Package 'erplots'

September 9, 2026

Title Model-Agnostic Exposure-Response Plots

Version 0.1.2

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, tibble, tidyselect, withr

Suggests erglm, 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 8.0.0

Config/Needs/website djnavarro/emaxnls, 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-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-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-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 2026-09-09 15:00:02 UTC

Contents

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 (Conover, *Practical Nonparametric Statistics*): 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.

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` 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 = 4, is_placebo = NULL)

cut_quantile(x, n = 4)
```

Arguments

`x`	Numeric vector
`n`	Number of bins
`is_placebo`	Logical vector indicating placebo samples

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` 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")`.

Value

A factor. `cut_exposure_quantile()`'s result additionally carries a `"breaks"` attribute holding the `n + 1` quantile cutpoints used to form the bins.

Examples

```
x <- rnorm(100)
cut_quantile(x)
cut_exposure_quantile(abs(x))
```

`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)
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.

Usage

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

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

er_summary(model, ...)
```

Arguments

`model`	A fitted exposure-response model object.
`newdata`	A data frame of covariate values at which to predict.
`conf_level`	Confidence level for the prediction interval.

`...`	Passed to methods.
`nsim`	Number of simulation replicates.
`seed`	Optional RNG seed.

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.

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 `...`.

`er_predict()` should return `newdata` with `fit_resp`, `ci_lower`, and `ci_upper`. `er_simulate()` should return `newdata` replicates with `sim_id` and `fit_resp`; it may additionally return `sim_resp` for response-level simulations. `er_summary()` should return `NULL` or a named list with optional keys such as `p_value`, `coefficients`, and `glance`.

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_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.

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, keep_strata = NULL, style = NULL, panel = "both", ...)
```

Arguments

object	Partially constructed plot (has S3 class er_plot).
keep_strata	Logical, indicating whether this layer should be split by the plot's stratification variable; defaults to TRUE if stratify_by was set in er_plot(), FALSE otherwise. See "Details" for how this interacts with a builder's structural family.
style	Function drawing the data layer – defaults to er_style_data_overlay(). Any function matching the standard (data, config, stratify, exposure, response, strata, theme, . signature and tagged with er_style_tag() can be supplied instead; see er_style() and "Details".

`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, unchanged, to `style` when it's called at build time – see `er_style()`'s "Passing extra arguments to a builder" section. Must be named. The built-in `er_style_data_overlay()`/`er_style_data_boxjitter()` builders read a `seed` from here to make their jitter reproducible across repeated `plot()` calls on the same object; omit it (the default) for a fresh random jitter on every render.

Details

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.

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 `"data"`, `er_plot_add_data()` errors informatively; an untagged builder is never checked (only `layout` is a hard requirement).

`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).

Value

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

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,
  bins = NULL,
  keep_strata = 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`	Function drawing each group panel – defaults to `er_style_group_boxplot()`. Applied to every grouping variable added by this call; see `er_style()` and "Details".
`bins`	Number of quantile bins used for continuous grouping variables (`NULL`, the default, uses `cut_quantile()`'s own default).
`keep_strata`	Logical, indicating whether this layer should be split by the plot's stratification variable; defaults to `TRUE` if `stratify_by` was set in `er_plot()`, `FALSE` otherwise. See "Details" for an error case.
`...`	Additional named arguments forwarded, unchanged, to `style` when it's called at build time (identically for every grouping variable added by this call) – see `er_style()`'s "Passing extra arguments to a builder" section. Must be named.

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 `"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.

Value

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

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,
  keep_strata = NULL,
  style = 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().
keep_strata	Logical; whether this layer should use stratification.
style	Function drawing the model curve/ribbon. Defaults to er_style_model_ribbonline().
conf_level	Confidence level for the prediction ribbon.
predict_args	A named list of additional arguments forwarded to er_predict() (e.g. a model-specific argument its er_predict() method requires beyond model/newdata/conf_level). Distinct from ...: predict_args reaches er_predict(), ... reaches style – see "Details".
...	Additional named arguments forwarded unchanged to style at build time.

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.

predict_args and ... serve two different consumers and are kept separate rather than sharing one ...: predict_args is spliced into the er_predict() call (e.g. predict_args = list(landmark_time = 90) for a model whose er_predict() method needs a landmark_time argument with no other slot in the fixed er_predict(model, newdata, conf_level) contract), while ... is forwarded to style alone (see er_style()'s "Passing extra arguments to a builder" section). Reusing a single ... for both would risk a silent name collision if a style builder and a model's er_predict() method happened to share an argument name for unrelated purposes.

Value

The input object, with the model layer added.

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()

# 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,
  keep_strata = NULL,
  style = NULL,
  bins = 4,
  conf_level = 0.95,
  ...
)
```

Arguments

object	Partially constructed plot, an er_plot object.
keep_strata	Logical, indicating whether this layer should be split by the plot's stratification variable; defaults to TRUE if stratify_by was set in er_plot(), FALSE otherwise.
style	Function drawing the quantile summary; defaults to er_style_quantile_errorbar() (point + error bar).
bins	Number of exposure bins (not counting placebo).
conf_level	Confidence level for the interval.
...	Additional named arguments forwarded, unchanged, to style when it's called at build time. Arguments must be named.

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().

Value

The input object, with the quantile layer added.

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,
  keep_strata = NULL,
  style = 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 produced model-based summaries; a purely descriptive builder (e.g. `er_style_summary_n()`) ignores it.
`keep_strata`	Logical, indicating whether this layer should be split by the plot's stratification variable; defaults to `TRUE` if `stratify_by` was set in `er_plot()`, `FALSE` otherwise.
`style`	Function drawing the summary annotation, defaulting to `er_style_summary_pvalue()`.
`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`). Ignored when `model` is `NULL`.
`summary_args`	A named list of additional arguments forwarded to `er_summary()`, distinct from `...` the same way `er_plot_add_model()`'s `predict_args` is distinct from its own `...` – see "Details" there.
`...`	Additional named arguments forwarded, unchanged, to `style` when it's called at build time; see `er_style()`'s "Passing extra arguments to a builder" section. Must be named.

Value

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

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* er_plot *object*

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* er_plot *object*

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.

`draw_key` A key-glyph function (e.g. `ggplot2::draw_key_point()`), passed as every geom's `key_glyph` argument.

`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). 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()`

`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()`)

Arguments are standardised to allow users to write their own 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 `zorder` 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 `"model"`, `"summary"`, `"quantile"`, `"data"`, `"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 = "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_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,
  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.
...	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.
box_width	Width of er_style_data_boxjitter()'s boxplot.
box_alpha	Transparency of er_style_data_boxjitter()'s boxplot fill.
show_outliers	Logical: whether er_style_data_boxjitter() draws outlier points.
jitter_height	Vertical jitter applied to raw points.
jitter_size	Point size for er_style_data_boxjitter()'s jittered points.
jitter_alpha	Transparency of er_style_data_boxjitter()'s jittered points.
alpha	Point transparency for er_style_data_overlay(); fill transparency for er_style_data_hex().
size	Point size for er_style_data_overlay().
bins	Number of hex bins for er_style_data_hex().

Details

Builders for the data layer (er_plot_add_data()) are tagged with the structural family they belong to via er_style_tag(). er_style_data_overlay() and er_style_data_hex() use the overlay layout, drawing in the main panel; er_style_data_boxjitter() uses the panel layout and is binary-response only. All built-in data builders are also tagged layer = "data", so er_plot_add_data() errors if given a builder tagged for another layer.

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, zorder = "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 `er_style()` for the shared builder interface these functions implement.

Value

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

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,
  size = 1,
  inner_range = c(0.25, 0.75),
  outer_range = c(0.05, 0.95),
  alpha_dot = 1,
  alpha_inner = 0.8,
  alpha_outer = 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.
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().
quantiles, quantile_linetype	Violin quantile positions and linetype for er_style_group_violin().
size	Overall size multiplier for er_style_group_linerange()'s dot and lines.
inner_range, outer_range	Quantile probabilities (length 2) for er_style_group_linerange()'s thick and thin lines.
alpha_dot, alpha_inner, alpha_outer	Per-part transparency for er_style_group_linerange()'s dot, inner line, and outer line.
jitter_height, jitter_size, jitter_alpha	Vertical jitter, point size, and transparency for er_style_group_boxjitter()/er_style_group_violin overlaid points.

Details

Builders for the group layer (er_plot_add_groups()) draw exposure distributions for grouping variables. 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. `er_style_group_boxjitter()`/`er_style_group_violinjitter()` are thin wrappers around `er_style_group_boxplot()`/`er_style_group_violin()` 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. All built-in group builders are tagged `layer = "group"`, so `er_plot_add_groups()` errors if given one tagged for another layer.

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

Value

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

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_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
...	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.
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.
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

Builders for the model layer (er_plot_add_model()), which draws the fitted curve (and, where applicable, its uncertainty) over the exposure range: er_style_model_ribbonline() (ribbon plus line, the default), er_style_model_line() (line only, no ribbon), and er_style_model_spaghetti() (a spaghetti plot of simulated draws, for models that implement er_simulate()). All three are tagged er_style_tag(fn, layer = "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()).

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().

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()`.
`errorbar_width`	Width of `er_style_quantile_errorbar()`'s error bars.
`label_size`	Text size for the per-bin value label.
`...`	Additional named arguments forwarded from `er_plot_add_quantiles()`'s own `....`
`vline_colour, vline_linetype`	Colour and linetype of quantile-bin boundary lines.
`vline_labels`	Logical: whether the `_vlines` builders also label each bin boundary with its exposure value.
`vline_label_position`	One of `"auto"`, `"top"`, `"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_inset`	Fraction of the response range `vline_labels` are inset from the panel edge.
`vline_label_digits`	Number of decimal places `vline_labels` are rounded to.
`pointrange_size, pointrange_linewidth`	Size and linewidth for `ggplot2::geom_pointrange()`.

Details

Builders for the `quantile` layer (`er_plot_add_quantiles()`) bin exposure into quantile groups and plot a response summary with an uncertainty interval. `er_style_quantile_errorbar()` and `er_style_quantile_pointrange()` are the base builders; their `_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. All built-in quantile builders are tagged `er_style_tag(fn, layer = "quantile")`, so `er_plot_add_quantiles()` errors informatively if handed a builder tagged for a different layer.

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.

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 `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()`.

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_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.
`label_size`	Label text size.
`label_colour`	Label text colour.
`label_fill`	Label background fill.
`...`	Additional named arguments forwarded from `er_plot_add_model()`'s own `...`.
`fields`	Fields from `glance` to include for `er_style_summary_gof()`.

Details

Builders for er_plot_add_summary() annotate the base panel with a summary statistic or descriptive label. er_style_summary_pvalue() draws a formatted p-value from the model's er_summary() result; er_style_summary_n() draws observation counts and doesn't have to originate from a fitted model at all. er_style_summary_coefficients() draws 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); it draws nothing if coefficients wasn't supplied, or if the layer is stratified. er_style_summary_gof() draws 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\u00b2) 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. All four builders are tagged er_style_tag(fn, layer = "summary"), so er_plot_add_summary() errors informatively if a builder tagged for a different layer is passed to it instead.

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().

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 *Tag a builder with structural/aesthetic metadata*

Description

Attaches the self-declared metadata a custom `er_style_*()`-style function can carry.

Usage

```
er_style_tag(
  style,
  layout = NULL,
  fill_role = NULL,
  y_role = NULL,
  layer = NULL,
  zorder = NULL,
  response_types = NULL,
  plot_by_types = NULL
)
```

Arguments

`style`	A function matching the standard `er_style_*()` signature (see `er_style()`).
`layout`	One of `"overlay"`, `"panel"`, `"categorical"`, or `"continuous"`, or `NULL` (the default) to leave this tag unset. The `"overlay"`/`"panel"` pair is for a data-layer builder (`er_plot_add_data()` documents what each structural family means); the `"categorical"`/`"continuous"` pair is for a VPC observed/simulated builder (`er_vpc_add_observed()`/`er_vpc_add_simulated()`) and marks whether it plots at discrete bin locations or at each bin's numeric exposure midpoint – see `er_style_vpc_observed()`/`er_style_vpc_simulated()`.
`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 `"model"`, `"summary"`, `"quantile"`, `"data"`, `"group"`, `"observed"`, or `"simulated"`, 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. See "Details".
`zorder`	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".

Details

The metadata to be supplied indicate which *structural* family a data-layer builder belongs to (layout), what a builder's fill aesthetic means when it isn't strata (fill_role), what a group-layer builder's y-axis means when it isn't the group variable itself (y_role), which layer a builder is meant to be plugged into (layer), and where an overlay-layout data builder's geoms sit relative to the model/summary/quantile layers when they share the main panel (zorder). All five arguments are optional and independent – pass only the ones a given builder needs, in one call, rather than chaining separate setters.

layout is a required tag for a data-layer builder: 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")

For a VPC observed/simulated builder, layout is optional but, when present on both the observed and simulated builder passed to a given er_vpc object, is checked for agreement: er_vpc_add_simulated() errors if the simulated builder's layout ("categorical", discrete bin locations; or "continuous", numeric bin-midpoint locations, e.g. er_style_vpc_simulated_quantile_ribbon()) disagrees with the observed builder's own. 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 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.

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.

layer is also optional, but unlike fill_role/y_role it isn't read for labelling. It's read by every er_plot_add_*() function (er_plot_add_model() checks style against "model"; er_plot_add_summary() checks style against "summary"; er_plot_add_quantiles() against "quantile"; er_plot_add_data() against "data"; er_plot_add_groups() against "group") to catch a builder plugged into the wrong layer – e.g. passing a quantile builder to er_plot_add_data() – 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. A custom builder that omits it is never checked: layer is opt-in, not a requirement like layout is for a data-layer builder.

zorder 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. `zorder` 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.

`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).

Value

`style`, with whichever of the `"er_style_layout"`/`"er_style_fill_role"`/`"er_style_y_role"`/`"er_style_layer"`/`"er_style_zorder"`/`"er_style_response_types"`/`"er_style_plot_by_types"` attributes were requested attached.

See Also

`er_plot_add_data()`, `er_style()`

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 = "data"
)
```

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,
  ...
)
```

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.
...	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.

Details

er_style_vpc_observed_mean_errorbar() (the default) 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 layout tag – pair it with er_style_vpc_simulated_mean_errorbar(), which mirrors the same adaptive logic.

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.

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$percentiles) on plot_by's own numeric scale when plot_by is numeric. Because it adapts its x-position family at build time rather than declaring one statically, it carries no layout tag. Unlike er_style_vpc_observed_quantile_line()/ er_style_vpc_simulated_quantile_ribbon(), it supports a categorical plot_by as well as a numeric one; like it, it requires a continuous/count response (a binary response's distribution is already fully described by its rate) and errors informatively without config$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.

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.

Value

A list of geoms; see er_style().

See Also

er_style(), er_style_vpc_simulated()

---

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,
  ...
)
```

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.

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.

errorbar_width — Width of er_style_vpc_simulated_mean_errorbar()'s and er_style_vpc_simulated_quantile_er 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.

Details

er_style_vpc_simulated_mean_errorbar() (the default) 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 `layout` tag – pair it with `er_style_vpc_observed_mean_errorbar()`, which mirrors the same adaptive logic.

`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.

`er_style_vpc_simulated_quantile_errorbar()` plots `config$percentiles` – a point + across-replicate percentile interval for each requested percentile – for pairing with `er_style_vpc_observed_quantile_errorbar(` Like that builder (and like `er_style_vpc_simulated_mean_errorbar()`), it adapts its x-position to `plot_by`'s type, carries no `layout` tag, supports both a numeric and a categorical `plot_by`, and requires a continuous/count response, erroring informatively without `config$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.

`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.

Value

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

See Also

`er_style()`, `er_style_vpc_observed()`

`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,
  response_type = "auto",
  plot_by = NULL,
  n_bins = 4,
  stratify_by = NULL,
  n_strata = 4,
  conf_level = 0.95,
  probs = c(0.1, 0.5, 0.9)
)
```

Arguments

data	Data frame or tibble containing the observed data.
exposure	Exposure variable (one variable, unquoted).
response	Response variable (one variable, unquoted).
response_type	One of "auto", "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.
stratify_by	Optional variable (unquoted) splitting the VPC into one facet panel per level, via ggplot2::facet_wrap(). A categorical variable is used as-is; a numeric variable is automatically split into n_strata quantile bins (placebo, i.e. 0, kept in its own bin when stratify_by is the exposure variable itself), with a message reporting that this happened. Must resolve to a different variable than plot_by. Defaults to NULL (no faceting, a single panel, matching prior behaviour).
n_strata	Number of quantile bins, when stratify_by is numeric. Ignored when stratify_by is NULL or categorical.
conf_level	Confidence level for both the observed- and simulated-side intervals. Must be strictly between 0 and 1.
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.

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).

Unlike er_plot(), er_vpc() has no stratification concept and always renders a single panel – see er_vpc_add_observed() for plot_by, the (orthogonal) 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 an* er_vpc *VPC*

Description

Bins the observed data by plot_by (see er_vpc()) and computes its response summary (rate/mean + confidence interval, plus empirical percentiles for a continuous/count response), for later comparison against a simulated layer added via er_vpc_add_simulated().

Usage

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

Arguments

object	Partially constructed VPC (has S3 class er_vpc).
style	A function determining how the observed layer is drawn; see er_style_vpc_observed().
...	Additional named arguments forwarded to style.

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.

See Also

er_vpc(), er_vpc_add_simulated(), er_style_vpc_observed()

---

er_vpc_add_simulated *Add the simulated-data layer to an* er_vpc *VPC*

Description

Bins simulated data using the same cutpoints er_vpc_add_observed() already computed, and summarizes it (mean + a percentile interval across replicates, plus simulated percentile bands for a continuous/count response).

Usage

```
er_vpc_add_simulated(
  object,
  model = NULL,
  sim = NULL,
  nsim = 100,
  seed = NULL,
  style = er_style_vpc_simulated_mean_errorbar,
  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.
nsim	Number of simulation replicates, only used with model.
seed	Optional RNG seed, only used with model.
style	A function determining how the simulated layer is drawn; see er_style_vpc_simulated().

simulate_args	A named list of additional arguments forwarded to er_simulate(), only used with model. Distinct from ... the same way er_plot_add_model()'s predict_args is distinct from its own ... – see that function's "Details" for the rationale.
...	Additional named arguments forwarded to style.

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.

See Also

er_vpc(), er_vpc_add_observed(), er_style_vpc_simulated()

---

er_vpc_build *Build and render an* er_vpc *object*

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 an* er_vpc *object*

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 format the rate/mean displayed in the observed/simulated summaries for a binary response (format_percent) or a continuous/count response (format_number).

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 (see the "Gotchas" section of AGENTS.md) 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