quietly = TRUE)) {
library(erglm)
mod <- erglm_model(ae1 ~ aucss, erglm_data, family = binomial())
erglm_data |>
  er_plot(aucss, ae1) |>
  er_plot_add_model(mod) |>
  er_plot_add_groups(aucss) |>
  plot()

# additive: a second call adds a second panel rather than replacing the first
erglm_data |>
  er_plot(aucss, ae1) |>
  er_plot_add_model(mod) |>
  er_plot_add_groups(aucss) |>
  er_plot_add_groups(treatment) |>
  plot()
}
```

---

er_plot_add_model *Add a fitted-model curve/ribbon layer*

Description

Adds the model layer: a fitted exposure-response curve with an uncertainty ribbon, or possibly a spaghetti plot of simulated draws.

Usage

```
er_plot_add_model(
  object,
  model,
  style = NULL,
  keep_strata = NULL,
  conf_level = 0.95,
  predict_args = list(),
  ...
)
```

Arguments

`object`	Partially constructed plot (has S3 class `er_plot`).
`model`	A fitted exposure-response model. Must implement `er_predict()`.
`style`	Style used to draw the model curve/ribbon layer. Can either be a string corresponding to one of the registered style labels (e.g., `"ribbonline"`, the default), or a builder function used to compute the relevant plot object (see "Styles" below).
`keep_strata`	Logical; whether this layer should use stratification. Defaults to `TRUE` when a stratification variable has been specified, and `FALSE` otherwise.
`conf_level`	Confidence level for the prediction ribbon. Defaults to `0.95`.
`predict_args`	A named list of additional arguments forwarded to `er_predict()` when generating model-based predictions.
`...`	Additional named arguments forwarded to the `style` builder function when the plot is built.

Details

This layer uses `er_predict()` to compute model predictions on the response scale. `model` may reference covariates beyond the exposure and strata variables. erplots fills any additional covariates from the plot data with a reference value (first factor level or numeric mean) when building the prediction grid. erplots does not check that `model` was fit on the same exposure/response as the plot; the caller must ensure compatibility.

Value

The input `object`, with the model layer added.

Styles

The following pre-defined styles are available for this layer. Please see the documentation for the corresponding builder function to see what customisation options are available:

Label	Builder	Description
`"ribbonline"`	`er_style_model_ribbonline()`	Fitted curve with an uncertainty ribbon (the default).
`"line"`	`er_style_model_line()`	Fitted curve only, no ribbon.

"spaghetti"	er_style_model_spaghetti()	Fitted curve plus a spaghetti plot of simulated draws, for models implen

See er_style() for details on how style builder functions are defined for the exposure-response mini-grammar, should a custom style be required.

See Also

er_plot(), er_plot_add_summary(), er_plot_add_quantiles(), er_plot_add_data(), er_plot_add_groups(), er_style()

Examples

```
if (requireNamespace("erglm", quietly = TRUE)) {
library(erglm)
mod <- erglm_model(ae1 ~ aucss, erglm_data, family = binomial())
erglm_data |>
  er_plot(aucss, ae1) |>
  er_plot_add_model(mod) |>
  plot()

# a spaghetti plot instead of the default ribbon
erglm_data |>
  er_plot(aucss, ae1) |>
  er_plot_add_model(mod, style = er_style_model_spaghetti) |>
  plot()

# the same spaghetti plot, selected by its registered label instead
# (see `?er_style_labels`)
erglm_data |>
  er_plot(aucss, ae1) |>
  er_plot_add_model(mod, style = "spaghetti") |>
  plot()

# plug in a fully custom model-curve builder
build_model_dashed <- function(data, config, stratify, exposure, response, strata, theme, ...) {
  ggplot2::geom_line(
    data = config$predictions,
    mapping = ggplot2::aes(x = .data[[exposure$name]], y = fit_resp),
    linetype = "dashed"
  )
}
erglm_data |>
  er_plot(aucss, ae1) |>
  er_plot_add_model(mod, style = build_model_dashed) |>
  plot()

# a model with a covariate beyond the exposure variable still works even when
# this layer isn't stratifying by it: `sex` is set to a reference value
# when building the prediction grid, which may not be what the user wants
mod_sex <- erglm_model(ae1 ~ aucss + sex, erglm_data, family = binomial())
erglm_data |>
```

```
  er_plot(aucss, ae1) |>
  er_plot_add_model(mod_sex) |>
  plot()
}
```

`er_plot_add_quantiles` *Add a quantile-binned response summary layer*

Description

Adds the quantile layer: exposure is cut into quantile bins (see `cut_exposure_quantile()`) and, within each bin, the response is summarised with a point estimate and confidence interval.

Usage

```
er_plot_add_quantiles(
  object,
  style = NULL,
  keep_strata = NULL,
  conf_level = 0.95,
  n_bins = 4,
  ties = "upward",
  quantile_type = 7,
  labeller = NULL,
  ...
)
```

Arguments

`object`	Partially constructed plot, an `er_plot` object.
`style`	Style used to draw the quantile summary layer. Can either be a string corresponding to one of the registered style labels (e.g., `"errorbar"`, the default), or a builder function used to compute the relevant plot object (see "Styles" below).
`keep_strata`	Logical; whether this layer should use stratification. Defaults to `TRUE` when a stratification variable has been specified, and `FALSE` otherwise.
`conf_level`	Confidence level for the interval. Defaults to `0.95`.
`n_bins`	Number of exposure bins (not counting placebo). Defaults to `4`.
`ties`, `quantile_type`, `labeller`	Passed straight through to `cut_exposure_quantile()` to control how the exposure variable is split into bins – see its documentation for what each controls.
`...`	Additional named arguments forwarded to the `style` builder function when the plot is built.

Details

The type of confidence interval shown depends on the `response_type` set in `er_plot()`:

- `"binary"`: Clopper-Pearson interval (see `ci_clopper_pearson()`)
- `"continuous"`: Student t-interval (see `ci_t()`)
- `"count"`: exact Poisson interval (see `ci_poisson()`)

Note that count responses are not automatically detected as such: they default to `"continuous"` and are summarised the same way as any other continuous response unless `response_type = "count"` is declared explicitly in `er_plot()`.

`n_bins`/`ties`/`quantile_type`/`labeller` are local to this layer – they aren't shared with `er_plot_add_groups()`, even when that layer groups by the same exposure variable. `er_plot_build()` warns (doesn't error) if the two disagree in that specific case; pass matching values to both calls to avoid the warning, or ignore it if the difference is intentional.

Value

The input `object`, with the quantile layer added.

Styles

The following pre-defined styles are available for this layer. Please see the documentation for the corresponding builder function to see what customisation options are available:

Label	Builder	Description
`"errorbar"`	`er_style_quantile_errorbar()`	Point + error bar per bin (the default).
`"errorbar_vlines"`	`er_style_quantile_errorbar_vlines()`	`"errorbar"` plus a labelled vline at every bin bou
`"pointrange"`	`er_style_quantile_pointrange()`	Point + range per bin, via `ggplot2::geom_pointr`
`"pointrange_vlines"`	`er_style_quantile_pointrange_vlines()`	`"pointrange"` plus a labelled vline at every bin b

See `er_style()` for details on how style builder functions are defined for the exposure-response mini-grammar, should a custom style be required.

See Also

`er_plot()`, `er_plot_add_model()`, `er_plot_add_summary()`, `er_plot_add_data()`, `er_plot_add_groups()`, `er_vpc()`, `er_style()`

Examples

```
if (requireNamespace("erglm", quietly = TRUE)) {
library(erglm)
mod <- erglm_model(ae1 ~ aucss, erglm_data, family = binomial())
erglm_data |>
  er_plot(aucss, ae1) |>
  er_plot_add_model(mod) |>
  er_plot_add_quantiles() |>
  plot()
```

```
# continuous response: bin means/t-intervals instead of rates/
# Clopper-Pearson intervals, auto-detected from the response column
mod3 <- erglm_model(biomarker_change ~ aucss, erglm_data, family = gaussian())
erglm_data |>
  er_plot(aucss, biomarker_change) |>
  er_plot_add_model(mod3) |>
  er_plot_add_quantiles() |>
  plot()

# count response: declare response_type = "count" explicitly for an
# exact Poisson interval instead of the t-interval approximation used
# by the auto-detected ("continuous") default
mod4 <- erglm_model(ae_count ~ aucss, erglm_data, family = poisson())
erglm_data |>
  er_plot(aucss, ae_count, response_type = "count") |>
  er_plot_add_model(mod4) |>
  er_plot_add_quantiles() |>
  plot()

# a pointrange instead of the default errorbar
erglm_data |>
  er_plot(aucss, ae1) |>
  er_plot_add_model(mod) |>
  er_plot_add_quantiles(style = er_style_quantile_pointrange) |>
  plot()

# the default errorbar, with dotted lines marking the quantile-bin
# boundaries
erglm_data |>
  er_plot(aucss, ae1) |>
  er_plot_add_model(mod) |>
  er_plot_add_quantiles(style = er_style_quantile_errorbar_vlines) |>
  plot()

# plug in a fully custom builder; see `?er_style`
build_quantile_crossbar <- function(data, config, stratify, exposure,
                                    response, strata, theme, ...) {
  ggplot2::geom_crossbar(
    data = config$summary,
    mapping = ggplot2::aes(x = x_mid, y = y_mid, ymin = ci_lower, ymax = ci_upper),
    inherit.aes = FALSE
  )
}
erglm_data |>
  er_plot(aucss, ae1) |>
  er_plot_add_model(mod) |>
  er_plot_add_quantiles(style = build_quantile_crossbar) |>
  plot()
}
```

er_plot_add_summary *Add a summary annotation layer*

Description

Adds the summary layer: a text/label annotation placed in whichever corner of the base panel is furthest from the observed data, computed from the raw `(exposure, response)` coordinates of the data.

Usage

```
er_plot_add_summary(
  object,
  model = NULL,
  style = NULL,
  keep_strata = NULL,
  conf_level = 0.95,
  summary_args = list(),
  ...
)
```

Arguments

`object`	Partially constructed plot (has S3 class `er_plot`).
`model`	A fitted exposure-response model, or `NULL` (the default). Only needed for builder styles (e.g. `er_style_summary_pvalue()`) that produce model-based summaries; a purely descriptive builder (e.g. `er_style_summary_n()`) ignores it.
`style`	Style used to draw the summary annotation layer. Can either be a string corresponding to one of the registered style labels (e.g., `"pvalue"`, the default), or a builder function used to compute the relevant plot object (see "Styles" below).
`keep_strata`	Logical; whether this layer should use stratification. Defaults to `TRUE` when a stratification variable has been specified, and `FALSE` otherwise.
`conf_level`	Confidence level forwarded to `er_summary()` (used, e.g., for the `conf_low`/`conf_high` columns of its `coefficients` result – see `?er_model_interface`). Defaults to `0.95`. Ignored when `model` is `NULL`.
`summary_args`	A named list of additional arguments forwarded to `er_summary()` when generating summaries.
`...`	Additional named arguments forwarded to the `style` builder function when the plot is built.

Value

The input `object`, with the summary layer added.

Styles

The following pre-defined styles are available for this layer. Please see the documentation for the corresponding builder function to see what customisation options are available:

Label	Builder	Description
"pvalue"	er_style_summary_pvalue()	A formatted p-value from the model's er_summary() result (the
"n"	er_style_summary_n()	Observation counts; model-agnostic, works with model = NULL.
"coefficients"	er_style_summary_coefficients()	One line per model parameter, from er_summary()'s coefficie
"gof"	er_style_summary_gof()	A goodness-of-fit annotation (N/AIC/BIC/R-squared) from er_s

See er_style() for details on how style builder functions are defined for the exposure-response mini-grammar, should a custom style be required.

See Also

er_plot(), er_plot_add_model(), er_plot_add_quantiles(), er_plot_add_data(), er_plot_add_groups(), er_style()

Examples

```
if (requireNamespace("erglm", quietly = TRUE)) {
library(erglm)
mod <- erglm_model(ae1 ~ aucss, erglm_data, family = binomial())
erglm_data |>
  er_plot(aucss, ae1) |>
  er_plot_add_model(mod) |>
  er_plot_add_summary(model = mod) |>
  plot()

# a purely descriptive annotation, with no model at all
erglm_data |>
  er_plot(aucss, ae1) |>
  er_plot_add_summary(style = er_style_summary_n) |>
  plot()
}
```

---

er_plot_build *Build and render an exposure-response plot*

Description

Assembles the layers into ggplot2 objects, applies shared theming and legend deduplication across layers, and composes the final output with patchwork.

Usage

```
er_plot_build(object)
```

Arguments

`object`	Partially constructed plot (has S3 class `er_plot`).

Details

The user does not typically invoke this function directly. Instead, it is called automatically when `plot()` is called.

Value

The input `object`, with `object$plot` (per-layer ggplot2 objects) and `object$output` (the final composed plot) populated.

See Also

`er_plot()`

`er_plot_theme` *Adjust theme/labels for an exposure-response plot*

Description

Set axis/legend labels, plot titles/captions, axis limits, theme objects, discrete and continuous scale objects, formatters, legend key glyph, and relative panel heights. This does not change which variable is mapped to which aesthetic.

Usage

```
er_plot_theme(
  object,
  xlab = NULL,
  ylab = NULL,
  strata_lab = NULL,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  xlim = NULL,
  ylim = NULL,
  theme_base = NULL,
  theme_extra = NULL,
  color_discrete = NULL,
  fill_discrete = NULL,
  color_continuous = NULL,
  fill_continuous = NULL,
  format_p = NULL,
  format_percent = NULL,
  format_number = NULL,
```

```
  draw_key = NULL,
  dodge_width = NULL,
  height_base = NULL,
  height_data = NULL,
  height_group = NULL
)
```

Arguments

object	Partially constructed plot (has S3 class er_plot)
xlab, ylab	Exposure/response axis label (single string).
strata_lab	Stratification legend label (single string). Errors if stratify_by wasn't set in er_plot() – there's no stratification legend to label.
title, subtitle, caption	Plot-level annotation text (single strings), applied via patchwork::plot_annotation() in er_plot_build().
xlim, ylim	Exposure/response axis limits (length-2, increasing numeric vectors, no NA). These are read lazily by every builder at build time, so it doesn't matter whether er_plot_theme() is called before or after the layers that use them. Unlike ggplot2::coord_cartesian()'s xlim/ylim, NA isn't accepted for either endpoint: object$exposure$limits/object$response$limits also drive non-cosmetic computations (e.g. the model curve's prediction grid, quantile-bin boundaries), where a NA bound has no well-defined meaning.
theme_base	A ggplot2 theme object (e.g. ggplot2::theme_minimal()) – the swappable overall visual theme, defaulting to ggplot2::theme_bw().
theme_extra	A ggplot2 theme object (e.g. from ggplot2::theme()) with additional theme tweaks layered on top of theme_base. See "Details" for its default and replacement semantics.
color_discrete, fill_discrete	A discrete ggplot2 scale object (e.g. ggplot2::scale_color_brewer(), ggplot2::scale_fill_viridi applied to every plot whose colour/fill aesthetic is mapped to the stratification variable – see "Details".
color_continuous, fill_continuous	A continuous ggplot2 scale object (e.g. ggplot2::scale_color_viridis_c(), ggplot2::scale_fill_gradient()), applied to every plot whose colour/fill aesthetic is mapped to something continuous other than the stratification variable – see "Details".
format_p, format_percent, format_number	Formatter functions (typically from scales::label_*()). Used by the summary/quantile layers to format p-values/rates/means for display. Default to scales::label_pvalue(accu = .001, add_p = TRUE), scales::label_percent(accuracy = 1), and scales::label_number(accur = 0.01) respectively.
draw_key	A key-glyph function (e.g. ggplot2::draw_key_point()), passed as every geom's key_glyph argument. Defaults to ggplot2::draw_key_rect().
dodge_width	Spacing between adjacent strata's horizontal offset in the quantile layer, as a fraction of the exposure range. A single positive number; see "Details".

height_base, height_data, height_group
: Relative panel heights (single positive numbers), for the base plot, data-layer panel(s), and group-layer panel(s) respectively. Default to 6, 2, and 3. Supplying only one leaves the other two unchanged.

Details

dodge_width is a stratification-layout setting used by er_style_quantile_errorbar()/er_style_quantile_pointrange (and their _vlines variants) to separate strata horizontally within each quantile bin. It belongs in er_plot_theme() because dodging is about stratification layout, not an individual builder's visual style. Defaults to 0.015 (set in er_plot()).

Every argument defaults to NULL, meaning "leave whatever was set before unchanged". This allows repeated calls to er_plot_theme() to update only the supplied fields, like ggplot2::theme(). There is no implicit way to reset a field to the er_plot() default.

color_discrete/fill_discrete apply only when a layer's colour/ fill aesthetic is mapped to stratification. Their continuous counterparts, color_continuous/fill_continuous, apply only when the aesthetic is mapped to a continuous quantity such as density or a continuous/count response value. If a custom builder adds its own scale, supplying one of these four will add a second scale and let ggplot2 choose the later one.

theme_extra defaults to a panel border plus legend.position = "bottom". Supplying a new value fully replaces this default rather than merging with it, so re-include the border/legend-position settings too if you want to keep them alongside your own additions.

Value

The input object, with the requested theme fields updated.

See Also

er_plot(), er_style()

Examples

```
if (requireNamespace("erglm", quietly = TRUE)) {
  library(erglm)
  mod <- erglm_model(ae1 ~ aucss, erglm_data, family = binomial())

  # axis labels, a title, and a swapped-in theme
  erglm_data |>
    er_plot(aucss, ae1) |>
    er_plot_add_model(mod) |>
    er_plot_theme(
      xlab = "AUC at steady state",
      ylab = "P(adverse event)",
      title = "Exposure-response for adverse events",
      theme_base = ggplot2::theme_minimal()
    ) |>
    plot()

  # repeated calls accumulate: this only touches xlim/ylim, leaving
```

```
  # the labels/title/theme set above unchanged
  erglm_data |>
    er_plot(aucss, ae1) |>
    er_plot_add_model(mod) |>
    er_plot_theme(xlab = "AUC at steady state") |>
    er_plot_theme(xlim = c(0, 3000)) |>
    plot()

  # widening the stratum-dodge spacing in a stratified quantile layer
  mod2 <- erglm_model(ae1 ~ aucss + sex, erglm_data, family = binomial())
  erglm_data |>
    er_plot(aucss, ae1, stratify_by = sex) |>
    er_plot_add_model(mod2) |>
    er_plot_add_quantiles() |>
    er_plot_theme(dodge_width = 0.15, strata_lab = "Sex") |>
    plot()
}
```

`er_style` *Builder functions for exposure-response plots*

Description

Documents the shared `function(data, config, stratify, exposure, response, strata, theme, ...)` signature every `er_style_*()` builder implements, including how to write a custom one.

Details

This page documents the shared interface all `er_style_*()` builders implement. The builders themselves are documented on their own family-specific pages, one per layer:

- `er_style_model()` – the `model` layer (`er_plot_add_model()`)
- `er_style_summary()` – the `summary` layer (`er_plot_add_summary()`)
- `er_style_quantile()` – the `quantile` layer (`er_plot_add_quantiles()`)
- `er_style_data()` – the `data` layer (`er_plot_add_data()`)
- `er_style_group()` – the `group` layer (`er_plot_add_groups()`)

`er_vpc()`/`er_tte()` have their own, separate builder interfaces – `er_style_vpc()`/`er_style_tte()` – since neither shares this signature exactly (`er_vpc_*()` builders have no `stratify`/`strata` pair; `er_tte_*()` builders take `time` instead of `exposure`/ `response`).

Arguments are standardised to allow users to write their own custom builders as needed.

Value

A geom, or a list of geoms. More precisely, a list of objects that can be added to a ggplot2 plot. The expectation is that these objects will be added to a partially constructed plot which, at a minimum, already has the base theme applied. For "model", "summary", "quantile", and "overlay", the pieces will be added to a plot that already has a coord that sets the axis limits (the base plot). For the "data" (panel-based, e.g. `er_style_data_boxjitter()`) and "group" plots, the plot object does not yet have a coord. The expectation, however, is that the builder will supply an x-axis limit that is consistent with the base plot. That is, since all layer plots use the exposure variable for the x-axis, they should use the values stored in `exposure$limits` to set the x-axis limits.

Arguments

Every `er_style_*()` builder receives:

- `data` – The original data frame
- `config` – Configuration for the specific plot
- `stratify` – Logical indicating whether to stratify
- `exposure` – Exposure variable
- `response` – Response variable
- `strata` – Stratification variable
- `theme` – Theme components
- `...` – Additional named arguments forwarded from the corresponding `er_plot_add_*()` call's own `...`; see "Passing extra arguments to a builder" below.

Writing your own builder

Every `er_style_*()` function above shares the signature documented in the "Arguments" section above, and that signature is a public part of the API, not an implementation detail: any function `function(data, config, stratify, exposure, response, strata, theme, ...)` that returns a geom or list of geoms can stand in for a built-in builder. This is the officially supported way to draw a layer differently from any of the built-in `style` options – e.g. a 2D density instead of a scatter for the data overlay, per-panel histograms instead of jittered points for the panel-based data layer, or a `geom_crossbar()` instead of a `geom_errorbar()`/`geom_pointrange()` for the quantile summary. (`er_style_quantile_pointrange()` started life as exactly this kind of custom builder – it was promoted to a built-in option once it proved to be a natural, low-risk alternative to `er_style_quantile_errorbar()`, with no new config requirements.)

Each `er_plot_add_*()` function takes a `style` argument that defaults to one built-in `er_style_*()` function and can be set to any other – built-in or custom – matching the standard signature: a custom builder can be plugged in without forking the package or reaching into the plot object's internal state. For the data layer specifically, `style` also has to declare which *structural* family it belongs to – a single call merged into the main panel, or one or more panels stacked below the base plot – via `er_style_tag()`, since `er_plot_add_data()` reads that tag off `style` to decide how to assemble the layer; the other four layers have only one structural call site, so no such tagging is needed there. See the `@examples` on `er_plot_add_model()`, `er_plot_add_quantiles()`, and `er_plot_add_data()` for worked custom builders (a dashed model curve, a quantile crossbar, and a data-overlay density, respectively). An overlay-layout data builder can additionally declare, via

the same er_style_tag() call's draw_order argument, whether its geoms are drawn before or after the model/summary/quantile layers when they share the main panel – relevant for a builder whose geoms cover the whole panel (e.g. er_style_data_hex()), which would otherwise bury those layers by drawing on top of them; see er_style_data() for the full explanation.

A custom builder receives the same pre-computed config a built-in builder would have received for that layer (e.g. config$predictions for model, config$summary for quantile) – it does not need to recompute anything erplots already derived from data/exposure/ response/strata; it only needs to turn that config into ggplot2 layers.

A custom builder can optionally self-declare which layer it's meant for via er_style_tag(builder, layer = ...) (one of "plot_model", "plot_summary", "plot_quantile", "plot_data", "plot_group"). Every er_plot_add_*() function checks a builder's layer tag, if it has one, against the layer it was actually passed to, erroring immediately if they disagree – e.g. passing a builder tagged layer = "plot_quantile" to er_plot_add_data() errors rather than calling the builder with a config shape it wasn't written for. This tag is entirely optional (unlike layout, which is mandatory for a data-layer builder specifically) – an untagged custom builder is simply never checked, so existing custom builders keep working unchanged. All built-in builders carry this tag.

All of the builders above feed a **singleton** layer: model, summary, quantile, data, and overlay each occupy a single slot in the plot's internal state, so calling the corresponding er_plot_add_*() function again overwrites that slot rather than combining builders. group (er_style_group_boxplot()/ er_style_group_violin()) is the one **additive** exception – each call to er_plot_add_groups() adds another named entry rather than replacing the previous one. See er_plot()'s "Layers are either singleton or additive" section for the full discussion.

The data slot's default, er_style_data_overlay(), needs no color_role tag: its colour aesthetic (when stratified) is always strata, since the response is already shown via y-position, so it shares the base plot's own strata legend directly. config$color_role matters for the "panel"-layout family instead, where it's "strata" for a binary response (as used by the built-in er_style_data_boxjitter(), whose colour aesthetic still means strata) or "response" for a continuous/count response, where the colour channel is already spoken for by the response value itself – there's no built-in "panel"-layout builder for that case today, but a custom builder tagged er_style_tag(builder, layout = "panel") can still opt into it; see er_plot_add_data() for the user-facing version of this rule.

Passing extra arguments to a builder

Every er_plot_add_*() function (er_plot_add_model(), er_plot_add_summary(), er_plot_add_quantiles(), er_plot_add_data(), er_plot_add_groups()) takes its own ..., which is forwarded unchanged to style when it's actually called at build time. Extra arguments must be named, since they're appended positionally after the seven standard arguments; an unnamed one errors immediately rather than silently binding to the wrong parameter. This is how a builder that needs a piece of information beyond what config already carries – something genuinely per-call rather than a fixed part of the layer's configuration – can accept it without a bespoke argument on every er_plot_add_*() function. The motivating built-in example is er_style_model_spaghetti(), which calls er_simulate() and, for models (like erglm's) that auto-select and report a seed when none is supplied, would otherwise always trigger that message:

```
erglm_data |>
  er_plot(aucss, ae1) |>
  er_plot_add_model(mod, style = er_style_model_spaghetti, seed = 9626) |>
  plot()
```

A builder that doesn't need any extra arguments simply declares `...` and ignores it – every built-in builder does exactly this except `er_style_model_spaghetti()`. A custom builder can read whichever named arguments it recognizes out of its own `...` (e.g. via `rlang::list2(...)`) and ignore the rest; unrecognised extra arguments are never an error at the builder itself, only at the `er_plot_add_*()` call site if they weren't named.

See Also

`er_style_model()`, `er_style_summary()`, `er_style_quantile()`, `er_style_data()`, `er_style_group()`, `er_style_tag()`, `er_style_vpc()`, `er_style_tte()`

`er_style_data` *Data layer builders for exposure-response plots*

Description

Builder functions for the `data` layer (`er_plot_add_data()`), drawing raw observations either as an overlay on the main panel or as separate boxplot/jitter panels.

Usage

```
er_style_data_boxjitter(
  data,
  config,
  stratify,
  exposure,
  response,
  strata,
  theme,
  box_width = 0.6,
  box_alpha = 0.4,
  show_outliers = FALSE,
  jitter_height = NULL,
  jitter_size = 1,
  jitter_alpha = 0.6,
  ...
)

er_style_data_overlay(
  data,
  config,
  stratify,
  exposure,
  response,
  strata,
  theme,
  jitter_height = NULL,
```

```
  alpha = 0.4,
  point_size = 1,
  ...
)

er_style_data_hex(
  data,
  config,
  stratify,
  exposure,
  response,
  strata,
  theme,
  bins = 30,
  alpha = 0.85,
  ...
)
```

Arguments

data	The original data frame.
config	Configuration for the specific plot.
stratify	Logical: whether to stratify.
exposure	Exposure variable.
response	Response variable.
strata	Stratification variable.
theme	Theme components.
box_width	Width of er_style_data_boxjitter()'s boxplot. Defaults to 0.6.
box_alpha	Transparency of er_style_data_boxjitter()'s boxplot fill. Defaults to 0.4.
show_outliers	Logical: whether er_style_data_boxjitter() draws outlier points. Defaults to FALSE, since its raw points are already shown via the jitter layer.
jitter_height	Vertical jitter applied to raw points. Defaults to NULL: er_style_data_boxjitter() resolves this to 0.3 when stratified or 0.15 otherwise; er_style_data_overlay() resolves it to 0.015 for a binary response (whose y-values would otherwise overplot into two solid lines) or 0 otherwise.
jitter_size	Point size for er_style_data_boxjitter()'s jittered points. Defaults to 1.
jitter_alpha	Transparency of er_style_data_boxjitter()'s jittered points. Defaults to 0.6.
...	Additional named arguments forwarded from er_plot_add_data()'s own .... er_style_data_overlay()/er_style_data_boxjitter() read a seed from here (via config$seed, NULL when not supplied) and pass it to ggplot2::position_jitter(), letting a caller make the jitter reproducible across repeated plot() calls on the same object; with no seed, each render draws a fresh jitter, as for any other jittered geom.

alpha	Point transparency for er_style_data_overlay() (defaults to 0.4); fill transparency for er_style_data_hex() (defaults to 0.85).
point_size	Point size for er_style_data_overlay(). Defaults to 1.
bins	Number of hex bins for er_style_data_hex(). Defaults to 30.

Details

See er_style() for the shared builder interface these functions implement.

Value

A geom, or a list of geoms; see er_style().

Choosing a builder

All three builders draw raw observations, but differ in structural family (see er_style_tag()'s layout) and which response types they support:

- er_style_data_overlay() (the default) – raw points, overlaid directly on the main panel (layout = "overlay"). Any response type.
- er_style_data_hex() – a 2D hexbin density overlay on the main panel instead of individual points (layout = "overlay"), useful when there are too many points for er_style_data_overlay() to stay legible. Requires the hexbin package.
- er_style_data_boxjitter() – a boxplot + jitter panel stacked below/above the main panel instead of an overlay (layout = "panel"). Binary-response only.

All built-in data builders are also tagged layer = "plot_data", so er_plot_add_data() errors if given a builder tagged for another layer.

Hex fill and draw order

er_style_data_hex() defaults to a light-grey-to-navy ("grey90" to "#132B43") fill gradient, so a cell's fill fades toward the panel background as its count approaches zero rather than starting at ggplot2's own default mid-intensity blue. Override it with er_plot_theme(fill_continuous = ...).

Because its geoms cover the whole panel, er_style_data_hex() is tagged er_style_tag(fn, draw_order = "background") (see er_style_tag()), so it's drawn before the model/summary/quantile layers rather than on top of them; its default alpha = 0.85 gives those layers a little extra visibility through even a densely populated hex cell.

See Also

er_style(), er_style_tag()

Examples

```
if (requireNamespace("erglm", quietly = TRUE)) {
  library(erglm)
  mod2 <- erglm_model(ae2 ~ aucss + sex, erglm_data, family = binomial())

  # er_style_data_overlay(): the default, raw points on the main panel
  erglm_data |>
    er_plot(aucss, ae2, stratify_by = sex) |>
    er_plot_add_model(mod2) |>
    er_plot_add_data(style = er_style_data_overlay) |>
    plot()

  # er_style_data_boxjitter(): binary-response only, boxplot + jitter
  # panels above/below the main panel instead of an overlay
  erglm_data |>
    er_plot(aucss, ae2, stratify_by = sex) |>
    er_plot_add_model(mod2) |>
    er_plot_add_data(style = er_style_data_boxjitter) |>
    plot()

  # overriding a builder's own visual defaults, e.g. larger/more
  # opaque points and a wider jitter
  erglm_data |>
    er_plot(aucss, ae2, stratify_by = sex) |>
    er_plot_add_model(mod2) |>
    er_plot_add_data(
      style = er_style_data_overlay,
      jitter_height = 0.1,
      alpha = 0.7,
      size = 2
    ) |>
    plot()
}
```

`er_style_group` *Group panel builders for exposure-response plots*

Description

Builder functions for the `group` layer (`er_plot_add_groups()`), drawing the exposure distribution for a grouping variable as a boxplot, violin, or histogram panel.

Usage

```
er_style_group_boxplot(
  data,
  config,
  stratify,
```

```
  exposure,
  response,
  strata,
  theme,
  alpha = 0.5,
  show_outliers = TRUE,
  ...
)

er_style_group_histogram(
  data,
  config,
  stratify,
  exposure,
  response,
  strata,
  theme,
  bins = 30,
  alpha = NULL,
  ...
)

er_style_group_violin(
  data,
  config,
  stratify,
  exposure,
  response,
  strata,
  theme,
  alpha = 0.5,
  quantiles = NULL,
  quantile_linetype = "solid",
  ...
)

er_style_group_linerange(
  data,
  config,
  stratify,
  exposure,
  response,
  strata,
  theme,
  scale_factor = 1,
  inner_range = c(0.25, 0.75),
  outer_range = c(0.05, 0.95),
  dot_alpha = 1,
```

```
  inner_alpha = 0.8,
  outer_alpha = 0.4,
  ...
)

er_style_group_boxjitter(
  data,
  config,
  stratify,
  exposure,
  response,
  strata,
  theme,
  alpha = 0.5,
  jitter_height = 0.15,
  jitter_size = 1,
  jitter_alpha = 0.6,
  ...
)

er_style_group_violinjitter(
  data,
  config,
  stratify,
  exposure,
  response,
  strata,
  theme,
  alpha = 0.5,
  quantiles = NULL,
  quantile_linetype = "solid",
  jitter_height = 0.15,
  jitter_size = 1,
  jitter_alpha = 0.6,
  ...
)
```

Arguments

data	The original data frame.
config	Configuration for the specific plot.
stratify	Logical: whether to stratify.
exposure	Exposure variable.
response	Response variable.
strata	Stratification variable.
theme	Theme components.

alpha
: Transparency of the geom. Defaults to `0.5` for `er_style_group_boxplot()`/`er_style_group_violin()`/`er_style_group_boxjitter()`/`er_style_group_violinjitter()`; `er_style_group_histogram()` defaults to `NULL`, which resolves to `0.5` when stratified or `0.8` otherwise.

show_outliers
: Logical: whether `er_style_group_boxplot()` draws the boxplot's own outlier points. Defaults to `TRUE`; `er_style_group_boxjitter()` sets this to `FALSE` when it wraps this builder, since its own jittered points already show every raw value, outliers included.

...
: Additional named arguments forwarded from `er_plot_add_groups()`'s own `....` `er_style_group_boxjitter()`/`er_style_group_violinjitter()` read a `seed` from here (`NULL` when not supplied) and use it to scope (via `withr::with_seed()`) the vertical jitter draw, letting a caller make the jitter reproducible across repeated `plot()` calls on the same object – the same opt-in-only mechanism `er_style_data_overlay()`/`er_style_data_boxjitter()` use for the data layer (see `er_style_data()`); with no `seed`, each render draws a fresh jitter.

bins
: Number of histogram bins for `er_style_group_histogram()`. Defaults to `30`.

quantiles, quantile_linetype
: Violin quantile positions and linetype for `er_style_group_violin()`. Default to `NULL` (no quantile lines drawn) and `"solid"` respectively.

scale_factor
: Overall size multiplier for `er_style_group_linerange()`'s dot and lines. Defaults to `1`.

inner_range, outer_range
: Quantile probabilities (length 2) for `er_style_group_linerange()`'s thick and thin lines. Default to `c(0.25, 0.75)` and `c(0.05, 0.95)` respectively.

dot_alpha, inner_alpha, outer_alpha
: Per-part transparency for `er_style_group_linerange()`'s dot, inner line, and outer line. Default to `1`, `0.8`, and `0.4` respectively.

jitter_height, jitter_size, jitter_alpha
: Vertical jitter, point size, and transparency for `er_style_group_boxjitter()`/`er_style_group_violin` overlaid points. Default to `0.15`, `1`, and `0.6` respectively.

Details

See `er_style()` for the shared builder interface these functions implement.

Value

A geom, or a list of geoms; see `er_style()`.

Choosing a builder

All six builders show the same thing – a grouping variable's exposure distribution – as one of a few visual idioms:

- `er_style_group_boxplot()` (the default) – a boxplot, group levels on the y-axis.
- `er_style_group_violin()` – a violin instead of a boxplot, same axis layout.
- `er_style_group_histogram()` – group levels on facet strips instead, freeing the y-axis for counts.

- `er_style_group_linerange()` – a median dot flanked by an inner-range and outer-range line, instead of a full boxplot/violin shape; same y-axis layout as the boxplot/violin builders.
- `er_style_group_boxjitter()` / `er_style_group_violinjitter()` – thin wrappers around `er_style_group_boxplot()`/ `er_style_group_violin()` that additionally overlay jittered raw exposure values (see "Jittered variants" below).

All built-in group builders are tagged `layer = "plot_group"`, so `er_plot_add_groups()` errors if given one tagged for another layer.

Axis and facet layout

`er_style_group_boxplot()` and `er_style_group_violin()` put group levels on the y-axis; `er_style_group_histogram` puts them on facet strips and frees the y-axis for counts; `er_style_group_linerange()` also puts group levels on the y-axis, summarising each level's exposure distribution as a median dot flanked by an inner-range and outer-range line rather than a full boxplot/violin shape.

Jittered variants

`er_style_group_boxjitter()`/`er_style_group_violinjitter()` are thin wrappers around `er_style_group_boxplot()` that additionally overlay jittered raw exposure values (vertical jitter only – exposure position on the x-axis is never perturbed), the same idea `er_style_data_boxjitter()` applies to the data layer.

See Also

`er_style()`

Examples

```
if (requireNamespace("erglm", quietly = TRUE)) {
  library(erglm)
  mod <- erglm_model(ae1 ~ aucss, erglm_data, family = binomial())

  # er_style_group_boxplot(): the default
  erglm_data |>
    er_plot(aucss, ae1) |>
    er_plot_add_model(mod) |>
    er_plot_add_groups(aucss, style = er_style_group_boxplot) |>
    plot()

  # er_style_group_violin(): a violin instead of a boxplot
  erglm_data |>
    er_plot(aucss, ae1) |>
    er_plot_add_model(mod) |>
    er_plot_add_groups(aucss, style = er_style_group_violin) |>
    plot()

  # er_style_group_histogram(): group levels on facet strips, with
  # the y-axis freed for counts
  erglm_data |>
    er_plot(aucss, ae1) |>
    er_plot_add_model(mod) |>
```

```
  er_plot_add_groups(aucss, style = er_style_group_histogram) |>
  plot()

# er_style_group_linerange(): median dot + inner/outer range lines,
# instead of a full boxplot/violin shape
erglm_data |>
  er_plot(aucss, ae1) |>
  er_plot_add_model(mod) |>
  er_plot_add_groups(aucss, style = er_style_group_linerange) |>
  plot()

# er_style_group_boxjitter(): the boxplot, with jittered raw
# exposure values overlaid on top
erglm_data |>
  er_plot(aucss, ae1) |>
  er_plot_add_model(mod) |>
  er_plot_add_groups(aucss, style = er_style_group_boxjitter) |>
  plot()

# er_style_group_violinjitter(): the violin, with jittered raw
# exposure values overlaid on top
erglm_data |>
  er_plot(aucss, ae1) |>
  er_plot_add_model(mod) |>
  er_plot_add_groups(aucss, style = er_style_group_violinjitter) |>
  plot()
}
```

`er_style_labels` *List builders registered for string-based* `style` *dispatch*

Description

Style builder functions can be tagged with a `"label"` attribute used to register a convenient short name for that style: `er_style_labels()` lists the currently registered styles, optionally filtered to a single `layer`.

Usage

```
er_style_labels(layer = NULL)
```

Arguments

`layer`	A single layer name (e.g. `"tte_summary"`), or `NULL` (the default) to list every registered label across all layers.

Value

A tibble with one row per registered `(layer, label)` pair and columns `layer`, `label`, and `style` (the builder function itself).

Examples

```
er_style_labels("tte_summary")
```

---

er_style_model *Model curve builders for exposure-response plots*

Description

Builder functions for the `model` layer (`er_plot_add_model()`), drawing the fitted exposure-response curve as a ribbon-and-line, a line alone, or a spaghetti plot of simulated draws.

Usage

```
er_style_model_ribbonline(
  data,
  config,
  stratify,
  exposure,
  response,
  strata,
  theme,
  ribbon_fill = "grey40",
  ribbon_alpha = 0.25,
  ribbon_edges = FALSE,
  linewidth = 1,
  ...
)

er_style_model_line(
  data,
  config,
  stratify,
  exposure,
  response,
  strata,
  theme,
  linewidth = 1,
  ...
)

er_style_model_spaghetti(
  data,
  config,
  stratify,
  exposure,
  response,
```

```
  strata,
  theme,
  alpha = NULL,
  linewidth = 1,
  nsim = 100L,
  ...
)
```

Arguments

data	The original data frame
config	Configuration for the specific plot
stratify	Logical indicating whether to stratify
exposure	Exposure variable
response	Response variable
strata	Stratification variable
theme	Theme components
ribbon_fill	Fill colour for er_style_model_ribbonline()'s ribbon. Only takes effect when the layer is unstratified – a stratified ribbon already maps fill to the strata variable, so this argument is ignored in that case. Default "grey40".
ribbon_alpha	Transparency of er_style_model_ribbonline()'s ribbon (0-1), stratified or not. Default 0.25.
ribbon_edges	Whether er_style_model_ribbonline() additionally draws a dashed ggplot2::geom_path() along the ribbon's own ci_lower/ci_upper bounds, on top of the shaded ribbon fill. Default FALSE (ribbon fill only).
linewidth	Width of the fitted curve's line, for all three model builders (er_style_model_ribbonline()/_line()'s single curve, er_style_model_spaghetti()'s mean curve drawn on top of the spaghetti draws). Default 1.
...	Additional named arguments forwarded from er_plot_add_model()'s own ...; see er_style()'s "Passing extra arguments to a builder" section. er_style_model_spaghetti() reads a seed from here (falling back to config$seed – currently always NULL for the model layer – when none is supplied) to pass to er_simulate(), letting a caller override erglm's auto-selected seed.
alpha	Transparency of er_style_model_spaghetti()'s individual simulated draws (0-1). Defaults to NULL, which uses 0.1 unstratified and 0.25 stratified. An explicit value overrides this for both cases uniformly.
nsim	Number of simulated draws for er_style_model_spaghetti(), passed to er_simulate(). Default 100L.

Details

See er_style() for the shared builder interface these functions implement, including how to write a custom builder of your own.

Value

A geom, or a list of geoms; see `er_style()`.

Choosing a builder

All three builders draw the same fitted exposure-response curve; which one to reach for is a choice of how to convey uncertainty around it:

- `er_style_model_ribbonline()` (the default) – the curve plus a shaded confidence ribbon.
- `er_style_model_line()` – the curve alone, omitting the ribbon.
- `er_style_model_spaghetti()` – the curve overlaid on individual simulated draws instead of a ribbon, for models that implement `er_simulate()`.

All three are tagged `er_style_tag(fn, layer = "plot_model")`, so `er_plot_add_model()` errors informatively if handed one of these tagged for a different layer entirely (e.g. `"summary"`, meant for `er_plot_add_summary()`). Each also carries a registered short-string label – `"ribbonline"`/`"line"`/`"spaghetti"` respectively – so `er_plot_add_model()`'s `style` argument can take that string instead of the function itself (see `er_style_labels()`).

See Also

`er_style()`

Examples

```
if (requireNamespace("erglm", quietly = TRUE)) {
  library(erglm)
  mod <- erglm_model(ae1 ~ aucss, erglm_data, family = binomial())

  # er_style_model_ribbonline(): ribbon + line, the default
  erglm_data |>
    er_plot(aucss, ae1) |>
    er_plot_add_model(mod, style = er_style_model_ribbonline) |>
    plot()

  # er_style_model_line(): line only, no ribbon
  erglm_data |>
    er_plot(aucss, ae1) |>
    er_plot_add_model(mod, style = er_style_model_line) |>
    plot()

  # er_style_model_spaghetti(): simulated draws instead of a ribbon;
  # `seed` is forwarded to `er_simulate()` via `...`
  erglm_data |>
    er_plot(aucss, ae1) |>
    er_plot_add_model(mod, style = er_style_model_spaghetti, seed = 4821) |>
    plot()

  # overriding a builder's own visual defaults: a thicker, less
  # saturated ribbon with its bounds outlined, and fewer/fainter
  # spaghetti draws
```

```
  erglm_data |>
    er_plot(aucss, ae1) |>
    er_plot_add_model(
      mod,
      style = er_style_model_ribbonline,
      ribbon_fill = "steelblue",
      ribbon_alpha = 0.15,
      ribbon_edges = TRUE,
      linewidth = 1.5
    ) |>
    plot()

  erglm_data |>
    er_plot(aucss, ae1) |>
    er_plot_add_model(
      mod,
      style = er_style_model_spaghetti,
      seed = 4821,
      nsim = 40L,
      alpha = 0.05
    ) |>
    plot()
}
```

---

er_style_quantile *Quantile summary builders for exposure-response plots*

Description

Builder functions for the `quantile` layer (`er_plot_add_quantiles()`), drawing a point/interval summary per exposure quantile bin as an error bar or a pointrange, optionally with bin-boundary vlines.

Usage

```
er_style_quantile_errorbar(
  data,
  config,
  stratify,
  exposure,
  response,
  strata,
  theme,
  point_size = 2,
  errorbar_width = 0.025,
  label_size = 3,
  ...
)
```

```
er_style_quantile_errorbar_vlines(
  data,
  config,
  stratify,
  exposure,
  response,
  strata,
  theme,
  point_size = 2,
  errorbar_width = 0.025,
  label_size = 3,
  vline_colour = "grey50",
  vline_linetype = "dotted",
  vline_labels = FALSE,
  vline_label_position = c("auto", "top", "bottom"),
  vline_label_size = 3,
  vline_label_colour = NULL,
  vline_label_fill = NULL,
  vline_label_inset = 0.05,
  vline_label_digits = 0,
  ...
)

er_style_quantile_pointrange(
  data,
  config,
  stratify,
  exposure,
  response,
  strata,
  theme,
  label_size = 3,
  pointrange_size = NULL,
  pointrange_linewidth = NULL,
  ...
)

er_style_quantile_pointrange_vlines(
  data,
  config,
  stratify,
  exposure,
  response,
  strata,
  theme,
  label_size = 3,
  pointrange_size = NULL,
```

```
  pointrange_linewidth = NULL,
  vline_colour = "grey50",
  vline_linetype = "dotted",
  vline_labels = FALSE,
  vline_label_position = c("auto", "top", "bottom"),
  vline_label_size = 3,
  vline_label_colour = NULL,
  vline_label_fill = NULL,
  vline_label_inset = 0.05,
  vline_label_digits = 0,
  ...
)
```

Arguments

`data`	The original data frame.
`config`	Configuration for the specific plot.
`stratify`	Logical: whether to stratify.
`exposure`	Exposure variable.
`response`	Response variable.
`strata`	Stratification variable.
`theme`	Theme components.
`point_size`	Point size for `er_style_quantile_errorbar()`. Defaults to `2`.
`errorbar_width`	Width of `er_style_quantile_errorbar()`'s error bars. Defaults to `0.025`.
`label_size`	Text size for the per-bin value label. Defaults to `3`.
`...`	Additional named arguments forwarded from `er_plot_add_quantiles()`'s own `....`
`vline_colour, vline_linetype`	Colour and linetype of quantile-bin boundary lines. Default to `"grey50"` and `"dotted"` respectively.
`vline_labels`	Logical: whether the `_vlines` builders also label each bin boundary with its exposure value. Defaults to `FALSE`.
`vline_label_position`	One of `"auto"` (the default), `"top"`, or `"bottom"` – vertical placement of `vline_labels`.
`vline_label_size, vline_label_colour, vline_label_fill`	Size, text colour, and background fill for `vline_labels`. `vline_label_size` defaults to `3`; `vline_label_colour`/`vline_label_fill` default to `NULL` (`ggplot2::geom_label()`'s own defaults).
`vline_label_inset`	Fraction of the response range `vline_labels` are inset from the panel edge. Defaults to `0.05`.
`vline_label_digits`	Number of decimal places `vline_labels` are rounded to. Defaults to `0`.
`pointrange_size, pointrange_linewidth`	Size and linewidth for `ggplot2::geom_pointrange()`. Default to `NULL`, which leaves `geom_pointrange()`'s own built-in size/linewidth defaults in effect.

Details

See er_style() for the shared builder interface these functions implement, including how to write a custom builder of your own.

Value

A geom, or a list of geoms; see er_style().

Choosing a builder

All four builders summarise the same per-bin point/interval; which one to reach for is a choice of visual idiom, independent of response type:

- er_style_quantile_errorbar() (the default) – a point with an error bar.
- er_style_quantile_pointrange() – a point with a range line (ggplot2::geom_pointrange()) instead of an error bar.
- er_style_quantile_errorbar_vlines() / er_style_quantile_pointrange_vlines() – the same two idioms, plus a dotted vertical line at every quantile-bin boundary (see "Boundary lines" below).

All built-in quantile builders are tagged er_style_tag(fn, layer = "plot_quantile"), so er_plot_add_quantiles() errors informatively if handed a builder tagged for a different layer.

Boundary lines

The _vlines variants add a line at every quantile-bin boundary – including the two outer boundaries at the minimum non-placebo exposure and the overall maximum exposure, not just the boundaries shared between two adjacent bins – so a reader can see every bin edge from the plot alone. A boundary whose exposure value falls outside a narrowed er_plot_theme() xlim is dropped (with a warning), the same way a quantile summary marker is.

Boundary labels

The _vlines variants can also label each boundary with its exposure value (vline_labels = TRUE, off by default). Labels are drawn with ggplot2::geom_label() (an opaque background, since a label sits directly on a vline spanning the full panel height) along either the top or bottom edge of the panel. vline_label_position = "auto" (the default) picks whichever vertical half doesn't contain the corner a summary annotation (er_plot_add_summary()) would place itself in – based on the same raw-data corner-crowdedness calculation the summary layer itself uses – so the two don't collide; this works whether or not a summary layer is actually present, since both layers compute the same deterministic quantity independently. Override with "top"/"bottom" to place labels manually instead.

Stratified dodging

When stratified, all four builders horizontally dodge each quantile bin's points/bars/labels apart by er_plot_theme()'s dodge_width (a fraction of the exposure range, default 0.05) – a cross-layer, stratification-wide setting controlled via er_plot_theme() rather than a per-builder argument here, since it's about how stratification lays out a dodged layer, not one builder's own visual style.

See Also

er_style()

Examples

```
if (requireNamespace("erglm", quietly = TRUE)) {
  library(erglm)
  mod <- erglm_model(ae1 ~ aucss, erglm_data, family = binomial())

  # er_style_quantile_errorbar(): point + error bar, the default
  erglm_data |>
    er_plot(aucss, ae1) |>
    er_plot_add_model(mod) |>
    er_plot_add_quantiles(style = er_style_quantile_errorbar) |>
    plot()

  # er_style_quantile_pointrange(): a pointrange instead
  erglm_data |>
    er_plot(aucss, ae1) |>
    er_plot_add_model(mod) |>
    er_plot_add_quantiles(style = er_style_quantile_pointrange) |>
    plot()

  # er_style_quantile_errorbar_vlines(): the default, plus dotted
  # lines marking every quantile-bin boundary, including the outer
  # edges at the minimum non-placebo and maximum exposure
  erglm_data |>
    er_plot(aucss, ae1) |>
    er_plot_add_model(mod) |>
    er_plot_add_quantiles(style = er_style_quantile_errorbar_vlines) |>
    plot()

  # Customize the quantile builder's appearance.
  erglm_data |>
    er_plot(aucss, ae1) |>
    er_plot_add_model(mod) |>
    er_plot_add_quantiles(
      style = er_style_quantile_errorbar,
      point_size = 4,
      errorbar_width = 0.08,
      label_size = 4
    ) |>
    plot()

  erglm_data |>
    er_plot(aucss, ae1) |>
    er_plot_add_model(mod) |>
    er_plot_add_quantiles(
      style = er_style_quantile_pointrange,
      label_size = 4,
      pointrange_size = 2,
      pointrange_linewidth = 1.2
```

```
  ) |>
  plot()

# widening the stratum-dodge spacing via er_plot_theme()
mod2 <- erglm_model(ae1 ~ aucss + sex, erglm_data, family = binomial())
erglm_data |>
  er_plot(aucss, ae1, stratify_by = sex) |>
  er_plot_add_model(mod2) |>
  er_plot_add_quantiles(style = er_style_quantile_errorbar) |>
  er_plot_theme(dodge_width = 0.15) |>
  plot()

# labeling every quantile-bin boundary (including the outer edges)
# with its exposure value, placed automatically to avoid the
# summary annotation
erglm_data |>
  er_plot(aucss, ae1) |>
  er_plot_add_model(mod) |>
  er_plot_add_summary(mod) |>
 er_plot_add_quantiles(style = er_style_quantile_errorbar_vlines, vline_labels = TRUE) |>
  plot()

# er_style_quantile_pointrange_vlines(): the pointrange equivalent of
# er_style_quantile_errorbar_vlines()
erglm_data |>
  er_plot(aucss, ae1) |>
  er_plot_add_model(mod) |>
  er_plot_add_quantiles(style = er_style_quantile_pointrange_vlines) |>
  plot()
}
```

`er_style_summary` *Summary annotation builders for exposure-response plots*

Description

Builder functions for the `summary` layer (`er_plot_add_summary()`), drawing a text/label annotation from a model's p-value, coefficients, goodness-of-fit statistics, or observation counts.

Usage

```
er_style_summary_pvalue(
  data,
  config,
  stratify,
  exposure,
  response,
  strata,
```

```
  theme,
  inset = 0.05,
  label_size = NULL,
  label_colour = NULL,
  label_fill = NULL,
  ...
)

er_style_summary_n(
  data,
  config,
  stratify,
  exposure,
  response,
  strata,
  theme,
  inset = 0.05,
  label_size = NULL,
  label_colour = NULL,
  label_fill = NULL,
  ...
)

er_style_summary_coefficients(
  data,
  config,
  stratify,
  exposure,
  response,
  strata,
  theme,
  inset = 0.05,
  label_size = NULL,
  label_colour = NULL,
  label_fill = NULL,
  ...
)

er_style_summary_gof(
  data,
  config,
  stratify,
  exposure,
  response,
  strata,
  theme,
  inset = 0.05,
  fields = c("n", "aic", "bic", "r_squared"),
```

```
  label_size = NULL,
  label_colour = NULL,
  label_fill = NULL,
  ...
)
```

Arguments

data	The original data frame.
config	Configuration for the specific plot.
stratify	Logical: whether to stratify.
exposure	Exposure variable.
response	Response variable.
strata	Stratification variable.
theme	Theme components.
inset	Distance from the panel edge for the annotation label. Defaults to 0.05.
label_size	Label text size. Defaults to NULL (ggplot2::geom_label()'s own default).
label_colour	Label text colour. Defaults to NULL (ggplot2::geom_label()'s own default).
label_fill	Label background fill. Defaults to NULL (ggplot2::geom_label()'s own default).
...	Additional named arguments forwarded from er_plot_add_model()'s own ....
fields	Fields from glance to include for er_style_summary_gof(), and the order they're shown in: one or more of "n", "aic", "bic", or "r_squared". Defaults to all four, in that order. A field is shown only when both present and non-NA in the model's glance result.

Details

See er_style() for the shared builder interface these functions implement, including how to write a custom builder of your own.

Value

A geom, or a list of geoms; see er_style().

Choosing a builder

Each builder draws a different kind of annotation, with its own data requirements:

- er_style_summary_pvalue() (the default) – a formatted p-value from the model's er_summary() result.
- er_style_summary_n() – observation counts. Doesn't have to originate from a fitted model at all.

- `er_style_summary_coefficients()` – one line per row of the model's `coefficients` table (see `er_summary()`'s `coefficients` field), useful for models with several parameters and no single privileged p-value (e.g. a multi-parameter nonlinear model). Draws nothing if `coefficients` wasn't supplied, or if the layer is stratified.
- `er_style_summary_gof()` – a single-line, comma-separated goodness-of-fit annotation from the model's `glance` field (see `er_summary()`) – a curated subset (N, AIC, BIC, R-squared) rather than every reserved `glance` column, showing only whichever of those four are actually present and non-`NA`. Same restrictions as `er_style_summary_coefficients()`: draws nothing if none of those fields are available, or if the layer is stratified.

Tags

All four builders are tagged `er_style_tag(fn, layer = "plot_summary")`, so `er_plot_add_summary()` errors informatively if a builder tagged for a different layer is passed to it instead.

See Also

`er_style()`

Examples

```
if (requireNamespace("erglm", quietly = TRUE)) {
  library(erglm)
  mod <- erglm_model(ae1 ~ aucss, erglm_data, family = binomial())

  # er_style_summary_pvalue(): the default, drawn from the model's own
  # er_summary()
  erglm_data |>
    er_plot(aucss, ae1) |>
    er_plot_add_model(mod) |>
    er_plot_add_summary(model = mod, style = er_style_summary_pvalue) |>
    plot()

  # er_style_summary_n(): model-agnostic observation count
  erglm_data |>
    er_plot(aucss, ae1) |>
    er_plot_add_model(mod) |>
    er_plot_add_summary(style = er_style_summary_n) |>
    plot()
}
```

---

er_style_tag *Register a builder's structural/aesthetic metadata*

Description

`er_style_tag()` is the shared self-declaration mechanism every `er_style_*()` builder – across all three grammars, `er_plot()`/ `er_vpc()`/`er_tte()` alike – can opt into, attaching metadata that the relevant `_add_*()` function later reads back off it and checks itself against.

Usage

```
er_style_tag(
  style,
  layout = NULL,
  vpc_layout = NULL,
  fill_role = NULL,
  y_role = NULL,
  layer = NULL,
  draw_order = NULL,
  response_types = NULL,
  plot_by_types = NULL,
  marker_source = NULL,
  label = NULL,
  overwrite = FALSE
)
```

Arguments

style	A function matching the standard signature for the grammar it's meant for – see er_style() (er_plot()), er_style_vpc() (er_vpc()), or er_style_tte() (er_tte()).
layout	One of "overlay" or "panel", or NULL (the default) to leave this tag unset. Data-layer (er_plot_add_data()) builders only – see "Details".
vpc_layout	One of "categorical" or "continuous", or NULL (the default) to leave this tag unset. VPC observed/simulated (er_vpc_add_observed()/er_vpc_add_simulated()) builders only – see "Details".
fill_role	A string naming what the builder's fill aesthetic represents, or NULL (the default) to leave this tag unset.
y_role	A string naming what the builder's y-axis represents, or NULL (the default) to leave this tag unset.
layer	One of "plot_model", "plot_summary", "plot_quantile", "plot_data", "plot_group", "vpc_observed", "vpc_simulated", "tte_curve", "tte_censor", "tte_risktable", "tte_model", or "tte_summary", naming which er_plot_add_*()/er_vpc_add_*(layer the builder is meant to be used with, or NULL (the default) to leave this tag unset. Each value is prefixed with the grammar it belongs to (plot_/vpc_/tte_) – see "Details".
draw_order	One of "foreground" or "background", or NULL (the default, equivalent to "foreground") to leave this tag unset. Only meaningful for an overlay-layout data builder; see "Details".
response_types	A character vector with one or more of "binary", "continuous", "count", or NULL (the default) to leave this tag unset (no restriction declared). For a VPC observed/simulated builder, declares which of er_vpc()'s response_type values the builder supports; see "Details".
plot_by_types	A character vector with one or more of "continuous", "discrete", or NULL (the default) to leave this tag unset. For a VPC observed/simulated builder, declares which of object$group$type values (see er_vpc()'s plot_by argument) the builder supports; see "Details".

`marker_source` One of `"summary"` or `"percentiles"`, naming which of a VPC observed/simulated builder's two config tables (`config$summary` or `config$percentiles`) it actually draws its marker(s) from, or `NULL` (the default) to leave this tag unset. See "Details".

`label` A single string, or `NULL` (the default) to leave this tag unset. Requires `layer` to also be set in the same call. Registers `style` so it can be selected by this short string (e.g. `style = "logrank"`) instead of the function itself, wherever the corresponding `_add_*()` function looks it up. See `er_style_labels()`.

`overwrite` Logical, default `FALSE`. Only meaningful together with `label`; ignored otherwise. Controls what happens when the `(layer, label)` pair is already registered to a *different* function: `FALSE` (the default) errors; `TRUE` replaces the existing registration unconditionally. Re-registering the identical function is always a silent no-op regardless of `overwrite`. See "Details".

Details

In that sense it functions as an informal builder registry – not a lookup table you register *into*, but a way of stamping a function with metadata another function can later read back off it and act on, entirely by attribute, with no central list anywhere. Every built-in builder carries a tag; nothing requires a custom builder to.

Ten tags exist today, each optional and independent – pass only the ones a given builder needs, in one call, rather than chaining separate setters. They fall into four groups, one per section below:

- Structural tags (`layout`, `vpc_layout`) – which structural family a builder belongs to.
- The layer tag (`layer`) – which layer a builder is meant to be plugged into.
- Rendering and labelling hints (`fill_role`, `y_role`, `draw_order`).
- Type-checking tags for VPC builders (`response_types`, `plot_by_types`, `marker_source`).
- Registering a label (`label`, `overwrite`).

Value

`style`, with whichever of the `"er_style_layout"`/`"er_style_vpc_layout"`/`"er_style_fill_role"`/`"er_style_y_role`
`"er_style_layer"`/`"er_style_draw_order"`/`"er_style_response_types"`/`"er_style_plot_by_types"`/`"er_style_v`
`"er_style_label"` attributes were requested attached. When `label` is supplied, `style` is also registered as a side effect – see `er_style_labels()`.

Structural tags

`layout` is a required tag for a data-layer builder specifically: `er_plot_add_data()` reads it off `style` to decide whether to place the output geoms into the main panel (`layout = "overlay"`) or to put them into separate strip-like panels above and below the main panel (`layout = "panel"`). No other layer or grammar uses this tag.

`vpc_layout` is the VPC analogue, but optional rather than required, and checked between two builders rather than read for a structural decision: when present on both the observed and simulated builder passed to a given `er_vpc` object, `er_vpc_add_simulated()` errors if they disagree (`"categorical"`, discrete bin locations; or `"continuous"`, numeric bin-midpoint locations, e.g. `er_style_vpc_simulated_quantile_ribbon()`). This catches the case where the two families

would otherwise silently plot at different x-positions for the same bin – e.g. pairing a builder that always plots at discrete bin labels with `er_style_vpc_simulated_quantile_ribbon()`'s numeric midpoints. Use a layout-matched pair instead (built-ins already are), or leave `vpc_layout` untagged – as `er_style_vpc_observed_mean_errorbar()`/`er_style_vpc_simulated_mean_errorbar()` and `er_style_vpc_observed_quantile_errorbar()`/`er_style_vpc_simulated_quantile_errorbar()` do, since both pairs adapt their x-position to `plot_by`'s type at build time rather than declaring one family statically – to skip the check entirely, the same opt-in treatment `layer` gets.

The layer tag

`layer` is optional, but unlike `fill_role`/`y_role` it isn't read for labelling. It's read by every `er_plot_add_*()`/`er_vpc_add_*()`/`er_tte_add_*()` function to catch a builder plugged into the wrong layer – e.g. passing a quantile builder to `er_plot_add_data()`, or an `er_plot()` summary builder to `er_tte_add_summary()` – with an informative error instead of whatever failure results from that layer's `config` shape not matching what the builder expects. All built-in builders carry this tag. It is a flat namespace checked only by string equality, but every value is grammar-prefixed by convention (`plot_`/`vpc_`/`tte_`) – e.g. `"plot_model"`/`"plot_summary"` for `er_plot()` vs. `"tte_model"`/`"tte_summary"` for `er_tte()` – so two grammars whose builders share neither a signature nor a `config` shape can never collide by accident, and the prefix also makes a value's owning grammar legible on sight, including in `er_style_labels()`'s own `layer` column. A custom builder that omits `layer` is never checked: it is opt-in, not a requirement like `layout` is for a data-layer builder.

Rendering and labelling hints

`fill_role` and `y_role` are both optional, and can be used to title a legend/axis correctly: `fill_role = "density"` (used by `er_style_data_hex()`) says a builder's `fill` aesthetic encodes bin density rather than strata; `y_role = "count"` (used by `er_style_group_histogram()`) says a group-layer builder's y-axis means counts rather than the group variable itself. A builder that omits either tag keeps the default behaviour (`fill` means strata; the y-axis is titled with the group variable's label), which is correct for most builders.

`draw_order` only applies to an overlay-layout data builder (`layout = "overlay"`), and controls whether its geoms are drawn before or after the model/summary/quantile layers when they share the main panel. `"foreground"`, the default for a builder that omits this tag (e.g. `er_style_data_overlay()`), draws the data geoms last, on top of everything else – appropriate for a sparse layer like individual points, which should never be hidden behind a model ribbon. `"background"` (used by `er_style_data_hex()`) draws the data geoms first, so a builder whose geoms cover the whole panel (leaving no gaps for what's underneath to show through) doesn't bury the model curve or summary annotation. `draw_order` has no effect on a panel-layout data builder (e.g. `er_style_data_boxjitter()`), since those geoms are drawn in their own separate panels, never sharing space with the model/ summary/quantile layers.

Type-checking tags for VPC builders

`response_types` and `plot_by_types` are both optional, and – unlike every other tag above – are checked against the *data*, not another builder: `er_vpc_add_observed()`/`er_vpc_add_simulated()` each check `style`'s declared `response_types` against `object$response$type` and `plot_by_types` against `object$group$type`, erroring immediately if the object's data isn't one the builder declared support for – e.g. `er_style_vpc_observed_quantile_line()` declares `response_types`

= c("continuous", "count") (it needs config$percentiles, never computed for a binary response) and plot_by_types = "continuous" (it draws a geom_line() connecting bins along the numeric midpoint, meaningless for an unordered categorical plot_by). This catches an incompatible builder/data pairing at the er_vpc_add_*() call site, before any binning or summarising happens, rather than only when the builder itself is finally invoked by plot()/er_vpc_build(). As with layer, both tags are opt-in – an untagged builder is never checked against either, so a custom builder that doesn't declare them keeps working unchanged (though it's then responsible for guarding against its own incompatible inputs, the way every built-in VPC builder still does internally as a fallback).

marker_source is also optional and VPC-specific. It's used when cropping a built VPC to xlim/ylim (see er_vpc_theme()) to decide which of a builder's two config tables to check for a marker falling outside the plotted axis limits. A builder that plots config$summary (e.g. er_style_vpc_observed_mean_errorbar()) should tag marker_source = "summary"; one that plots config$percentiles (e.g. er_style_vpc_observed_quantile_l er_style_vpc_observed_quantile_errorbar()) should tag marker_source = "percentiles". An untagged builder has both tables checked, which is always safe but can produce a spurious warning about a table the builder never actually draws from.

Registering a label

label is unlike every tag above, in that it isn't purely descriptive: supplying it registers style as a side effect, so that the corresponding _add_*() function can accept the string in place of style itself. It requires layer in the same call – the registry is keyed by (layer, label), not label alone, which is what lets two different layers reuse the same label string with no ambiguity (each _add_*() function only ever looks inside its own layer's partition). Wired up for every _add_*() function in the package – see er_style_labels() for the full list of registered (layer, label) pairs.

Re-registering the same (layer, label) pair with the identical function is always a silent no-op (this is what makes reloading the package, which re-tags every built-in builder, safe). Re-registering it with a *different* function errors by default – e.g. re-running a script that edits a custom labelled builder's body and re-tags it hits this – unless overwrite = TRUE is passed, which replaces the registration unconditionally.

See Also

er_plot_add_data(), er_style(), er_style_vpc(), er_style_tte(), er_style_labels()

Examples

```
build_data_density <- er_style_tag(
  function(data, config, stratify, exposure, response, strata, theme, ...) {
    ggplot2::geom_density_2d(
      data = data,
      mapping = ggplot2::aes(x = .data[[exposure$name]], y = .data[[response$name]])
    )
  },
  layout = "overlay",
  layer = "plot_data"
)
```

er_style_tte *Builder functions for time-to-event plots*

Description

Documents the shared `function(data, config, stratify, time, strata, theme, ...)` signature every `er_style_tte_*()` builder implements, including how to write a custom one. The TTE analogue of `er_style()`'s shared interface for the `er_plot()` grammar, adapted for a time x-axis/ survival-probability y-axis instead of exposure/response.

Details

This page documents the shared interface all `er_style_tte_*()` builders implement. The builders themselves are documented on their own family-specific pages, one per layer:

- `er_style_tte_curve()` – the `curve` layer (`er_tte_add_curve()`)
- `er_style_tte_censor()` – the `censor` layer (`er_tte_add_censor()`)
- `er_style_tte_risktable()` – the `risktable` layer (`er_tte_add_risktable()`)
- `er_style_tte_summary()` – the `summary` layer (`er_tte_add_summary()`)
- `er_style_tte_model()` – the `model` layer (`er_tte_add_model()`)

`er_style_tte_risktable_text()` is the one family whose geoms are drawn into a separate patchwork panel stacked below the curve, rather than onto the curve panel directly – its builder still receives the same standard signature, but the returned geoms are composed differently at build time.

Value

A geom, or a list of geoms. More precisely, a list of objects that can be added to a ggplot2 plot, on top of a partially constructed panel that already has the base theme and a coord applied (time on the x-axis, survival probability on the y-axis for every layer except `risktable`, whose panel has no such coord).

Arguments

Every `er_style_tte_*()` builder receives:

- `data` – The original data frame (`object$data`).
- `config` – Configuration for the specific layer; see each family-specific page below for what it populates.
- `stratify` – Logical: whether the fit is stratified (`!is.null(object$strata)`).
- `time` – `object$time` (`name`/`label`/`limits`).
- `strata` – `object$strata` (`var`/`label`), or `NULL` when unstratified.
- `theme` – Theme components (`object$theme`).
- `...` – Additional named arguments forwarded from the corresponding `er_tte_add_*()` call's own `...`; see "Passing extra arguments to a builder" below.

Writing your own builder

Every `er_style_tte_*()` function above shares the signature documented in the "Arguments" section above, and that signature is a public part of the API: any function `function(data, config, stratify, time, strata, ...)` that returns a geom or list of geoms can stand in for a built-in builder, passed as `style` to the matching `er_tte_add_*()` function.

A custom builder can self-declare which layer it's meant for via `er_style_tag(builder, layer = ...)`, one of `"curve"`, `"censor"`, `"risktable"`, `"tte_summary"`, or `"tte_model"`. Every `er_tte_add_*()` function checks a builder's `layer` tag, if it has one, against the layer it was actually passed to, erroring immediately if they disagree. `"tte_summary"`/`"tte_model"` are deliberately distinct from `er_plot_add_summary()`/`er_plot_add_model()`'s own `"summary"`/`"model"` tags – the two grammars' summary/model builders share no signature or `config` contents, so a builder written for `er_plot()` would otherwise silently pass the tag check if passed to `er_tte_add_summary()`/`er_tte_add_model()` instead. This tag is entirely optional; an untagged custom builder is simply never checked. See `er_style_tag()` for the full set of attributes a builder can carry.

All five layers are **singleton**: calling the corresponding `er_tte_add_*()` function again replaces that layer's builder rather than adding another one. Unlike `er_plot()`'s `group` layer, no TTE layer is additive.

Passing extra arguments to a builder

Every `er_tte_add_*()` function (`er_tte_add_curve()`, `er_tte_add_censor()`, `er_tte_add_risktable()`, `er_tte_add_summary()`, `er_tte_add_model()`) takes its own `...`, forwarded unchanged to `style` when it's actually called at build time. Extra arguments must be named, since they're appended positionally after the six standard arguments; an unnamed one errors immediately rather than silently binding to the wrong parameter. A builder that doesn't need any extra arguments simply declares `...` and ignores it – every built-in TTE builder does exactly this.

See Also

`er_style()`, `er_style_tte_curve()`, `er_style_tte_censor()`, `er_style_tte_risktable()`, `er_style_tte_summary()`, `er_style_tte_model()`, `er_style_tag()`

---

er_style_tte_censor *Censoring-mark builders for the TTE grammar*

Description

Builder functions for the `censor` layer (`er_tte_add_censor()`), marking each censoring time directly on the Kaplan-Meier curve.

Usage

```
er_style_tte_censor_ticks(
  data,
  config,
  stratify,
```

```
  time,
  strata,
  theme,
  shape = 3,
  point_size = 2,
  stroke = 0.75,
  ...
)
```

Arguments

data	The original data frame (object$data).
config	Configuration for the censor layer (populated by er_tte_add_censor()): config$table (the subset of the tidy KM table where n_censor > 0, with a strata column when stratified).
stratify	Logical: whether the fit is stratified (!is.null(object$strata)).
time	object$time (name/label/limits).
strata	object$strata (var/label), or NULL when unstratified.
theme	object$theme.
shape	Point shape for a censoring mark. Default 3 (a plus sign), the conventional Kaplan-Meier censoring glyph.
point_size	Point size. Default 2.
stroke	Point stroke width. Default 0.75.
...	Additional named arguments forwarded from er_tte_add_censor()'s own ....

Details

See er_style_tte() for the shared interface every TTE-grammar builder implements.

A censoring-only row of the KM table (n_censor > 0, n_event == 0) carries the survival value the curve already had going into that time – Kaplan-Meier survival only drops at an *event* time – so a mark drawn at (time, surv) lands exactly on the step curve without any extra lookup.

Stratified colour maps to config$table's own strata column, the same already-cleaned stratum label er_style_tte_curve_km() uses, so a censoring mark takes on the colour of the curve it sits on. The marks never contribute their own legend entry (show.legend = FALSE) – the curve layer's legend already identifies each stratum.

er_style_tte_censor_ticks() is tagged er_style_tag(fn, layer = "censor"), so er_tte_add_censor() errors informatively if handed a builder tagged for a different layer.

Value

A geom, or a list of geoms.

See Also

er_tte_add_censor(), er_style_tte()

Examples

```
library(survival)
lung |>
  er_tte(time, status == 2) |>
  er_tte_add_curve() |>
  er_tte_add_censor(style = er_style_tte_censor_ticks, shape = 124, point_size = 3) |>
  plot()
```

---

er_style_tte_curve *Kaplan-Meier curve builders for the TTE grammar*

Description

Builder functions for the curve layer (er_tte_add_curve()), drawing the Kaplan-Meier estimate as a step function with an optional step-shaped confidence band.

Usage

```
er_style_tte_curve_km(
  data,
  config,
  stratify,
  time,
  strata,
  theme,
  show_ci = TRUE,
  ribbon_alpha = 0.15,
  linewidth = 1,
  ...
)
```

Arguments

data	The original data frame (object$data).
config	Configuration for the curve layer (populated by er_tte_add_curve()): config$table (the tidy KM table, with a (0, 1) origin row prepended per stratum), config$time_upper (the time-axis upper limit, needed so the last confidence-band interval has somewhere to end), config$conf_level.
stratify	Logical: whether the fit is stratified (!is.null(object$strata)).
time	object$time (name/label/limits).
strata	object$strata (var/label), or NULL when unstratified.
theme	object$theme.
show_ci	Whether to draw the confidence band. Default TRUE.
ribbon_alpha	Transparency of the confidence band (0-1). Default 0.15.
linewidth	Width of the step curve's line. Default 1.
...	Additional named arguments forwarded from er_tte_add_curve()'s own ....

Details

See er_style_tte() for the shared interface every TTE-grammar builder implements.

A Kaplan-Meier confidence band is a step function, just like the curve itself, but ggplot2 has no built-in "step ribbon" geom (unlike ggplot2::geom_step() for the line). er_style_tte_curve_km() works around this with ggplot2::geom_rect(): one rectangle per interval between consecutive event/censoring times, with xmin/xmax the interval's start/end time and ymin/ymax the interval's constant lower/upper bound – visually identical to a step ribbon, without needing a bespoke stat.

Stratified colour/fill both map to config$table's own strata column (the already-cleaned stratum label, e.g. "Q1" or a categorical level) rather than the original stratify_by column on data, since that's what config$table actually carries. er_tte_build() retitles the resulting legend with strata$label (e.g. "sex") afterwards, so a builder itself never needs to know the original variable's name.

er_style_tte_curve_km() is tagged er_style_tag(fn, layer = "curve"), so er_tte_add_curve() errors informatively if handed a builder tagged for a different layer.

Value

A geom, or a list of geoms.

See Also

er_tte_add_curve(), er_style_tte()

Examples

```
library(survival)
lung |>
  er_tte(time, status == 2) |>
  er_tte_add_curve(style = er_style_tte_curve_km, ribbon_alpha = 0.3) |>
  plot()
```

---

er_style_tte_model *Model-curve builders for the TTE grammar*

Description

Builder functions for the model layer (er_tte_add_model()), drawing a fitted parametric S(t) curve with an optional uncertainty band.

Usage

```
er_style_tte_model_line(
  data,
  config,
  stratify,
  time,
```

```
  strata,
  theme,
  show_ci = TRUE,
  ribbon_alpha = 0.15,
  linewidth = 1,
  ...
)
```

Arguments

data	The original data frame (object$data).
config	Configuration for the model layer (populated by er_tte_add_model()): config$predictions (the prediction tibble from er_predict_survival(), with time/fit_survival/ci_lower/ci_upper columns), config$time_grid, config$conf_level.
stratify	Logical: whether the fit is stratified (!is.null(object$strata)).
time	object$time (name/label/limits).
strata	object$strata (var/label), or NULL when unstratified.
theme	object$theme.
show_ci	Whether to draw the confidence band. Default TRUE.
ribbon_alpha	Transparency of the confidence band (0-1). Default 0.15.
linewidth	Width of the curve's line. Default 1.
...	Additional named arguments forwarded from er_tte_add_model()'s own ....

Details

See er_style_tte() for the shared interface every TTE-grammar builder implements.

Unlike er_style_tte_curve_km()'s Kaplan-Meier step curve, config$predictions is a smooth prediction grid (one row per newdata row x config$time_grid value), so er_style_tte_model_line() draws an ordinary ggplot2::geom_line()/ggplot2::geom_ribbon() pair rather than a step function.

Stratified colour/fill both map to config$predictions's own strata column (named after strata$var) rather than a fixed name – unlike er_style_tte_curve_km(), which always reads a column literally named strata (the tidied Kaplan-Meier table's own naming). er_tte_build() still retitles the resulting legend with strata$label afterwards.

er_style_tte_model_line() is tagged er_style_tag(fn, layer = "tte_model") – distinct from er_plot_add_model()'s own "model" tag, since the two grammars' model builders share no signature or config contents – so er_tte_add_model() errors informatively if handed a builder tagged for a different layer (including er_plot_add_model()'s own builders).

Value

A geom, or a list of geoms.

See Also

er_tte_add_model(), er_style_tte_curve_km(), er_style_tte()

er_style_tte_risktable

Number-at-risk builders for the TTE grammar

Description

Builder functions for the `risktable` layer (`er_tte_add_risktable()`), drawing a row of risk counts per stratum at a grid of time points. Unlike every other TTE-grammar builder, this one's geoms are drawn into their own patchwork panel below the curve, not onto the curve's panel directly – see `er_tte_build()`.

Usage

```
er_style_tte_risktable_text(
  data,
  config,
  stratify,
  time,
  strata,
  theme,
  text_size = 3.5,
  show_percent = FALSE,
  ...
)
```

Arguments

`data`	The original data frame (`object$data`).
`config`	Configuration for the risktable layer (populated by `er_tte_add_risktable()`): `config$table` (`time`/`n_risk`/`strata`/ `n_baseline` – the stratum's time-zero number at risk, used by `show_percent` below – one row per requested time break per stratum) and `config$breaks` (the time breaks themselves, also used as the curve panel's x-axis ticks).
`stratify`	Logical: whether the fit is stratified (`!is.null(object$strata)`).
`time`	`object$time` (`name`/`label`/`limits`).
`strata`	`object$strata` (`var`/`label`), or `NULL` when unstratified.
`theme`	`object$theme`.
`text_size`	Size of the risk-count text. Default `3.5`.
`show_percent`	Whether to append each break's `n_risk` as a percentage of that stratum's own baseline (time-zero) size, formatted via `er_tte_theme()`'s `format_percent`. Default `FALSE` (a bare count, the previous behaviour).
`...`	Additional named arguments forwarded from `er_tte_add_risktable()`'s own `....`

Details

See er_style_tte() for the shared interface every TTE-grammar builder implements.

Rows are ordered top-to-bottom in the same order strata first appear in config$table (reversed, since a ggplot2 discrete y-axis plots its first level at the bottom); an unstratified fit gets a single "All" row.

show_percent = TRUE displays "<n_risk> (<percent>%)" instead of a bare n_risk, using er_tte_theme()'s format_percent (defaulting to scales::label_percent(accuracy = 1)) to format n_risk / n_baseline – see config above for where n_baseline comes from.

er_style_tte_risktable_text() is tagged er_style_tag(fn, layer = "risktable"), so er_tte_add_risktable() errors informatively if handed a builder tagged for a different layer.

Value

A geom, or a list of geoms.

See Also

er_tte_add_risktable(), er_style_tte()

Examples

```
library(survival)
lung |>
  er_tte(time, status == 2) |>
  er_tte_add_curve() |>
  er_tte_add_risktable(style = er_style_tte_risktable_text, text_size = 4) |>
  plot()
```

---

er_style_tte_summary *Summary annotation builders for the TTE grammar*

Description

Builder functions for the summary layer (er_tte_add_summary()), drawing a corner-placed text/label annotation from a log-rank test comparing survival across stratify_by's levels, a supplied model's er_summary() result, or observation/event counts.

Usage

```
er_style_tte_summary_logrank(
  data,
  config,
  stratify,
  time,
  strata,
  theme,
```

```
  inset = 0.05,
  label_size = NULL,
  label_colour = NULL,
  label_fill = NULL,
  ...
)

er_style_tte_summary_n(
  data,
  config,
  stratify,
  time,
  strata,
  theme,
  inset = 0.05,
  label_size = NULL,
  label_colour = NULL,
  label_fill = NULL,
  ...
)

er_style_tte_summary_coefficients(
  data,
  config,
  stratify,
  time,
  strata,
  theme,
  inset = 0.05,
  label_size = NULL,
  label_colour = NULL,
  label_fill = NULL,
  ...
)

er_style_tte_summary_gof(
  data,
  config,
  stratify,
  time,
  strata,
  theme,
  inset = 0.05,
  fields = c("n", "aic", "bic", "r_squared"),
  label_size = NULL,
  label_colour = NULL,
  label_fill = NULL,
  ...
```

)

Arguments

data	The original data frame (object$data).
config	Configuration for the summary layer (populated by er_tte_add_summary()): config$logrank_p_value (the log-rank test's p-value, or NULL when fewer than 2 strata levels are present in the data), config$summary (the supplied model's raw er_summary() result, or NULL when no model was supplied), and config$corner_distance (how uncrowded each panel corner is, relative to the plotted survival curve(s) – see er_style()'s ?er_plot_add_summary()-analogous corner-placement idiom).
stratify	Logical: whether this layer was added with keep_strata = TRUE (the default whenever stratify_by was set in er_tte()) – see er_tte_add_summary().
time	object$time (name/label/limits).
strata	object$strata (var/label).
theme	object$theme – theme$format_p/theme$format_number format the annotation's numbers.
inset	Distance from the panel edge for the annotation label, as a fraction of the panel's width/height. Default 0.05.
label_size	Label text size. Defaults to NULL (ggplot2::geom_label()'s own default).
label_colour	Label text colour. Defaults to NULL (ggplot2::geom_label()'s own default).
label_fill	Label background fill. Defaults to NULL (ggplot2::geom_label()'s own default).
...	Additional named arguments forwarded from er_tte_add_summary()'s own ....
fields	Fields from glance to include for er_style_tte_summary_gof(), and the order they're shown in: one or more of "n", "aic", "bic", or "r_squared". Defaults to all four, in that order. A field is shown only when both present and non-NA in the model's glance result.

Details

See er_style_tte() for the shared builder interface these functions implement.

Value

A geom, or a list of geoms.

Choosing a builder

Each builder draws a different kind of annotation, with its own data requirements:

- er_style_tte_summary_logrank() (the default) – a log-rank p-value comparing survival across stratify_by's levels. Draws nothing on an unstratified object, or one with fewer than 2 strata levels present in the data.

- `er_style_tte_summary_n()` – subject/event counts. Doesn't depend on a model or `stratify_by` at all.
- `er_style_tte_summary_coefficients()` – one line per row of a supplied model's `coefficients` table (see `er_summary()`). Draws nothing without a model supplying `coefficients`, or if the layer is stratified.
- `er_style_tte_summary_gof()` – a single-line goodness-of-fit summary from a supplied model's `glance` field. Same requirements and restrictions as `er_style_tte_summary_coefficients()`.

Log-rank test

`er_style_tte_summary_logrank()` places its annotation in whichever of the panel's 4 corners is currently furthest from the survival curve(s), using the same `(0, 1)`-rescaled corner-distance calculation `er_plot_add_summary()`'s own p-value annotation uses to avoid a plot's raw data points – here applied to the curve's own `(time, surv)` coordinates instead, since there's no raw per-subject scatter in this grammar for the annotation to avoid. It draws nothing if `config$logrank_p_value` is `NULL` (fewer than 2 strata levels present, including an unstratified object).

Subject and event counts

`er_style_tte_summary_n()` draws subject and event counts – one line per stratum when `stratify` is `TRUE`, a single overall line otherwise – and doesn't depend on a model or `stratify_by` at all.

Model coefficients and goodness-of-fit

`er_style_tte_summary_coefficients()` draws one line per row of the supplied model's `coefficients` table (see `er_summary()`'s `coefficients` field); it draws nothing if `coefficients` wasn't supplied, or if the layer is stratified. `er_style_tte_summary_gof()` draws a single-line, comma-separated goodness-of-fit annotation from the model's `glance` field – a curated subset (N, AIC, BIC, R-squared) rather than every reserved `glance` column, showing only whichever of those four are actually present and non-`NA`; it draws nothing if none of them are available, or if the layer is stratified.

Tags

All four builders are tagged `er_style_tag(fn, layer = "tte_summary")` – distinct from `er_plot_add_summary()`'s own `"summary"` tag, since the two grammars' summary builders share no signature (`exposure`/`response` vs. `time`) – so `er_tte_add_summary()` errors informatively if a builder tagged for a different layer is passed to it instead (including `er_plot_add_summary()`'s own builders).

See Also

`er_tte_add_summary()`, `er_style_tte()`

Examples

```
library(survival)
lung |>
  transform(sex = factor(sex, labels = c("Male", "Female"))) |>
  er_tte(time, status == 2, stratify_by = sex) |>
  er_tte_add_curve() |>
```

```
  er_tte_add_summary(style = er_style_tte_summary_logrank, label_fill = "white") |>
  plot()

# a purely descriptive annotation, with no log-rank test at all
lung |>
  er_tte(time, status == 2) |>
  er_tte_add_curve() |>
  er_tte_add_summary(style = er_style_tte_summary_n) |>
  plot()
```

`er_style_vpc` *Builder functions for VPC plots*

Description

Documents the shared `function(data, config, exposure, response, theme, ...)` signature every `er_style_vpc_*()` builder implements, including how to write a custom one. The VPC analogue of `er_style()`'s shared interface for the `er_plot()` grammar.

Details

This page documents the shared interface all `er_style_vpc_*()` builders implement. The builders themselves are documented on their own family-specific pages, one per side of the observed/simulated comparison:

- `er_style_vpc_observed()` – the `observed` layer (`er_vpc_add_observed()`)
- `er_style_vpc_simulated()` – the `simulated` layer (`er_vpc_add_simulated()`)

Value

A geom, or a list of geoms. More precisely, a list of objects that can be added to a ggplot2 plot, on top of a partially constructed plot that already has the base theme and a coord applied.

Arguments

Every `er_style_vpc_*()` builder receives:

- `data` – The original data frame passed to `er_vpc()`.
- `config` – Configuration for the specific layer (`observed` or `simulated`); see `er_style_vpc_observed()`/`er_style_vp` for what each populates.
- `exposure` – The plot's `exposure` variable (`object$exposure`) – not necessarily the variable plotted on the x-axis; see "Exposure is not always the x-axis variable" below.
- `response` – The plot's `response` variable (`object$response`).
- `theme` – Theme components (`object$theme`).
- `...` – Additional named arguments forwarded from the corresponding `er_vpc_add_*()` call's own `...`; see "Passing extra arguments to a builder" below.

Unlike er_style()'s signature, there is no stratify/strata pair: er_vpc()'s own stratify_by is facet-only (no colour/fill precedence rule to thread through a builder), so every builder-facing config table already carries the faceting column it needs, and no builder ever has to branch on stratification itself.

Exposure is not always the x-axis variable

plot_by (object$group), not exposure, drives a VPC's x-axis; the two only coincide when the caller didn't override plot_by. A builder that needs the x-axis variable's own label or numeric range should read object$group$label/config$group_limits instead of exposure$label/ exposure$limits – see er_vpc()'s plot_by argument.

Writing your own builder

Every er_style_vpc_*() function above shares the signature documented in the "Arguments" section above, and that signature is a public part of the API: any function function(data, config, exposure, response, the that returns a geom or list of geoms can stand in for a built-in builder, passed as style to er_vpc_add_observed()/ er_vpc_add_simulated().

A custom builder can self-declare metadata via er_style_tag(): vpc_layout ("categorical"/"continuous", checked for agreement between the observed and simulated builders paired on one er_vpc object – distinct from the data layer's own layout tag, which uses an unrelated "overlay"/"panel" pair), layer ("observed"/ "simulated", checked against the layer the builder was actually passed to), response_types/plot_by_types (checked against object$response$type/object$group$type), and marker_source (which of config$summary/config$percentiles the builder actually draws from, used by er_vpc_theme()'s xlim/ylim cropping). All five are optional and independent – see er_style_tag() for the full explanation of each, including which built-in builders use them and why.

Both observed and simulated are **singleton** layers: calling er_vpc_add_observed()/er_vpc_add_simulated() again replaces the previous builder rather than adding another one.

Passing extra arguments to a builder

er_vpc_add_observed()/er_vpc_add_simulated() each take their own . . . , forwarded unchanged to style when it's actually called at build time. Extra arguments must be named, since they're appended positionally after the five standard arguments; an unnamed one errors immediately rather than silently binding to the wrong parameter. A builder that doesn't need any extra arguments simply declares . . . and ignores it – every built-in VPC builder does exactly this.

See Also

er_style(), er_style_vpc_observed(), er_style_vpc_simulated(), er_style_tag()

er_style_vpc_observed *Observed-layer builders for VPC plots*

Description

Builder functions for the `observed` layer (`er_vpc_add_observed()`), drawing the observed side of a visual predictive check as a mean/rate + confidence interval per bin (the default, adaptive to `plot_by`'s type), a continuous-x line of empirical percentiles, or a point/interval per bin *and* per requested percentile.

Usage

```
er_style_vpc_observed_quantile_line(
  data,
  config,
  exposure,
  response,
  theme,
  point_size = 1.5,
  ...
)

er_style_vpc_observed_quantile_errorbar(
  data,
  config,
  exposure,
  response,
  theme,
  point_size = 1.5,
  errorbar_width = NULL,
  dodge = 0,
  prob_dodge_width = 0,
  ...
)

er_style_vpc_observed_mean_errorbar(
  data,
  config,
  exposure,
  response,
  theme,
  point_size = 2,
  errorbar_width = NULL,
  dodge = 0,
  show_label = FALSE,
  label_size = 3,
  ...
```

)

Arguments

data	The original data frame.
config	Configuration for the observed layer.
exposure	Exposure variable.
response	Response variable.
theme	Theme components.
point_size	Point size for all three point/interval builders. Defaults to 1.5 for er_style_vpc_observed_quantile_li er_style_vpc_observed_quantile_errorbar(), or 2 for er_style_vpc_observed_mean_errorbar(
...	Additional named arguments forwarded from er_vpc_add_observed()'s own ....
errorbar_width	Width of er_style_vpc_observed_mean_errorbar()'s and er_style_vpc_observed_quantile_erro error bars. Interpreted differently depending on plot_by's type: for a categorical plot_by, it's a bar width in the same implied unit-scaled category-gap units ggplot2::geom_errorbar() normally expects; for a numeric plot_by, it's a fraction of plot_by's own range (config$group_limits), so 1 would span the full range. Defaults to NULL, which resolves to 0.15 (er_style_vpc_observed_quantile_errorbar()) or 0.2 (er_style_vpc_observed_mean_errorbar()) for a categorical plot_by, or 0.025 for a numeric one.
dodge	Horizontal offset (as a fraction of plot_by's own range, like errorbar_width for a numeric plot_by) applied to all of this builder's error bars/points, for both er_style_vpc_observed_mean_errorbar() and er_style_vpc_observed_quantile_errorbar(). Default 0 (no offset, the previous behaviour). Useful for manually separating the observed layer from an overlapping simulated one at the same bin – e.g. dodge = -0.01 on the observed builder paired with dodge = 0.01 on the corresponding simulated builder. Only supported when plot_by is numeric; a nonzero value is ignored with a warning for a categorical plot_by, where dodging isn't implemented yet.
prob_dodge_width	Horizontal spread (as a fraction of plot_by's own range) applied to er_style_vpc_observed_quantile_ requested probs within a single bin, symmetrically centred on that bin's own position (added on top of dodge, if also supplied). Default 0 (all probs plotted at the same position, the previous behaviour). Useful when several probs' error bars overlap enough to be unreadable. Same numeric-plot_by-only restriction as dodge.
show_label	For er_style_vpc_observed_mean_errorbar() only: whether to draw config$summary's y_mid_lbl (the rate/mean, formatted via er_vpc_theme()'s format_percent/format_number) as a text label just above each point's upper CI bound. Default FALSE (no label, the previous behaviour).
label_size	Text size for show_label's label. Defaults to 3.

Details

See er_style_vpc() for the shared interface every VPC-grammar builder implements.

Value

A list of geoms; see `er_style()`.

Choosing a builder

All three builders plot the observed side of a bin against the simulated side drawn by their `er_style_vpc_simulated()` counterpart; which one to reach for depends on how much of the response's distribution you need to see, and what kind of response/`plot_by` you have:

- `er_style_vpc_observed_mean_errorbar()` (the default) – one point + confidence interval per bin, summarising the rate/mean only. Works for every response type and either kind of `plot_by`. Start here unless you specifically need percentile bands.
- `er_style_vpc_observed_quantile_line()` – a continuous line per requested percentile, for a fuller picture of the response's distribution across bins. Requires a continuous/count response and a numeric `plot_by`; pairs with `er_style_vpc_simulated_quantile_ribbon()`.
- `er_style_vpc_observed_quantile_errorbar()` – a point + interval per requested percentile per bin, the discrete-bin analogue of the line idiom above. Same response-type restriction, but also works with a categorical `plot_by`; pairs with `er_style_vpc_simulated_quantile_errorbar()`.

Mean/errorbar (default)

`er_style_vpc_observed_mean_errorbar()` plots `config$summary`'s rate/mean + confidence interval, adapting its x-position to `plot_by`'s type (`config$is_numeric_group`): equally spaced at each bin's categorical (or quantile-bin) label when `plot_by` is categorical, or at each bin's numeric median (`x_median`, from `config$summary`) on `plot_by`'s own numeric scale when `plot_by` is numeric. Because it adapts its x-position family at build time rather than declaring one statically, it carries no `vpc_layout` tag – pair it with `er_style_vpc_simulated_mean_errorbar()`, which mirrors the same adaptive logic.

Percentile line

`er_style_vpc_observed_quantile_line()` plots `config$percentiles` – one line per requested percentile – at each bin's numeric midpoint on `plot_by`'s own numeric scale, for pairing with `er_style_vpc_simulated_quantile_ribbon()`. `config$percentiles` is only computed for a continuous/count response (see `er_vpc()`'s `probs` argument); calling `er_style_vpc_observed_quantile_line()` without it errors.

Percentile errorbar

`er_style_vpc_observed_quantile_errorbar()` plots `config$percentiles` – a point + confidence interval (via `ci_quantile()`) for each requested percentile – for pairing with `er_style_vpc_simulated_quantile_e`
Like `er_style_vpc_observed_mean_errorbar()`, it adapts its x-position to `plot_by`'s type (`config$is_numeric_group`): equally spaced at each bin's categorical (or quantile-bin) label when `plot_by` is categorical, or at each bin's numeric median (`x_median`, from `config$percentiles`) on `plot_by`'s own numeric scale when `plot_by` is numeric. Because it adapts its x-position family at build time rather than declaring one statically, it carries no `vpc_layout` tag. Unlike `er_style_vpc_observed_quantile_line()`/`er_style_vpc_simulated_quantile_ribbon()`, it supports a categorical `plot_by` as well as a numeric one; like it, it requires a continuous/count response (a binary response's distribution is already fully described by its rate) and errors informatively without `config$percentiles`. When

more than one percentile is requested, all of them are currently plotted at the same x-position within a bin rather than dodged apart, so overlapping error bars/points are only distinguishable by their y-position – dodging support may be added in a future release.

Legends and overlap

Each builder maps a constant `color = "Observed"`, so ggplot2 merges its legend entry with whatever the paired simulated-layer builder maps for `"Simulated"` into a single combined legend.

In the worst case – `er_style_vpc_observed_quantile_errorbar()` paired with `er_style_vpc_simulated_quantile_er` for a numeric `plot_by` with several `probs` – up to `2 * length(probs)` error bars land at the exact same x-position within a bin (every `probs` value, for both the observed and simulated layers), which can be unreadable. `dodge` (separating the observed and simulated layers) and `prob_dodge_width` (spreading a single layer's own `probs` apart) are both opt-in, manual escape hatches for this – see their own argument docs above. Neither is automatic, because which collision is actually occurring (source-vs-source, `probs`-vs-`probs`, or both) depends on the data at hand.

See Also

`er_style_vpc()`, `er_style_vpc_simulated()`

Examples

```
if (requireNamespace("erglm", quietly = TRUE)) {
  library(erglm)
  mod <- erglm_model(ae2 ~ aucss + sex, erglm_data, family = binomial())

  # er_style_vpc_observed_mean_errorbar(): the default, adaptive to
  # plot_by's type
  erglm_data |>
    er_vpc(aucss, ae2, plot_by = aucss) |>
    er_vpc_add_observed(style = er_style_vpc_observed_mean_errorbar) |>
   er_vpc_add_simulated(model = mod, seed = 6203, style = er_style_vpc_simulated_mean_errorbar) |>
    plot()

  # er_style_vpc_observed_quantile_line(): continuous-x percentile
  # lines, paired with the matching simulated ribbon builder
  mod2 <- erglm_model(biomarker_change ~ aucss, erglm_data, family = gaussian())
  erglm_data |>
    er_vpc(aucss, biomarker_change, plot_by = aucss) |>
    er_vpc_add_observed(style = er_style_vpc_observed_quantile_line) |>
    er_vpc_add_simulated(
      model = mod2, seed = 8417, style = er_style_vpc_simulated_quantile_ribbon
    ) |>
    plot()
}
```

er_style_vpc_simulated *Simulated-layer builders for VPC plots*

Description

Builder functions for the `simulated` layer (`er_vpc_add_simulated()`), drawing the simulated side of a visual predictive check as a mean + percentile interval per bin (the default, adaptive to `plot_by`'s type), continuous-x percentile bands, or a point/interval per bin *and* per requested percentile.

Usage

```
er_style_vpc_simulated_quantile_ribbon(
  data,
  config,
  exposure,
  response,
  theme,
  ribbon_alpha = 0.3,
  ribbon_edges = FALSE,
  edge_linetype = "dotted",
  edge_linewidth = 0.5,
  edge_colour = "grey50",
  median_linetype = "dashed",
  median_linewidth = 0.5,
  median_colour = "grey30",
  ...
)

er_style_vpc_simulated_quantile_errorbar(
  data,
  config,
  exposure,
  response,
  theme,
  point_size = 1.5,
  errorbar_width = NULL,
  dodge = 0,
  prob_dodge_width = 0,
  ...
)

er_style_vpc_simulated_mean_errorbar(
  data,
  config,
  exposure,
```

```
  response,
  theme,
  point_size = 2,
  errorbar_width = NULL,
  dodge = 0,
  show_label = FALSE,
  label_size = 3,
  ...
)
```

Arguments

data	The original data frame.
config	Configuration for the simulated layer.
exposure	Exposure variable.
response	Response variable.
theme	Theme components.
ribbon_alpha	Fill transparency for er_style_vpc_simulated_quantile_ribbon()'s bands. Defaults to 0.3.
ribbon_edges	Whether er_style_vpc_simulated_quantile_ribbon() additionally draws a line along each band's own ci_lower/ci_upper bounds, on top of the shaded ribbon fill – mirrors er_style_model_ribbonline()'s own ribbon_edges argument. Default FALSE (ribbon fill only). Useful when several requested percentiles' bands overlap: the edge lines stay legible even where the fills merge into an indistinguishable blob.
edge_linetype, edge_linewidth, edge_colour	Styling for er_style_vpc_simulated_quantile_ribbon()'s optional edge lines (only drawn when ribbon_edges = TRUE). Defaults to a thin, light dotted line ("dotted", 0.5, "grey50") that stays unobtrusive even when several bands' edges overlap.
median_linetype, median_linewidth, median_colour	Styling for er_style_vpc_simulated_quantile_ribbon()'s median line, drawn at each band's y_mid. Defaults match the previous fixed styling ("dashed", 0.5, "grey30").
...	Additional named arguments forwarded from er_vpc_add_simulated()'s own ....
point_size	Point size for both point/interval builders. Defaults to 1.5 for er_style_vpc_simulated_quantile_err or 2 for er_style_vpc_simulated_mean_errorbar().
errorbar_width	Width of er_style_vpc_simulated_mean_errorbar()'s and er_style_vpc_simulated_quantile_e error bars. Interpreted differently depending on plot_by's type: for a categorical plot_by, it's a bar width in the same implied unit-scaled category-gap units ggplot2::geom_errorbar() normally expects; for a numeric plot_by, it's a fraction of plot_by's own range (config$group_limits), so 1 would span the full range. Defaults to NULL, which resolves to 0.15 (er_style_vpc_simulated_quantile_errorbar() or 0.2 (er_style_vpc_simulated_mean_errorbar()) for a categorical plot_by, or 0.025 for a numeric one.

dodge — Horizontal offset (as a fraction of plot_by's own range, like errorbar_width for a numeric plot_by) applied to all of this builder's error bars/points, for both er_style_vpc_simulated_mean_errorbar() and er_style_vpc_simulated_quantile_errorbar(). Default 0 (no offset, the previous behaviour); pair with an opposite-signed dodge on the corresponding observed builder to manually separate the two layers where they'd otherwise overlap at the same bin. See er_style_vpc_observed()'s own dodge docs for the full explanation, including the categorical-plot_by restriction.

prob_dodge_width — Horizontal spread (as a fraction of plot_by's own range) applied to er_style_vpc_simulated_quantile requested probs within a single bin. Default 0 (the previous behaviour); see er_style_vpc_observed()'s own prob_dodge_width docs.

show_label — For er_style_vpc_simulated_mean_errorbar() only: whether to draw config$summary's y_mid_lbl (the mean, formatted via er_vpc_theme()'s format_percent/format_number) as a text label just above each point's upper CI bound. Default FALSE (no label, the previous behaviour); see er_style_vpc_observed()'s own show_label docs.

label_size — Text size for show_label's label. Defaults to 3.

Details

See er_style_vpc() for the shared interface every VPC-grammar builder implements.

Value

A list of geoms; see er_style().

Choosing a builder

All three builders plot the simulated side of a bin against the observed side drawn by their er_style_vpc_observed() counterpart; which one to reach for depends on how much of the response's distribution you need to see, and what kind of response/plot_by you have:

- er_style_vpc_simulated_mean_errorbar() (the default) – one point + percentile interval per bin, summarising the mean only. Works for every response type and either kind of plot_by. Start here unless you specifically need percentile bands.
- er_style_vpc_simulated_quantile_ribbon() – a shaded band per requested percentile, for a fuller picture of the response's distribution across bins. Requires a continuous/count response and a numeric plot_by; pairs with er_style_vpc_observed_quantile_line().
- er_style_vpc_simulated_quantile_errorbar() – a point + interval per requested percentile per bin, the discrete-bin analogue of the ribbon idiom above. Same response-type restriction, but also works with a categorical plot_by; pairs with er_style_vpc_observed_quantile_errorbar().

Mean/errorbar (default)

er_style_vpc_simulated_mean_errorbar() plots config$summary's mean + percentile interval (of the mean, across replicates), adapting its x-position to plot_by's type (config$is_numeric_group): equally spaced at each bin's categorical (or quantile-bin) label when plot_by is categorical, or at

each bin's numeric median (x_median, from config$summary) on the plot_by's own numeric scale when plot_by is numeric. Because it adapts its x-position family at build time rather than declaring one statically, it carries no vpc_layout tag – pair it with er_style_vpc_observed_mean_errorbar(), which mirrors the same adaptive logic.

Percentile ribbon

er_style_vpc_simulated_quantile_ribbon() plots config$percentiles – one shaded band (median line + interval) per requested percentile – at each bin's numeric midpoint on plot_by's own numeric scale, for pairing with er_style_vpc_observed_quantile_line(). config$percentiles is only computed for a continuous/count response (see er_vpc()'s probs argument); calling er_style_vpc_simulated_qua without it errors.

Percentile errorbar

er_style_vpc_simulated_quantile_errorbar() plots config$percentiles – a point + across-replicate percentile interval for each requested percentile – for pairing with er_style_vpc_observed_quantile_errorbar(Like that builder (and like er_style_vpc_simulated_mean_errorbar()), it adapts its x-position to plot_by's type, carries no vpc_layout tag, supports both a numeric and a categorical plot_by, and requires a continuous/count response, erroring informatively without config$percentiles. As with the observed-layer counterpart, when more than one percentile is requested they are currently all plotted at the same x-position within a bin rather than dodged apart.

Legends and overlap

er_style_vpc_simulated_mean_errorbar()/er_style_vpc_simulated_quantile_errorbar() map a constant color = "Simulated"; er_style_vpc_simulated_quantile_ribbon() maps a constant fill = "Simulated". ggplot2 merges either into the paired observed builder's own "Observed" legend entry (same aesthetic) into one combined legend; the ribbon's fill legend is separate from the point/errorbar builders' color legend.

When several requested percentiles' bands sit close together (small per-bin samples, few simulated replicates, or probs values close to one another), er_style_vpc_simulated_quantile_ribbon()'s bands can overlap enough that the shaded fills merge into a single indistinguishable region, and its median lines – all styled identically – become the only way to tell the bands apart, which fails wherever two of them cross. ribbon_edges = TRUE mitigates this by drawing each band's own ci_lower/ci_upper bounds as a line (see edge_linetype/edge_linewidth/edge_colour), which stays legible even where the fills themselves are illegible.

See Also

er_style_vpc(), er_style_vpc_observed()

Examples

```
if (requireNamespace("erglm", quietly = TRUE)) {
  library(erglm)
  mod <- erglm_model(ae2 ~ aucss + sex, erglm_data, family = binomial())

  # er_style_vpc_simulated_mean_errorbar(): the default, adaptive to
  # plot_by's type
```

```
  erglm_data |>
    er_vpc(aucss, ae2, plot_by = aucss) |>
    er_vpc_add_observed(style = er_style_vpc_observed_mean_errorbar) |>
   er_vpc_add_simulated(model = mod, seed = 6203, style = er_style_vpc_simulated_mean_errorbar) |>
    plot()

  # er_style_vpc_simulated_quantile_errorbar(): a point + interval per
  # requested percentile, paired with the matching observed builder
  mod2 <- erglm_model(biomarker_change ~ aucss, erglm_data, family = gaussian())
  erglm_data |>
    er_vpc(aucss, biomarker_change, plot_by = aucss) |>
    er_vpc_add_observed(style = er_style_vpc_observed_quantile_errorbar) |>
    er_vpc_add_simulated(
      model = mod2, seed = 8417, style = er_style_vpc_simulated_quantile_errorbar
    ) |>
    plot()
}
```

---

er_tte *The time-to-event plotting mini-language*

Description

Create an er_tte specification for a time-to-event plot. Build the plot by adding layers for survival curves, censoring markers, risk tables, textual summaries, and model predictions; render with plot()/print() or er_tte_build().

Usage

```
er_tte(data, time, event, stratify_by = NULL, conf_level = 0.95)
```

Arguments

data	Data frame or tibble containing the observed data.
time	Event/censoring time (unquoted expression, evaluated in data). Must be non-negative.
event	Event indicator (unquoted expression, evaluated in data): TRUE/1 for an event, FALSE/0 for censoring.
stratify_by	Optional stratification variable (unquoted, bare column name), used as-is. Must be discrete – a numeric column errors. Defaults to NULL (a single, unstratified curve).
conf_level	Confidence level for the Kaplan-Meier confidence band. Must be strictly between 0 and 1. Defaults to 0.95.

Details

er_tte() computes the (single-arm) Kaplan-Meier estimate once, via survival::survfit(), and stores the fit plus a tidy per-event-time table (time, n_risk, n_event, n_censor, surv, lower, upper) on object$km. Layers added afterwards – the curve (er_tte_add_curve()), censoring marks (er_tte_add_censor()), a number-at-risk panel (er_tte_add_risktable()), summary annotation (er_tte_add_summary()), and a parametric model overlay (er_tte_add_model()) – read from this shared fit rather than recomputing it (the model layer alone reads from the caller-supplied model instead, via er_predict_survival()).

Unlike er_plot()/er_vpc(), time/event accept arbitrary tidy-eval expressions, not just bare column names – time-to-event data very commonly needs an inline transform to get an event indicator (e.g. status == 2 for a coded status variable, or !is.na(progression_date)), and requiring the caller to first dplyr::mutate() that column into existence would just be boilerplate. The evaluated time/event vectors are stored as .er_tte_time/.er_tte_event columns on object$data; their rlang::as_label()-derived text is kept as object$time$label/ object$event$label for display purposes.

event must evaluate to a logical vector (TRUE = event occurred) or a numeric vector taking only the values 0 (censored) and 1 (event) – exactly the same binary encoding er_plot() requires of a response_type = "binary" response.

Optional stratify_by splits the Kaplan-Meier estimate into one curve per level, via survival::survfit()'s ~ strata formula side. It must name a discrete/categorical variable – mirroring er_plot()/ er_vpc()'s own stratify_by, a numeric one errors; bin it yourself first with cut_quantile()/cut_exposure_quantile() and pass the resulting factor, for full control over bin count/tie-breaking/labels. Unlike time/event, stratify_by must be a bare column name (not an arbitrary expression), matching exposure/response/stratify_by elsewhere in the package. objectkmtable gains a strata column when stratified; object$strata (var/label) mirrors er_vpc()'s own object$strata.

Value

An (empty of layers) plot object of class er_tte, with the Kaplan-Meier fit already computed on object$km.

See Also

er_model_interface

Examples

```
library(survival)
lung |>
  er_tte(time, status == 2)

# `lung$sex` is coded numerically (1/2); `stratify_by` requires a
# discrete variable, so convert it to a factor first
lung |>
  transform(sex = factor(sex, labels = c("Male", "Female"))) |>
  er_tte(time, status == 2, stratify_by = sex)
```

`er_tte_add_censor` *Add a censoring-marks layer*

Description

Adds the censoring layer to a TTE plot, showing the times at which a subject was censored, read from the fit already stored internally within the plot object.

Usage

```
er_tte_add_censor(object, style = NULL, ...)
```

Arguments

`object`	Partially constructed plot (has S3 class `er_tte`).
`style`	Style used to draw the censoring marks layer. Can either be a string corresponding to one of the registered style labels (e.g., `"ticks"`, the default), or a builder function used to compute the relevant plot object (see "Styles" below).
`...`	Additional named arguments forwarded to the `style` builder function when the plot is built.

Value

The input `object`, with the censor layer added.

Styles

The following pre-defined styles are available for this layer. Please see the documentation for the corresponding builder function to see what customisation options are available:

Label	Builder	Description
`"ticks"`	`er_style_tte_censor_ticks()`	Tick marks at each censoring time, on the curve's current step height (the only

See `er_style_tte()` for details on how style builder functions are defined for the TTE mini-grammar, should a custom style be required.

See Also

`er_tte()`, `er_style_tte_censor_ticks()`

Examples

```
library(survival)
lung |>
  er_tte(time, status == 2) |>
  er_tte_add_curve() |>
  er_tte_add_censor() |>
  plot()
```

`er_tte_add_curve` *Add a Kaplan-Meier curve layer*

Description

Adds the curve layer to a TTE plot: a Kaplan-Meier step curve with a confidence band, computed from the fit already contained within the plot object.

Usage

```
er_tte_add_curve(object, style = NULL, ...)
```

Arguments

`object`	Partially constructed plot (has S3 class `er_tte`).
`style`	Style used to draw the Kaplan-Meier curve and ribbon. Can either be a string corresponding to one of the registered style labels (e.g., `"km"`, the default), or a builder function used to compute the relevant plot object (see "Styles" below).
`...`	Additional named arguments forwarded to the `style` builder function when the plot is built.

Value

The input `object`, with the curve layer added.

Styles

The following pre-defined styles are available for this layer. Please see the documentation for the corresponding builder function to see what customisation options are available:

Label	Builder	Description
`"km"`	`er_style_tte_curve_km()`	Kaplan-Meier step curve with a confidence band (the only built-in, and the default).

See `er_style_tte()` for details on how style builder functions are defined for the TTE mini-grammar, should a custom style be required.

See Also

er_tte(), er_style_tte_curve_km()

Examples

```
library(survival)
lung |>
  er_tte(time, status == 2) |>
  er_tte_add_curve() |>
  plot()

lung |>
  transform(sex = factor(sex, labels = c("Male", "Female"))) |>
  er_tte(time, status == 2, stratify_by = sex) |>
  er_tte_add_curve() |>
  plot()
```

---

er_tte_add_model *Add a model-based survival curve overlay*

Description

Adds the model layer to a TTE plot: a fitted survival curve with an uncertainty band derived from the corresponding time-to-event model.

Usage

```
er_tte_add_model(
  object,
  model,
  style = NULL,
  keep_strata = NULL,
  conf_level = 0.95,
  time_grid = NULL,
  predict_args = list(),
  ...
)
```

Arguments

object	Partially constructed plot (has S3 class er_tte).
model	A fitted time-to-event model. Must implement er_predict_survival().
style	Style used to draw the model-based survival curve. Can either be a string corresponding to one of the registered style labels (e.g., "line", the default), or a builder function used to compute the relevant plot object (see "Styles" below).

keep_strata	Logical; whether this layer should use stratification. Defaults to TRUE when a stratification variable has been specified, and FALSE otherwise.
conf_level	Confidence level for the prediction band. Defaults to 0.95.
time_grid	Numeric vector of times at which to predict S(t), or NULL (the default) to use 100 points evenly spaced across the time range.
predict_args	A named list of additional arguments forwarded to er_predict_survival() when generating model-based predictions.
...	Additional named arguments forwarded to the style builder function when the plot is built.

Details

The model layer of a TTE plot is used to display predictions generated from an underlying survival model (e.g., parametric accelerated failure time model, Cox proportional hazards model, etc). It uses the model object to create the predictions, using the er_predict_survival() method for the relevant model class to do the work. The model object is permitted to reference covariates other than the plot stratification variable: see the details section of er_plot_add_model() for the specifics.

Note that erplots does not check that model was fit on the same time/event variables passed to the plot itself; it is left to the user to ensure that the data set provided to the model is consistent with the data provided to the TTE plot.

Value

The input object, with the model layer added.

Styles

The following pre-defined styles are available for this layer. Please see the documentation for the corresponding builder function to see what customisation options are available:

Label	Builder	Description
"line"	er_style_tte_model_line()	Fitted S(t) curve with an uncertainty band (the only built-in, and the default).

See er_style_tte() for details on how style builder functions are defined for the TTE mini-grammar, should a custom style be required.

See Also

er_tte(), er_style_tte_model_line(), er_model_interface

`er_tte_add_risktable` *Add a number-at-risk panel*

Description

Adds the at-risk table layer to a TTE plot: a separate panel stacked below the curve, showing the number of subjects still at risk at a grid of time points, computed from the fit already stored internally within the plot object.

Usage

```
er_tte_add_risktable(object, style = NULL, times = NULL, n_times = 6, ...)
```

Arguments

`object`	Partially constructed plot (has S3 class `er_tte`).
`style`	Style used to generate the at-risk table in the plot. Can either be a string corresponding to one of the registered style labels (e.g., `"text"`, the default), or a builder function used to compute the relevant plot object (see "Styles" below).
`times`	Numeric vector of time points at which to report the number at risk, or `NULL` (the default) to use `n_times` evenly spaced breaks across the time range.
`n_times`	Number of evenly spaced breaks to use when `times` is `NULL`. Must be a single whole number of at least 2. Ignored when `times` is supplied. Defaults to `6`.
`...`	Additional named arguments forwarded to the `style` builder function when the plot is built.

Details

The time breaks used for the number-at-risk grid also become the x-axis tick marks on the primary plot, so the two panels' collected x-axis lines up exactly (see `er_tte_build()`).

Value

The input `object`, with the risktable layer added.

Styles

The following pre-defined styles are available for this layer. Please see the documentation for the corresponding builder function to see what customisation options are available:

Label	Builder	Description
`"text"`	`er_style_tte_risktable_text()`	Number-at-risk counts as a text grid, one row per stratum (the only built-in, a

See `er_style_tte()` for details on how style builder functions are defined for the TTE mini-grammar, should a custom style be required.

See Also

er_tte(), er_style_tte_risktable_text()

Examples

```
library(survival)
lung |>
  er_tte(time, status == 2) |>
  er_tte_add_curve() |>
  er_tte_add_risktable() |>
  plot()
```

---

er_tte_add_summary *Add a summary annotation layer*

Description

Adds the summary layer to a TTE plot: a corner-placed text/label annotation, summarising one or more aspects of the plot or the data.

Usage

```
er_tte_add_summary(
  object,
  model = NULL,
  style = NULL,
  keep_strata = NULL,
  conf_level = 0.95,
  summary_args = list(),
  ...
)
```

Arguments

object	Partially constructed plot (has S3 class er_tte).
model	A fitted time-to-event model implementing er_summary(), or NULL (the default). Only needed for styles that produce model-based summaries, ignored by other style builder functions.
style	Style used to produce the summary layer annotation. Can either be a string corresponding to one of the registered style labels (e.g., "logrank", the default), or a builder function used to compute the relevant plot object (see "Styles" below).
keep_strata	Logical; whether this layer should use stratification. Defaults to TRUE when a stratification variable has been specified, and FALSE otherwise.
conf_level	Confidence level forwarded to er_summary() (see ?er_model_interface). Defaults to 0.95. Ignored when model is NULL.

summary_args — A named list of additional arguments forwarded to er_summary() when generating summaries.

... — Additional named arguments forwarded to the style builder function when the plot is built.

Details

The annotation is placed in whichever corner of the panel is currently furthest from the plotted survival curve(s), computed the same way er_plot_add_summary()'s corner-placed annotation avoids the raw data – see er_style_tte_summary_logrank().

The default log-rank builder draws nothing on an unstratified object, or one with only 1 stratum level present in the data, rather than erroring – a log-rank test needs at least 2 groups to compare. Other builders (e.g. er_style_tte_summary_n()) work regardless of stratification.

Value

The input object, with the summary layer added.

Styles

The following pre-defined styles are available for this layer. Please see the documentation for the corresponding builder function to see what customisation options are available:

Label	Builder	Description
"logrank"	er_style_tte_summary_logrank()	Log-rank test p-value comparing survival across stratify
"n"	er_style_tte_summary_n()	Subject/event counts; model- and stratification-agnostic.
"coefficients"	er_style_tte_summary_coefficients()	One line per model parameter, from model's er_summary(
"gof"	er_style_tte_summary_gof()	A goodness-of-fit annotation from model's er_summary()

See er_style_tte() for details on how style builder functions are defined for the TTE mini-grammar, should a custom style be required.

See Also

er_tte(), er_style_tte_summary_logrank()

Examples

```
library(survival)
lung |>
  transform(sex = factor(sex, labels = c("Male", "Female"))) |>
  er_tte(time, status == 2, stratify_by = sex) |>
  er_tte_add_curve() |>
  er_tte_add_summary() |>
  plot()

# a purely descriptive annotation, with no model or log-rank test at all
lung |>
  er_tte(time, status == 2) |>
```

```
er_tte_add_curve() |>
er_tte_add_summary(style = er_style_tte_summary_n) |>
plot()
```

`er_tte_build` *Build and render a time-to-event plot*

Description

Assembles the layers for a time-to-event plot object: a survival panel that displays the curve, censor, summary, and model layers' geoms, when present. When a risk table layer is also present the result contains two panels stacked vertically.

Usage

```
er_tte_build(object)
```

Arguments

`object`	Partially constructed plot (has S3 class `er_tte`).

Details

The user does not typically invoke this function directly. Instead, it is called automatically when `plot()` is called.

Value

The input `object`, with `object$output` (the composed plot – a single ggplot2 object, or a patchwork object when the risktable layer is present) populated.

See Also

`er_tte()`, `er_tte_add_curve()`, `er_tte_add_censor()`, `er_tte_add_risktable()`, `er_tte_add_summary()`

er_tte_theme *Adjust theme/labels for a time-to-event plot*

Description

Set axis/legend labels, plot titles/captions, axis limits, theme objects, formatters, the legend key glyph, and relative panel heights for a time-to-event plot. This does not change which variable is mapped to which aesthetic – that's the builder's job via `style` (see er_style_tte_curve).

Usage

```
er_tte_theme(
  object,
  xlab = NULL,
  ylab = NULL,
  strata_lab = NULL,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  xlim = NULL,
  ylim = NULL,
  theme_base = NULL,
  theme_extra = NULL,
  format_p = NULL,
  format_percent = NULL,
  format_number = NULL,
  draw_key = NULL,
  height_curve = NULL,
  height_risktable = NULL
)
```

Arguments

`object`	Partially constructed plot (has S3 class `er_tte`).
`xlab`	Time axis label (single string). See "Details" for why `xlim` (not this) drives layout decisions; `xlab` is purely cosmetic.
`ylab`	Survival-probability axis label (single string).
`strata_lab`	Stratification legend label (single string). Errors if `stratify_by` wasn't set in `er_tte()` – there's no stratification legend to label.
`title, subtitle, caption`	Plot-level annotation text (single strings).
`xlim`	Time-axis limits (length-2, increasing numeric vector, no `NA`) – overwrites `object$time$limits`. See "Details" for the call-order caveat.
`ylim`	Survival-probability axis limits (length-2, increasing numeric vector, no `NA`). Purely cosmetic; defaults to `c(0, 1)`.

theme_base
: A ggplot2 theme object (e.g. ggplot2::theme_minimal()) – the swappable overall visual theme, defaulting to ggplot2::theme_bw().

theme_extra
: A ggplot2 theme object (e.g. from ggplot2::theme()) with additional theme tweaks layered on top of theme_base. See "Details" for its default and replacement semantics.

format_p
: Formatter function (typically from scales::label_pvalue()), used by er_tte_add_summary()'s log-rank annotation.

format_percent
: Formatter function (typically from scales::label_percent()), used by er_style_tte_risktable_te show_percent argument to format a risk-table count as a percentage of its stratum's baseline size.

format_number
: Formatter function (typically from scales::label_number()), used by er_tte_add_summary()'s er_style_tte_summary_coefficients()/er_style_tte_summary_gof() builders to format coefficient/goodness-of-fit values.

draw_key
: A key-glyph function (e.g. ggplot2::draw_key_point()), passed as the curve/model layers' key_glyph argument.

height_curve, height_risktable
: Relative panel heights (single positive numbers), used when er_tte_add_risktable()'s panel is stacked below the curve panel. Supplying only one leaves the other unchanged.

Details

Every argument defaults to NULL, meaning "leave whatever was set before unchanged". This allows repeated calls to er_tte_theme() to update only the supplied fields, like ggplot2::theme(). There is no implicit way to reset a field to the er_tte() default.

xlim overwrites object$time$limits directly (the same structural field er_tte() itself sets from the data), rather than a purely cosmetic ggplot2::coord_cartesian() zoom – mirroring er_plot_theme()'s own xlim (not er_vpc_theme()'s, which is display-only). This matters because object$time$limits also drives non-cosmetic defaults computed at *add-layer* time: er_tte_add_model()'s default time_grid and er_tte_add_risktable()'s default times both span it. Call er_tte_theme(xlim = ...) *before* those layers when you want the narrower/wider range reflected in their defaults; calling it after only changes the curve panel's own x-axis display. ylim, by contrast, is purely cosmetic (survival probability is always modelled on [0, 1]; this only changes what's displayed), stored on object$theme$ylim and defaulting to c(0, 1).

theme_extra defaults to a panel border plus legend.position = "bottom". Supplying a new value fully replaces this default rather than merging with it, so re-include the border/legend-position settings too if you want to keep them alongside your own additions.

title/subtitle/caption are applied via a single ggplot2::labs() call on the curve panel – unlike er_plot_theme()'s patchwork::plot_annotation() indirection, this is enough even when er_tte_add_risktable()'s panel is stacked below it via patchwork::wrap_plots(), since the curve panel is always the top-most one.

Value

The input object, with the requested theme fields updated.

See Also

er_tte(), er_style_tte_curve

Examples

```
library(survival)
lung |>
  er_tte(time, status == 2) |>
  er_tte_add_curve() |>
  er_tte_theme(
    xlab = "Days", ylab = "Survival probability",
    title = "Overall survival"
  ) |>
  plot()
```

---

er_vpc *The exposure-response VPC mini-language*

Description

Create an er_vpc specification for a visual predictive check. Build the plot by adding an observed layer and a simulated layer, and render with plot()/print() or er_vpc_build().

Usage

```
er_vpc(
  data,
  exposure,
  response,
  stratify_by = NULL,
  response_type = "auto",
  plot_by = NULL,
  n_bins = 4,
  ties = "upward",
  quantile_type = 7,
  labeller = NULL,
  conf_level = 0.95,
  probs = c(0.1, 0.5, 0.9),
  seed = NULL
)
```

Arguments

data	Data frame or tibble containing the observed data.
exposure	Exposure variable (one variable, unquoted).
response	Response variable (one variable, unquoted).

`stratify_by` Optional variable (unquoted) splitting the VPC into one facet panel per level, via `ggplot2::facet_wrap()`, used as-is. Must be discrete – a numeric column errors; bin it yourself first with `cut_quantile()`/`cut_exposure_quantile()` and pass the resulting factor, for full control over bin count/tie-breaking/labels. Must resolve to a different variable than `plot_by`. Defaults to `NULL` (no faceting, a single panel, matching prior behaviour).

`response_type` One of `"auto"` (the default), `"binary"`, `"continuous"`, or `"count"`.

`plot_by` Variable (unquoted) plotted on the x-axis and used to bin/group the observed vs. simulated comparison. Defaults to `exposure`. A numeric variable is split into `n_bins` quantile bins (placebo, i.e. `0`, kept in its own bin when `plot_by` is the exposure variable itself); a categorical variable is used as-is, with no binning.

`n_bins` Number of quantile bins, when `plot_by` is numeric. Defaults to `4`.

`ties`, `quantile_type`, `labeller`
Control how a numeric `plot_by` is split into quantile bins – passed straight through to `cut_exposure_quantile()`, see its documentation for what each controls. Set here, on `er_vpc()` itself, rather than on `er_vpc_add_observed()`/`er_vpc_add_simulated(` because the two layers must always agree on how `plot_by` is binned – `er_vpc_add_simulated()` reuses these exact settings (via `cut_exposure_quantile()`'s attributes on the observed layer's own binned column) rather than re-resolving them, so both sides always stay in sync.

`conf_level` Confidence level for both the observed- and simulated-side intervals. Must be strictly between 0 and 1. Defaults to `0.95`.

`probs` Percentiles to compute for a percentile-based builder (e.g. `er_style_vpc_observed_quantile_line()`/`er_style_vpc_observed_quantile_errorbar()`/`er_style_vpc_simulated_quantile_errorbar()` ignored by the default adaptive mean/errorbar pair). Only computed for a continuous/count response. Defaults to `c(0.1, 0.5, 0.9)`.

`seed` Optional single number seeding the observed layer's random tie-break when `ties` is `"split-even"` (ignored otherwise). `NULL` (the default) draws from the ambient RNG stream. The simulated layer's own `"split-even"` tie-break is instead seeded by `er_vpc_add_simulated()`'s own `seed` argument – the two are independent random draws over different data (the observed rows vs. the, typically larger, simulated replicate pool), so each is seeded by the call that actually performs it.

Details

`er_vpc_add_observed()` bins the observed data and computes its response summary; `er_vpc_add_simulated()` must be added afterwards, since it reuses the observed layer's own binning decision so both sides share identical bin boundaries. Both layers are singletons (a second call replaces the previous one).

`er_vpc()`'s own stratification (`stratify_by`) is real, but simpler than `er_plot()`'s: a facet-only split via `ggplot2::facet_wrap()`, with no colour/facet precedence rule to reconcile (see `stratify_by` below). It's orthogonal to `plot_by` – see `er_vpc_add_observed()` for `plot_by`, the variable plotted on the x-axis and used to bin/group the comparison. Whether `plot_by` is `"continuous"` (numeric, quantile-binned) or `"discrete"` (used as-is) is auto-detected from the column's type and stored on `object$group$type`, mirroring how `object$response$type` records the response's type.

Value

An (empty) plot object of class `er_vpc`.

See Also

`er_vpc_add_observed()`, `er_vpc_add_simulated()`, `er_vpc_build()`, er_model_interface

Examples

```
if (requireNamespace("erglm", quietly = TRUE)) {
library(erglm)
mod <- erglm_model(ae2 ~ aucss + sex, erglm_data, family = binomial())

erglm_data |>
  er_vpc(aucss, ae2, plot_by = aucss) |>
  er_vpc_add_observed() |>
  er_vpc_add_simulated(model = mod, seed = 9984) |>
  plot()
}
```

`er_vpc_add_observed` *Add the observed-data layer to a VPC plot*

Description

Bins the observed data for a VPC plot and computes binned response summaries for later comparison against a simulated data layer.

Usage

```
er_vpc_add_observed(object, style = er_style_vpc_observed_mean_errorbar, ...)
```

Arguments

`object`	Partially constructed VPC (has S3 class `er_vpc`).
`style`	Style used to draw the VPC observed data layer. Can either be a string corresponding to one of the registered style labels (e.g., `"mean_errorbar"`, the default), or a builder function used to compute the relevant plot object (see "Styles" below).
`...`	Additional named arguments forwarded to the `style` builder function when the plot is built.

Details

`plot_by`/`n_bins`/`conf_level`/`probs` are set once on `er_vpc()` itself (rather than here) so the observed and simulated layers can't disagree about how the comparison is binned or summarized.

Value

`object`, with `object$layer$observed` populated.

Styles

The following pre-defined styles are available for this layer. Please see the documentation for the corresponding builder function to see what customisation options are available:

Label	Builder	Description
`"mean_errorbar"`	`er_style_vpc_observed_mean_errorbar()`	Mean/rate + CI per bin, x-position adaptive to
`"quantile_line"`	`er_style_vpc_observed_quantile_line()`	One line per requested percentile against a co
`"quantile_errorbar"`	`er_style_vpc_observed_quantile_errorbar()`	Point + error bar per bin and per requested pe

See `er_style_vpc()` for details on how style builder functions are defined for the VPC mini-grammar, should a custom style be required.

See Also

`er_vpc()`, `er_vpc_add_simulated()`, `er_style_vpc_observed()`

---

er_vpc_add_simulated *Add the simulated-data layer to a VPC plot*

Description

Bins the simulation data for a VPC plot and computes binned response summaries for comparison against the observed data layer.

Usage

```
er_vpc_add_simulated(
  object,
  model = NULL,
  sim = NULL,
  style = er_style_vpc_simulated_mean_errorbar,
  nsim = 100,
  seed = NULL,
  simulate_args = list(),
  ...
)
```

Arguments

object	Partially constructed VPC (has S3 class er_vpc), which must already have an observed layer (see er_vpc_add_observed()).
model	A fitted model implementing er_simulate() with sim_resp. Mutually exclusive with sim.
sim	Simulated data with matching exposure/response/plot_by columns and sim_id. Mutually exclusive with model.
style	Style used to draw the VPC simulation layer. Can either be a string corresponding to one of the registered style labels (e.g., "mean_errorbar", the default), or a builder function used to compute the relevant plot object (see "Styles" below).
nsim	Number of simulation replicates, only used with model. Defaults to 100.
seed	Optional RNG seed. Used for model's own simulation draws, and also (regardless of whether sim/model was supplied) to seed this layer's random tie-break when er_vpc()'s ties is "split-even" – see er_vpc()'s own seed argument for the observed layer's independent tie-break seed.
simulate_args	A named list of additional arguments forwarded to er_simulate() when simulating from the model.
...	Additional named arguments forwarded to the style builder function when the plot is built.

Details

sim and model are mutually exclusive; supply exactly one. model is preferred when it implements er_simulate() with sim_resp, since a VPC needs response-level simulated observations rather than only mean predictions – this function errors informatively if sim_resp isn't available.

conf_level/probs are set once on er_vpc() itself (rather than here), so the observed and simulated layers always agree on them.

Value

object, with object$layer$simulated populated.

Styles

The following pre-defined styles are available for this layer. Please see the documentation for the corresponding builder function to see what customisation options are available:

Label	Builder	Description
"mean_errorbar"	er_style_vpc_simulated_mean_errorbar()	Mean/rate + CI per bin, x-position adaptive
"quantile_ribbon"	er_style_vpc_simulated_quantile_ribbon()	One ribbon per requested percentile against
"quantile_errorbar"	er_style_vpc_simulated_quantile_errorbar()	Point + error bar per bin and per requested p

See er_style_vpc() for details on how style builder functions are defined for the VPC mini-grammar, should a custom style be required.

See Also

er_vpc(), er_vpc_add_observed(), er_style_vpc_simulated()

---

er_vpc_build *Build and render a VPC plot*

Description

Assembles the observed/simulated layers into a single ggplot2 object.

Usage

```
er_vpc_build(object)
```

Arguments

object	Partially constructed VPC (has S3 class er_vpc).

Details

The user does not typically invoke this function directly. Instead, it is called automatically when plot() is called.

Value

The input object, with object$output (the composed ggplot2 plot) populated.

See Also

er_vpc()

---

er_vpc_theme *Adjust theme/labels for a VPC plot*

Description

Set axis/legend labels, plot titles/captions, axis limits, theme objects, and formatters for a VPC. This does not change which variable is mapped to which aesthetic – that's the builder's job via style (see er_style()).

Usage

```
er_vpc_theme(
  object,
  xlab = NULL,
  ylab = NULL,
  strata_lab = NULL,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  xlim = NULL,
  ylim = NULL,
  theme_base = NULL,
  theme_extra = NULL,
  format_percent = NULL,
  format_number = NULL
)
```

Arguments

object	Partially constructed VPC (has S3 class er_vpc).
xlab	Label for the VPC's x-axis (single string) – see "Details" for why this labels plot_by, not exposure.
ylab	Response axis label (single string).
strata_lab	Facet strip label prefix (single string), e.g. the "Sex" in a "Sex: Female" strip. Errors if stratify_by wasn't set in er_vpc() – there's no facet strip to relabel.
title, subtitle, caption	Plot-level annotation text (single strings).
xlim, ylim	Axis limits (length-2, increasing numeric vectors), applied via ggplot2::coord_cartesian(clip = "off").
theme_base	A ggplot2 theme object (e.g. ggplot2::theme_minimal()) – the swappable overall visual theme, defaulting to ggplot2::theme_bw().
theme_extra	A ggplot2 theme object (e.g. from ggplot2::theme()) with additional theme tweaks layered on top of theme_base. See "Details" for its default and replacement semantics.
format_percent, format_number	Formatter functions (typically from scales::label_*()), used to compute config$summary$y_mid_lbl for a binary response (format_percent) or a continuous/count response (format_number). Displayed by er_style_vpc_observed_mean_errorbar()/er_style_vpc_simulated_mean_errorba only when their own show_label = TRUE (default FALSE) – otherwise computed but not drawn by any built-in builder. Default to scales::label_percent(accuracy = 1) and scales::label_number(accuracy = 0.01) respectively.

Details

Every argument defaults to NULL, meaning "leave whatever was set before unchanged". This allows repeated calls to er_vpc_theme() to update only the supplied fields, like ggplot2::theme(). There is no implicit way to reset a field to the er_vpc() default.

xlab labels plot_by (stored on object$group$label), not exposure – plot_by drives the VPC's actual x-axis, and the two only coincide when the caller didn't override plot_by in er_vpc().

theme_extra defaults to a panel border plus legend.position = "bottom". Supplying a new value fully replaces this default rather than merging with it, so re-include the border/legend-position settings too if you want to keep them alongside your own additions.

Unlike er_plot_theme(), there is no color_discrete/fill_discrete argument here: the observed-vs-simulated colour/fill distinction uses a fixed, shared scale (with "Observed"/"Simulated" always mapped to the same two hues) to keep the two aligned across builders that mix colour and fill for the same idea, and swapping it out is not yet supported. Adding + ggplot2::scale_colour_manual(...)/+ ggplot2::scale_fill_manual(...) to the built/returned ggplot2 object remains the escape hatch for this, and for any other tweak not covered by this function's arguments (e.g. draw_key, which isn't wired up for any built-in VPC builder).

Value

The input object, with the requested theme fields updated.

See Also

er_vpc(), er_style()

Examples

```
if (requireNamespace("erglm", quietly = TRUE)) {
  library(erglm)
  mod <- erglm_model(ae1 ~ aucss, erglm_data, family = binomial())

  erglm_data |>
    er_vpc(aucss, ae1) |>
    er_vpc_add_observed() |>
    er_vpc_add_simulated(model = mod, seed = 1234) |>
    er_vpc_theme(
      xlab = "AUC at steady state",
      ylab = "Probability of event",
      title = "Visual predictive check"
    ) |>
    plot()
}
```

Index

∗ **datasets**
    erplots_data, 8

ci_clopper_pearson, 3
ci_clopper_pearson(), *4–6*, *23*
ci_poisson, 4
ci_poisson(), *23*
ci_quantile, 5
ci_quantile(), *73*
ci_t, 6
ci_t(), *4*, *23*
cut(), *7*
cut_exposure_quantile *(*cut_quantile*)*, 6
cut_exposure_quantile(), *14*, *18*, *22*, *80*, *92*
cut_quantile, 6
cut_quantile(), *14*, *18*, *80*, *92*

er_model_interface, 10, *14*, *80*, *84*, *93*
er_plot, 13
er_plot(), *16*, *19*, *21*, *23*, *26–29*, *32*, *53*, *80*, *92*
er_plot_add_data, 14
er_plot_add_data(), *14*, *16*, *19*, *21*, *23*, *26*, *30–35*, *54*, *55*, *57*
er_plot_add_groups, 17
er_plot_add_groups(), *14*, *16*, *21*, *23*, *26*, *30*, *36*, *39*, *40*
er_plot_add_model, 19
er_plot_add_model(), *11*, *14*, *16*, *19*, *23*, *26*, *30*, *31*, *42–44*, *52*, *59*, *63*, *84*
er_plot_add_quantiles, 22
er_plot_add_quantiles(), *3*, *4*, *6*, *14*, *16*, *18*, *19*, *21*, *26*, *30*, *31*, *45*, *47*, *48*
er_plot_add_summary, 25
er_plot_add_summary(), *11*, *14*, *16*, *19*, *21*, *23*, *30*, *44*, *48*, *50*, *53*, *59*, *68*, *87*
er_plot_build, 26
er_plot_build(), *13*, *14*, *18*, *23*, *28*
er_plot_theme, 27
er_plot_theme(), *14*, *48*, *90*, *98*
er_predict *(*er_model_interface*)*, 10
er_predict(), *20*
er_predict_survival *(*er_model_interface*)*, 10
er_predict_survival(), *63*, *80*, *83*, *84*
er_simulate *(*er_model_interface*)*, 10
er_simulate(), *21*, *32*, *43*, *44*, *95*
er_style, 30
er_style(), *12*, *15*, *16*, *19*, *21*, *23*, *26*, *29*, *35*, *39*, *40*, *43*, *44*, *48*, *49*, *52–54*, *57–59*, *67*, *69*, *70*, *73*, *77*, *96*, *98*
er_style_data, 33
er_style_data(), *30*, *32*, *33*, *39*
er_style_data_boxjitter *(*er_style_data*)*, 33
er_style_data_boxjitter(), *15*
er_style_data_hex *(*er_style_data*)*, 33
er_style_data_hex(), *15*, *56*
er_style_data_overlay *(*er_style_data*)*, 33
er_style_data_overlay(), *15*
er_style_group, 36
er_style_group(), *30*, *33*
er_style_group_boxjitter *(*er_style_group*)*, 36
er_style_group_boxjitter(), *19*
er_style_group_boxplot *(*er_style_group*)*, 36
er_style_group_boxplot(), *19*
er_style_group_histogram *(*er_style_group*)*, 36
er_style_group_histogram(), *18*, *19*, *56*
er_style_group_linerange *(*er_style_group*)*, 36
er_style_group_linerange(), *19*
er_style_group_violin *(*er_style_group*)*, 36
er_style_group_violin(), *18*, *19*

er_style_group_violinjitter (er_style_group), 36
er_style_group_violinjitter(), 19
er_style_labels, 41
er_style_labels(), 44, 55–57
er_style_model, 42
er_style_model(), 30, 33
er_style_model_line (er_style_model), 42
er_style_model_line(), 20
er_style_model_ribbonline (er_style_model), 42
er_style_model_ribbonline(), 20, 76
er_style_model_spaghetti (er_style_model), 42
er_style_model_spaghetti(), 12, 21, 32
er_style_quantile, 45
er_style_quantile(), 30, 33
er_style_quantile_errorbar (er_style_quantile), 45
er_style_quantile_errorbar(), 23, 29
er_style_quantile_errorbar_vlines (er_style_quantile), 45
er_style_quantile_errorbar_vlines(), 8, 23
er_style_quantile_pointrange (er_style_quantile), 45
er_style_quantile_pointrange(), 23, 29
er_style_quantile_pointrange_vlines (er_style_quantile), 45
er_style_quantile_pointrange_vlines(), 23
er_style_summary, 50
er_style_summary(), 30, 33
er_style_summary_coefficients (er_style_summary), 50
er_style_summary_coefficients(), 12, 26
er_style_summary_gof (er_style_summary), 50
er_style_summary_gof(), 26
er_style_summary_n (er_style_summary), 50
er_style_summary_n(), 25, 26
er_style_summary_pvalue (er_style_summary), 50
er_style_summary_pvalue(), 12, 25, 26
er_style_tag, 53
er_style_tag(), 16, 18, 31–33, 35, 59, 70
er_style_tte, 58
er_style_tte(), 30, 33, 54, 57, 60, 62, 63, 65, 67, 68, 81, 82, 84, 85, 87
er_style_tte_censor, 59
er_style_tte_censor(), 58, 59
er_style_tte_censor_ticks (er_style_tte_censor), 59
er_style_tte_censor_ticks(), 81
er_style_tte_curve, 61, 89, 91
er_style_tte_curve(), 58, 59
er_style_tte_curve_km (er_style_tte_curve), 61
er_style_tte_curve_km(), 60, 63, 82, 83
er_style_tte_model, 62
er_style_tte_model(), 58, 59
er_style_tte_model_line (er_style_tte_model), 62
er_style_tte_model_line(), 84
er_style_tte_risktable, 64
er_style_tte_risktable(), 58, 59
er_style_tte_risktable_text (er_style_tte_risktable), 64
er_style_tte_risktable_text(), 58, 85, 86, 90
er_style_tte_summary, 65
er_style_tte_summary(), 58, 59
er_style_tte_summary_coefficients (er_style_tte_summary), 65
er_style_tte_summary_coefficients(), 87
er_style_tte_summary_gof (er_style_tte_summary), 65
er_style_tte_summary_gof(), 87
er_style_tte_summary_logrank (er_style_tte_summary), 65
er_style_tte_summary_logrank(), 87
er_style_tte_summary_n (er_style_tte_summary), 65
er_style_tte_summary_n(), 87
er_style_vpc, 69
er_style_vpc(), 30, 33, 54, 57, 72, 74, 77, 78, 94, 95
er_style_vpc_observed, 71
er_style_vpc_observed(), 69, 70, 77, 78, 94
er_style_vpc_observed_mean_errorbar (er_style_vpc_observed), 71
er_style_vpc_observed_mean_errorbar(), 56, 57, 78, 94, 97

er_style_vpc_observed_quantile_errorbar (er_style_vpc_observed), 71
er_style_vpc_observed_quantile_errorbar(), 5, 56, 57, 77, 78, 92, 94
er_style_vpc_observed_quantile_line (er_style_vpc_observed), 71
er_style_vpc_observed_quantile_line(), 56, 57, 77, 78, 92, 94
er_style_vpc_simulated, 75
er_style_vpc_simulated(), 69, 70, 73, 74, 96
er_style_vpc_simulated_mean_errorbar (er_style_vpc_simulated), 75
er_style_vpc_simulated_mean_errorbar(), 56, 73, 95, 97
er_style_vpc_simulated_quantile_errorbar (er_style_vpc_simulated), 75
er_style_vpc_simulated_quantile_errorbar(), 56, 73, 92, 95
er_style_vpc_simulated_quantile_ribbon (er_style_vpc_simulated), 75
er_style_vpc_simulated_quantile_ribbon(), 55, 56, 73, 92, 95
er_summary (er_model_interface), 10
er_summary(), 25, 26, 52, 53, 65, 67, 68, 86, 87
er_tte, 79
er_tte(), 53, 67, 81, 83, 84, 86–91
er_tte_add_censor, 81
er_tte_add_censor(), 58–60, 80, 88
er_tte_add_curve, 82
er_tte_add_curve(), 58, 61, 62, 80, 88
er_tte_add_model, 83
er_tte_add_model(), 11, 13, 58, 62, 63, 80, 90
er_tte_add_risktable, 85
er_tte_add_risktable(), 58, 64, 65, 80, 88, 90
er_tte_add_summary, 86
er_tte_add_summary(), 58, 65, 67, 68, 80, 88, 90
er_tte_build, 88
er_tte_build(), 62–64, 79, 85
er_tte_theme, 89
er_tte_theme(), 64, 65
er_vpc, 91
er_vpc(), 23, 53, 54, 69, 70, 73, 78, 80, 93–98
er_vpc_add_observed, 93
er_vpc_add_observed(), 4–6, 54, 56, 69–72, 92, 93, 95, 96
er_vpc_add_simulated, 94
er_vpc_add_simulated(), 4–6, 11, 12, 54–56, 69, 70, 75, 76, 92–94
er_vpc_build, 96
er_vpc_build(), 91, 93
er_vpc_theme, 96
er_vpc_theme(), 57, 70, 72, 77, 90
erplots_data, 8

ggplot2::coord_cartesian(), 28
ggplot2::draw_key_point(), 28, 90
ggplot2::draw_key_rect(), 28
ggplot2::geom_label(), 48, 52, 67
ggplot2::geom_line(), 63
ggplot2::geom_path(), 43
ggplot2::geom_pointrange(), 23, 47, 48
ggplot2::geom_rect(), 62
ggplot2::geom_ribbon(), 63
ggplot2::geom_step(), 62
ggplot2::position_jitter(), 34
ggplot2::scale_color_brewer(), 28
ggplot2::scale_color_viridis_c(), 28
ggplot2::scale_fill_gradient(), 28
ggplot2::scale_fill_viridis_d(), 28
ggplot2::theme(), 28, 29, 90, 97
ggplot2::theme_bw(), 28, 90, 97
ggplot2::theme_minimal(), 28, 90, 97

stats::quantile(), 7

withr::with_seed(), 39