

Package ‘easyViz’

January 24, 2026

Title Easy Visualization of Conditional Effects from Regression Models

Version 2.0.1

Description Offers a flexible and user-friendly interface for visualizing conditional effects from a broad range of regression models, including mixed-effects and generalized additive (mixed) models. Compatible model types include `lm()`, `rlm()`, `glm()`, `glm.nb()`, `betareg()`, and `gam()` (from 'mgcv'); nonlinear models via `nls()`; generalized least squares via `gls()`; and survival models via `coxph()` (from 'survival'). Mixed-effects models with random intercepts and/or slopes can be fitted using `lmer()`, `glmer()`, `glmer.nb()`, `glmmTMB()`, or `gam()` (from 'mgcv', via smooth terms). Plots are rendered using base R graphics with extensive customization options. Approximate confidence intervals for `nls()` and `betareg()` models are computed using the delta method. Robust standard errors for `rlm()` are computed using the sandwich estimator (Zeileis 2004) <[doi:10.18637/jss.v011.i10](https://doi.org/10.18637/jss.v011.i10)>. For beta regression using 'betareg', see Cribari-Neto and Zeileis (2010) <[doi:10.18637/jss.v034.i02](https://doi.org/10.18637/jss.v034.i02)>. For mixed-effects models with 'lme4', see Bates et al. (2015) <[doi:10.18637/jss.v067.i01](https://doi.org/10.18637/jss.v067.i01)>. For models using 'glmmTMB', see Brooks et al. (2017) <[doi:10.32614/RJ-2017-066](https://doi.org/10.32614/RJ-2017-066)>. Methods for generalized additive models using 'mgcv' follow Wood (2017) <[doi:10.1201/9781315370279](https://doi.org/10.1201/9781315370279)>.

Maintainer Luca Corlatti <lucac1980@yahoo.it>

Imports stats, utils, graphics, grDevices

Suggests MASS, sandwich, nlme, numDeriv, betareg, statmod, survival, lme4, glmmTMB, mgcv

License GPL-3

Encoding UTF-8

RoxygenNote 7.3.3

NeedsCompilation no

Author Luca Corlatti [aut, cre]

Repository CRAN

Date/Publication 2026-01-24 15:00:02 UTC

Contents

easyViz	2
---------	---

easyViz

*Easy Visualization of Conditional Effects from Regression Models***Description**

easyViz offers a flexible and user-friendly interface for visualizing conditional effects from a broad range of regression and mixed-effects models using base R graphics.

Usage

```
easyViz(
  model,
  data,
  predictor,
  by = NULL,
  by_breaks = NULL,
  pred_type = "response",
  pred_range_limit = TRUE,
  pred_on_top = FALSE,
  pred_resolution = 101,
  num_conditioning = "median",
  cat_conditioning = "mode",
  fix_values = NULL,
  re_form = NULL,
  rlm_vcov = "HC0",
  backtransform_response = NULL,
  xlim = NULL,
  ylim = NULL,
  xlab = NULL,
  ylab = NULL,
  cat_labels = NULL,
  font_family = "sans",
  las = 1,
  bty = "o",
  plot_args = list(),
  show_data_points = TRUE,
  binary_data_type = "plain",
  bins = 10,
  jitter_data_points = FALSE,
  point_col = rgb(0, 0, 0, alpha = 0.4),
  point_pch = 16,
  point_cex = 0.75,
  pred_line_col = "black",
  pred_line_lty = c(1, 2, 3, 4, 5, 6),
  pred_line_lwd = 2,
  ci_level = 0.95,
```

```

  ci_type = "polygon",
  ci_polygon_col = c("gray", "black", "lightgray", "darkgray", "gray", "black"),
  ci_polygon_alpha = 0.5,
  ci_line_col = "black",
  ci_line_lty = c(1, 2, 3, 4, 5, 6),
  ci_line_lwd = 1,
  pred_point_col = c("black", "gray", "darkgray", "lightgray", "black", "gray"),
  pred_point_pch = 16,
  pred_point_cex = 1,
  ci_bar_col = "black",
  ci_bar_lty = 1,
  ci_bar_lwd = 1,
  ci_bar_caps = 0.1,
  add_legend = NULL,
  legend_position = "out",
  legend_horiz = FALSE,
  legend_title = NULL,
  legend_labels = NULL,
  legend_title_size = 1,
  legend_label_size = 0.9,
  legend_args = list(),
  show_conditioning = FALSE,
  plot = TRUE
)

```

Arguments

model	[required] A fitted model object (e.g., <code>model = your.model</code>). Supported models include a wide range of regression types, including linear, robust linear, non-linear, generalized least squares, generalized linear, survival, mixed-effects, and generalized additive (mixed) models. Compatible model-fitting functions include: <code>stats::lm</code> , <code>MASS::rlm</code> , <code>nlme::gls</code> , <code>stats::nls</code> , <code>stats::glm</code> , <code>MASS::glm.nb</code> , <code>betareg::betareg</code> , <code>survival::coxph</code> , <code>lme4::lmer</code> , <code>lme4::glmer</code> , <code>lme4::glmer.nb</code> , <code>glmmTMB::glmmTMB</code> , and <code>mgcv::gam</code> .
data	[required] The data frame used to fit the model (e.g., <code>data = your.data</code>). This data frame is used internally for generating predictions. <i>All variables used in the model formula (including predictors, offset variables, grouping variables, and interaction terms) must be present in this data frame</i> . If the model was fitted without using a <code>data</code> argument (e.g., using variables from the global environment), you must ensure that <code>data</code> includes all required variables. Otherwise, prediction may fail or produce incorrect results.
predictor	[required] The name of the target explanatory variable to be plotted (e.g., <code>predictor = "x1"</code>).
by	The name of an interaction or additional variable for conditioning (e.g., <code>by = "x2"</code>). If supplied, <code>easyViz()</code> conditions predictions on <code>by</code> as follows: <ul style="list-style-type: none"> • Categorical (factor/character): a separate prediction line (or point) is plotted for each level.

- Numeric with few distinct values (default: ≤ 6 unique values): the numeric values are treated as discrete groups, and a separate prediction line is plotted for each value. This is useful for variables coded as numbers (e.g., 0/1, 1–5, small ordinal scales).
- Numeric with many distinct values (default: > 6 unique values): by is treated as continuous and predictions are shown at representative values (by default the 10th, 50th, and 90th percentiles), unless overridden by by_breaks.
- Grouping variable in random effects: if by corresponds to a variable used as a grouping term (as in, e.g., `(1|group)` or `s(group, bs="re")`) and `re_form = NULL`, predictions are conditional on group-specific random effects.

Although easyViz does not natively support direct visualization of three-way interactions in a multi-panel plot, this can be easily achieved by combining the by and fix_values arguments. For example, if your model includes a term like `x1*x2*x3`, you can visualize the effect of `x1` across levels of `x2` by setting `predictor = "x1"`, `by = "x2"`, and fixing `x3` at a specific value using `fix_values = c(x3 = ...)`. Repeating this with different values of `x3` produces multiple plots that can be arranged to visualize the full three-way interaction. See the Examples section for a demonstration of how to apply this approach.

<code>by_breaks</code>	Optional numeric vector specifying the values of a numeric conditioning variable to include in the plot (e.g., <code>by_breaks = c(-2, 0, 2)</code> or <code>by_breaks = quantile(your.data\$x2, c(0.25, 0.5, 0.75))</code>). For numeric by, this selects the cross-sections to plot and overrides the default quantiles (0.1, 0.5, 0.9). Ignored if by is categorical.
<code>pred_type</code>	Character string indicating the type of predictions to plot. Either "response" (default), which returns predictions on the original outcome scale by applying the inverse of the model's link function (e.g., probabilities for binary models), or "link", which returns predictions on the linear predictor (link) scale (e.g., log-odds, log-counts, or other transformed scales depending on the model). For <i>survival models</i> (<code>coxph</code>), <code>pred_type = "link"</code> returns predictions on the linear predictor scale (log-hazard ratio), while <code>pred_type = "response"</code> returns hazard ratios. Survival probabilities are not produced because they require a time point and the baseline hazard.
<code>pred_range_limit</code>	Logical. Applies only when the predictor is numeric and a categorical by variable is specified. If TRUE (default), the prediction range for each level of the by variable is limited to the range of the predictor observed within that level. This avoids extrapolating predictions beyond the available data for each subgroup. If FALSE, predictions span the entire range of the predictor across all levels of the by variable. If the by variable is numeric, <code>pred_range_limit</code> is automatically set to FALSE, since numeric by values are treated as continuous rather than grouping factors.
<code>pred_on_top</code>	Logical. If TRUE, prediction lines (and their confidence intervals) for numeric predictors are drawn after raw data, so they appear on top. Default is FALSE, which draws predictions underneath the data. This has no effect for categorical predictors — for those, predictions are always drawn on top of raw data.
<code>pred_resolution</code>	Number of prediction points to use for numeric predictors. Defaults to 101,

consistent with `visreg`. The default should work well in most cases. Increasing `pred_resolution` may be particularly helpful when the predictor spans a wide range or when visualizing nonlinear relationships (e.g., splines or polynomials), to ensure smooth and accurate rendering of the effect. **Note:** A higher value may slightly increase computation time, especially when combined with many levels of a by variable.

`num_conditioning`

How to condition non-target numeric predictors. Either "median" (default) or "mean". This determines how numeric variables that are not directly plotted are held constant during prediction, while varying the predictor of interest. To fix specific variables at custom values instead, use the `fix_values` argument.

`cat_conditioning`

How to condition non-target categorical predictors. Either "mode" (default) or "reference". As for "num_conditioning", conditioning means holding these variables constant while varying the predictor of interest. If multiple levels are equally frequent when "mode" is selected, the level chosen will be the first in the factor's level order (which by default is alphabetical and typically coincides with the reference level, unless explicitly re-leveled). This behavior also applies to grouping variables used as random effects when `re_form` = `NULL`. To fix categorical variables (including grouping variables) at specific levels, use `fix_values`.

`fix_values`

A named vector or named list specifying fixed values for one or more variables during prediction. Supports both numeric and categorical variables. For numeric variables, specify a fixed value (e.g., `fix_values = c(x = 1)`). For categorical variables, provide the desired level as a character string or factor (e.g., `fix_values = c(group = "levelA")`). Multiple values should be provided as a list (e.g., `fix_values = list(x = 1, group = "levelA")`). This overrides the default conditioning behavior specified via `num_conditioning` and `cat_conditioning`. This argument also applies to *grouping variables used as random effects*: when `re_form` = `NULL`, predictions are conditional on the level specified in `fix_values`; if not specified, the level is chosen based on `cat_conditioning`. This argument is also useful for *setting offset variables*. For count models with a log link in which the offset is specified as `offset(log(exposure))`, `easyViz` interprets the model as a rate model and, by default, fixes the exposure variable to 1 in the prediction grid. Raw response values are correspondingly scaled by the exposure so that both data points and predictions are displayed on the same unit-rate scale (e.g., detections per day; see `show_data_points` for more details). To obtain predictions on a different rate scale, fix the exposure at the desired value using `fix_values`. If the offset is specified outside the model formula, for `gam()` and `glmer()` models it is treated as `offset = 0` during prediction (i.e., `exposure = 1` for log-link models). This matches `easyViz`'s default behavior, but in such cases `easyViz` cannot vary the exposure. Include the offset inside the formula (e.g., `offset(log(exposure))`) to enable full control. If the default behavior is not desired (i.e., if you do not want automatic rate standardization or fixing exposure to 1), you may still use an offset by specifying a pre-transformed exposure variable (e.g., `log_exposure = log(exposure)` and `offset(log_exposure)`). In this case, `easyViz` treats the offset as a generic additive term on the link scale. Model predictions are computed correctly, but the

offset variable is conditioned using the default rules for numeric covariates (see `num_conditioning`) or values supplied via `fix_values`. Because the exposure structure is no longer explicit, raw data are plotted on the original response scale and no automatic rate standardization is applied. **Additional uses:** `fix_values` is useful also for forcing predictions at specific values or ensuring consistent conditioning across models, for example when you want to visualize the effect of a predictor at a specific level of an interacting variable, without conditioning on all levels. E.g., to plot the conditional effect of a continuous predictor `x1` at a specific value of another variable `x2` (numeric or categorical), simply set `fix_values = c(x2 = ...)` and omit the `by` argument. This creates a clean single-effect plot for `x1` at the desired level of `x2`, without plotting multiple lines or groups as `by` would. This argument can also be used to visualize three-way interactions when combined with `by`. See the `by` argument for details, and the Examples section for a demonstration of how to apply this approach.

`re_form`

A formula specifying which random effects to include when generating predictions:

- `re_form = NULL` (default): produces group-specific predictions, conditional on the random-effect levels present in the data. By default, `easyViz` fixes grouping variables at their mode (i.e., the most frequent level), so the prediction reflects the conditional estimate for that group. You can override this by explicitly fixing the grouping variable via `fix_values` (e.g., `fix_values = c(group = "levelA")`). If all levels are equally frequent and no value is specified, the first level (in factor order) is used, which is usually alphabetical unless re-leveled. If `by` corresponds to a grouping variable used in a random effect, predictions are visualized for all group levels (i.e., conditional predictions).
- `re_form = NA` or `re_form = ~0`: produces population-level (i.e., marginal) predictions by excluding random effects from the prediction step. The random effects are still part of the fitted model and influence the estimation of fixed effects and their uncertainty, but they are not included when computing predicted values. This is equivalent to assuming random effects are zero — representing an "average" group or subject.

This argument is relevant for mixed-effects models only (e.g., from `lme4`, `glmmTMB`, or `mgcv:::gam()`). For `mgcv:::gam()` models, random effects can be modeled using smooth terms like `s(group, bs = "re")`. Although `predict.gam()` does not support a `re.form` argument, `easyViz` emulates its behavior: `re_form = NULL` includes random-effect smooths, while `re_form = NA` or `~0` excludes them via the `exclude` argument in `predict.gam()`. **Note:** For models fitted with `lme4` (e.g., `lmer()`, `glmer()`), standard errors are not available when `re_form = NULL`.

`r1m_vcov`

Robust variance type for `MASS:::r1m` models. May be one of the sandwich types: "HC0", "HC1", "HC2", "HC3", "HC4", "HC4m", "HC5", "const", "HC". Alternatively, users may provide a covariance matrix (used directly), or a function `f(model)` returning a covariance matrix. Default is "HC0".

`backtransform_response`

A custom function to back-transform predictions for transformed response variables (e.g., `exp` for log-transformed responses, or `function(x) x^2` for square

root-transformed responses). **Note:** For log-transformed responses, recall that in lognormal models the mean on the original scale is not simply $\exp(x)$ due to Jensen's inequality. If you want the expected value of a lognormal response, use a function such as `function(x) exp(x + sigma2/2)`, where `sigma2` is the residual variance on the log scale (e.g., `sigma2 <- sigma(your.model)^2`). **Note:** If you wish to model a transformed response, it is recommended to apply the transformation directly in the model formula (e.g., `log(y) ~ ...`), rather than modifying the response variable in the data set. This ensures that observed data points are correctly plotted on the original (back-transformed) scale. Otherwise, raw data and predicted values may not align properly in the plot.

`xlim` x-axis limits for the plot (e.g., `xlim = c(0, 10)`). Defaults to automatic scaling based on the data range. Applies to both numeric and categorical predictors. For categorical variables, x-axis positions are treated as integer values (e.g., 1, 2, ..., k), and adjusting `xlim` (e.g., `xlim = c(0.5, k + 0.5)`) can control spacing and margins around the plotted levels.

`ylim` y-axis limits for the plot (e.g., `ylim = c(10, 20)`). Defaults to automatic scaling based on the data and prediction range.

`xlab` x-axis labels (e.g., `xlab = "x"`). Defaults to "predictor".

`ylab` y-axis labels (e.g., `ylab = "y"`). Defaults to "response".

`cat_labels` Custom labels for levels of a categorical predictor (e.g., `cat_labels = c("Level A", "Level B", "Level C")`).

`font_family` Font family for the plot. E.g., "sans" (default), "serif", "mono".

`las` Text orientation for axis labels (default: 1).

`bty` Box type around the plot. E.g., "o" (default), "n", "L".

`plot_args` A named list of additional graphical parameters passed to base R's `plot()` function. These arguments allow users to override default appearance settings in a flexible way. Common options include axis label size, color, label text, tick mark spacing, and coordinate scaling. **Note:** Only arguments recognized by `plot.default()` are supported. Parameters that must be set via `par()` (such as `mar`, `oma`, `mfrow`, `mpg`) are *not* applied through `plot_args`. If you wish to adjust those settings, set them directly using `par()` before calling `easyViz()`. Many valid parameters are documented in both `?plot.default` and `?par`. In `plot_args`, they are passed to `plot()`, not to `par()`. Common `plot()` parameters you may override:

- Label/Text size and style: `cex.lab`, `cex.axis`, `cex.main`, `font.lab`, `font.axis`, `font.main`.
- Colors: `col.lab`, `col.axis`, `col.main`, `col.sub`, `col`, `bg`, `fg`.
- Label/Text content: `xlab`, `ylab`, `main`, `sub`.
- Box and axis rendering: `bty`, `axes`, `frame.plot`, `ann`.
- Coordinate settings and tick spacing: `xlim`, `ylim`, `xaxs`, `yaxs`, `xaxp`, `yaxp`, `asp`, `xlog`, `ylog`.

For a full list of supported parameters, see `?plot.default` and `?par`. Example usage:

```
plot_args = list(main = "Title", cex.lab = 1.2, col.axis = "gray40", xaxp = c(0, 10, 5)).
```

`show_data_points`

Logical. Whether to display raw data points (default: TRUE). For *binomial models* where the response is expressed as `cbind(successes, failures)` or as a proportion `successes / trials`, the raw data points shown on the y-axis are plotted as proportions: `successes / (successes + failures)` or `successes / trials`, respectively. For *count models with a log link* that include an offset specified as `offset(log(exposure))`, `easyViz` interprets the model as a rate model. Raw response values are rescaled as `(count / exposure) * exposure_ref` and, by default, `exposure_ref = 1`, so points are displayed on the unit-rate scale (e.g., detections per day). The prediction grid uses the same reference exposure value, ensuring that points and predictions are on the same scale. To use a different rate scale, set the exposure reference value via `fix_values` (e.g., `fix_values = c(exposure = 7)` for detections per 7 days). If this default behavior is not desired, the offset can instead be specified using a pre-transformed exposure variable (e.g., `log_exposure = log(exposure)` and `offset(log_exposure)` in the model formula). In this case, `easyViz` does not apply automatic rate standardization and treats the offset as a generic additive term on the link scale (see `fix_values` for details). For *survival models* (`coxph`), raw data points are not displayed, because survival outcomes involve event times and censoring and are not directly comparable to the plotted linear predictor (or hazard ratio) scale.

`binary_data_type`

For binary responses, how to display raw data points in the plot. Either "plain" (default), which plots each individual 0/1 observation as-is, or "binned", which groups observations into intervals (bins) of the predictor and plots the proportion of 0s and 1s within each bin. This makes it easier to visualize trends in binary outcomes, especially when many points overlap.

`bins`

Number of bins for displaying binary response raw data when `binary_data_type = "binned"` (default: 10).

`jitter_data_points`

Logical. If TRUE, raw data points are jittered horizontally to reduce overplotting. Applies to both categorical and numeric predictors. Default is FALSE. For categorical predictors, jittering helps distinguish overlapping points. For numeric predictors, it can be useful when many data points share the same x-value (e.g., integers or rounding).

`point_col`

Point color for raw data (default: `rgb(0, 0, 0, alpha = 0.4)`). Can be specified as a color name (e.g., "gray"), an integer (e.g., 1), or an RGB (e.g., `rgb(0, 0, 0, alpha = 0.4)`) or hex string (e.g., "#808080"). When the focal predictor is numeric, raw data points are plotted at the observation level. This typically matters when visualizing interactions involving a grouping variable (via `by`). In this case, `point_col` must be either a single value (applied to all points) or a vector of length equal to the number of observations in the data supplied to `easyViz()` (e.g., generated via `ifelse(group == "levelA", "blue", "red")` or similar logic; see the Example section). When the focal predictor is categorical and points are plotted for different levels of a grouping variable (via `by`), `point_col` can be a vector of colors, with one color per group (e.g., `point_col = c("blue", "red")`; see the Example section).

Tip: For large data sets with many overlapping data points, it is recommended to use semi-transparent colors to reduce overplotting. You can achieve this by set-

	ting a low alpha value (e.g., <code>rgb(1, 0, 0, alpha = 0.1)</code>), or by using <code>adjustcolor()</code> with the argument <code>alpha.f</code> (e.g., <code>adjustcolor("red", alpha.f = 0.1)</code>). In such cases, consider setting <code>pred_on_top = TRUE</code> to ensure that prediction lines and confidence intervals remain clearly visible above the dense cloud of raw data points.
<code>point_pch</code>	Point shape for raw data (default: 16). Dynamic: accepts multiple values when points are plotted for different values/levels of a variable. The same grouping logic described for <code>point_col</code> applies.
<code>point_cex</code>	Point size for raw data (default: 0.75). Dynamic: accepts multiple values when points are plotted for different values/levels of a variable. The same grouping logic described for <code>point_col</code> applies.
<code>pred_line_col</code>	Color of the predicted line for numeric predictors (default: "black"). Can be specified as a color name, number or RGB/hex string. Dynamic: accepts multiple values (e.g., <code>c("red", "green", "blue")</code>) when multiple lines are plotted (i.e., when <code>by</code> is specified).
<code>pred_line_lty</code>	Type of the predicted line for numeric predictors (default: 1). Dynamic: accepts multiple values (e.g., <code>c(1, 2, 3)</code>) when multiple lines are plotted (i.e., when <code>by</code> is specified).
<code>pred_line_lwd</code>	Width of the predicted line for numeric predictors (default: 2). Dynamic: accepts multiple values (e.g., <code>c(1, 2, 3)</code>) when multiple lines are plotted (i.e., when <code>by</code> is specified).
<code>ci_level</code>	Confidence level for the intervals (between 0 and 1). Defaults to 0.95. For example, <code>ci_level = 0.85</code> plots 85 percent confidence intervals.
<code>ci_type</code>	Type of confidence intervals for numeric predictors. Either "polygon" (default) to draw shaded confidence bands, "lines" to draw lines, or NULL to suppress confidence intervals for numeric predictors. Note: <code>ci_type = NULL</code> does <i>not</i> suppress confidence bars for categorical predictors; these are always shown unless manually suppressed via custom logic (e.g., by setting <code>ci_bar_lwd = 0</code>).
<code>ci_polygon_col</code>	Color for confidence interval polygon (default: "gray"). Requires <code>ci_type = "polygon"</code> . Can be specified as a color name, number or RGB/hex string. Dynamic: accepts multiple values (e.g., <code>c("red", "green", "blue")</code>) when CIs are plotted for multiple lines (i.e., when <code>by</code> is specified).
<code>ci_polygon_alpha</code>	Numeric value between 0 and 1 controlling the transparency of confidence interval bands when <code>ci_type = "polygon"</code> . Default is 0.5. Higher values make the band more opaque; lower values make it more transparent.
<code>ci_line_col</code>	Color for confidence interval lines (default: "black"). Requires <code>ci_type = "lines"</code> . Can be specified as a color name, number or RGB/hex string. Dynamic: accepts multiple values (e.g., <code>c("red", "green", "blue")</code>) when CIs are plotted for multiple lines (i.e., when <code>by</code> is specified).
<code>ci_line_lty</code>	Type for confidence interval lines (default: 1). Requires <code>ci_type = "lines"</code> . Dynamic: accepts multiple values (e.g., <code>c(1, 2, 3)</code>) when CIs are plotted for multiple lines (i.e., when <code>by</code> is specified).
<code>ci_line_lwd</code>	Width for confidence interval lines (default: 1). Requires <code>ci_type = "lines"</code> . Dynamic: accepts multiple values (e.g., <code>c(1, 2, 3)</code>) when CIs are plotted for multiple lines (i.e., when <code>by</code> is specified).

<code>pred_point_col</code>	Color for predicted point values of categorical predictors (default: "black"). Can be specified as a color name, number or RGB/hex string. When <code>by</code> is specified (interaction plots), <code>pred_point_col</code> may be a vector with one color per group (i.e., per level/value of <code>by</code>); the same group color is then used for predicted points across all levels of the focal predictor.
<code>pred_point_pch</code>	Shape for predicted point values of categorical predictors (default: 16). Dynamic: accepts multiple values (e.g., <code>c(1, 2, 3)</code>) when points are plotted for an interaction (i.e., when <code>by</code> is specified). The same grouping logic described for <code>pred_point_col</code> applies.
<code>pred_point_cex</code>	Size for predicted point values of categorical predictors (default: 1). Dynamic: accepts multiple values (e.g., <code>c(1, 2, 3)</code>) when points are plotted for an interaction (i.e., when <code>by</code> is specified). The same grouping logic described for <code>pred_point_col</code> applies.
<code>ci_bar_col</code>	Color for confidence interval bars (default: "black"). Applies only when the predictor is categorical. Can be a single color (applied to all CI bars) or a vector of colors. When <code>by</code> is used, CI bars are drawn by looping first over the levels of the focal predictor and then over the levels of the grouping variable. Colors are assigned following this order, so they may need to be repeated to match predictor levels within each group. For example, with 2 levels of <code>x</code> and 2 groups for <code>by</code> , four CI bars are drawn, and a length-4 vector can be used to assign colors to each bar (e.g., <code>c("blue", "blue", "red", "red")</code> ; see the Examples section).
<code>ci_bar_lty</code>	Type for confidence interval bars (default: 1). Applies only when the predictor is categorical. Follows the same assignment logic as <code>ci_bar_col</code> .
<code>ci_bar_lwd</code>	Width for confidence interval bars (default: 1). Applies only when the predictor is categorical. Follows the same assignment logic as <code>ci_bar_col</code> . To suppress confidence interval bars, set <code>ci_bar_lwd</code> = 0 (line width of zero).
<code>ci_bar_caps</code>	Size of the caps on confidence interval bars (default: 0.1). Applies only when the predictor is categorical. Follows the same assignment logic as <code>ci_bar_col</code> . Increase for more visible caps, set to 0 to remove caps and draw plain vertical bars.
<code>add_legend</code>	Logical. Whether to draw a legend for the <code>by</code> variable. By default, a legend is drawn automatically (i.e., <code>add_legend</code> = TRUE) when <code>by</code> is supplied and omitted otherwise. Set <code>add_legend</code> = FALSE to suppress the legend even when <code>by</code> is present.
<code>legend_position</code>	Legend position. Either a named position string ("top", "bottom", "left", "right", "topleft", "topright", "bottomleft", "bottomright"), the special keyword "out", or a numeric vector <code>c(x, y)</code> specifying exact coordinates for manual placement. When a <code>by</code> variable is specified, a legend is drawn automatically with default position "out", i.e., a horizontal legend is drawn above the plotting region. All other values follow standard base R legend positioning rules. Advanced manual placement outside the plot region is possible by temporarily increasing margins with <code>par(mar = ...)</code> and/or allowing drawing outside the plot region with <code>par(xpd = TRUE)</code> , then adjusting the legend position using <code>inset</code> or explicit coordinates. For example, to place a legend to the right of the axes, you may need to increase the right margin (e.g., <code>par(mar = c(5,</code>

4, 4, 8))) and set `par(xpd = TRUE)` before calling `easyViz()`. You can then fine-tune placement using `inset` via `legend_args` when `legend_position` is a keyword (i.e., "topright", "right", "bottomright"). If `legend_position` is given as explicit coordinates `c(x, y)`, `inset` is not used because the legend is positioned directly at `(x, y)`. See the Examples section for demonstrations.

<code>legend_horiz</code>	Logical. If TRUE, the legend is drawn horizontally (side-by-side). If FALSE, the legend is drawn vertically (stacked). Note: when <code>legend_position = "out"</code> , <code>easyViz</code> may automatically draw the legend horizontally (and adjust legend settings) to improve readability, unless overridden by user-supplied <code>legend_args</code> .
<code>legend_title</code>	Optional character string controlling the legend title. If <code>legend_title</code> is <i>not specified</i> , and a <code>by</code> variable is present, <code>easyViz()</code> automatically uses the name of the <code>by</code> variable as the legend title, and legend labels correspond to the levels of <code>by</code> (e.g., "A", "B", "C"). If <code>legend_title</code> is explicitly set to NULL, no legend title is drawn, and legend labels revert to the verbose form "by = level" (e.g., "group = A"). If <code>legend_title</code> is a character string, it is used as the legend title, and legend labels correspond to the levels of <code>by</code> . In all cases, legend labels can be manually overridden using <code>legend_labels</code> .
<code>legend_labels</code>	Custom labels for the legend (e.g., <code>legend_labels = c("Level A", "Level B", "Level C")</code>).
<code>legend_title_size</code>	Numeric. Text size for the legend title (default: 1).
<code>legend_label_size</code>	Numeric. Text size for the legend labels (default: 0.9).
<code>legend_args</code>	A named list of additional arguments passed to base R's <code>legend()</code> function. These allow fine-tuned control over the appearance and placement of the legend and override the high-level options provided by <code>legend_position</code> , <code>legend_title</code> and other <code>legend_*</code> arguments. For example, you can adjust the legend's box style, border color, spacing, point size, or background color. Common options include: <ul style="list-style-type: none"> • Point and line appearance: <code>pch</code>, <code>col</code>, <code>pt.cex</code>, <code>pt.lwd</code>, <code>lty</code>, <code>lwd</code>. • Layout and spacing: <code>ncol</code>, <code>x.intersp</code>, <code>y.intersp</code>, <code>inset</code>, <code>xjust</code>, <code>yjust</code>. • Text style and color: <code>cex</code>, <code>text.col</code>, <code>font</code>, <code>adj</code>. • Box and background: <code>bty</code>, <code>box.lwd</code>, <code>box.col</code>, <code>bg</code>. • Title control: <code>title</code>, <code>title.col</code>, <code>title.cex</code>, <code>title.adj</code>. For a full list of supported parameters, see <code>?legend</code> . Example usage: <code>legend_args = list(bty = "o", box.col = "black", pt.cex = 1.5)</code> . Tip: Legends can be pushed outside the plotting region by combining <code>par(xpd = TRUE)</code> and wider margins (e.g., via <code>par(mar = ...)</code>), and by supplying appropriate coordinates or negative <code>inset</code> values through <code>legend_args</code> .
<code>show_conditioning</code>	Logical. If TRUE, <code>easyViz</code> prints a concise summary in the R console describing how predictions are conditioned. The message reports: <ul style="list-style-type: none"> • Whether predictions from mixed-effects or GAM models are conditional on random effects (<code>re_form = NULL</code>) or represent marginal / population-level predictions (<code>re_form = NA</code> or <code>~0</code>).

- For numeric by variables, the values (e.g., quantiles or user-specified `by_breaks`) at which predictions are evaluated.
- Which variables are held fixed during prediction (and their values).
- Which variables vary across the prediction grid (typically the focal predictor and, if specified, the `by` variable).

This option does not affect the plot or returned values and is intended as a diagnostic aid to improve transparency and reproducibility. Default is FALSE.

plot	Logical. If TRUE (default), <code>easyViz</code> produces a plot. If FALSE, no plot is drawn and the function only returns the predicted values (as an invisible <code>easyviz.pred.df</code> object). This is useful when you want to extract or store the predictions (e.g., in a data frame) without generating any graphical output.
------	--

Details

This function provides an easy-to-use yet highly flexible tool for visualizing conditional effects from a wide range of regression models, including mixed-effects and generalized additive (mixed) models. Compatible model types include `lm`, `rlm`, `glm`, `glm.nb`, `betareg`, and `mgcv:::gam`; nonlinear models via `nls`; generalized least squares via `gls`; survival models via `survival::coxph`. Mixed-effects models with random intercepts and/or slopes can be fitted using `lmer`, `glmer`, `glmer.nb`, `glmmTMB`, or `mgcv:::gam` (via smooth terms). The function handles nonlinear relationships (e.g., splines, polynomials), two-way interactions, and supports visualization of three-way interactions via conditional plots. Plots are rendered using base R graphics with extensive customization options available through the `plot_args` and `legend_args` argument. Users can pass any valid graphical parameters accepted by `plot`, `par` or `legend` enabling full control over axis/legend labels, font styles, colors, margins, and more.

Tip: To customize plot appearance, look for argument names by prefix. Arguments starting with `point_` control the appearance of raw data. Arguments starting with `pred_` control the appearance of predicted values (lines or points). Arguments starting with `ci_` adjust the display of confidence intervals (polygons, lines or bars). Arguments starting with `legend_` control the appearance of the legend. This naming convention simplifies styling: just type the prefix (`point`, `pred`, `ci`, or `legend`) to discover relevant arguments.

The arguments `model`, `data`, and `predictor` are required. The function will return an error if any of them is missing or invalid.

Value

A base R plot visualizing the conditional effect of a predictor on the response variable. Additionally, a data frame is invisibly returned containing the predictor values, conditioning variables, predicted values (`fit`), and lower and upper confidence limits. The confidence interval columns are labeled according to the specified level (e.g., `95lcl` and `95ucl` for `ci_level = 0.95`). To extract prediction data for further use (e.g., custom plotting or tabulation), assign the output to an object: `pred.df <- easyViz(...)`. You can then inspect it using `head(pred.df)` or save it with `write.csv(pred.df, ...)`.

Examples

```
#-----
# Load required packages
```

```

#-----
library(MASS)
library(nlme)
library(betareg)
library(survival)
library(lme4)
library(glmmTMB)
library(mgcv)

#-----
# Simulate dataset
#-----
set.seed(123)
n <- 100
x1 <- rnorm(n)
x2 <- rnorm(n)
x3 <- runif(n, 0, 5)
x4 <- factor(sample(letters[1:3], n, replace = TRUE))
group_levels <- paste0("G", 1:10)
group <- factor(sample(group_levels, n, replace = TRUE))

# Generate random intercepts for each group
group_effects <- rnorm(length(group_levels), mean = 0, sd = 2) # non-zero variance
names(group_effects) <- group_levels
group_intercept <- group_effects[as.character(group)]

# Non-linear continuous response
true_y <- 5 * sin(x3) + 3 * x1 + group_intercept + model.matrix(~x4)[, -1] %*% c(2, -2)
noise <- rnorm(n, sd = 3)
y <- as.vector(true_y + noise)

# Binary response with group effect added to logit
logit_p <- 2 * x1 - 1 + group_intercept
p <- 1 / (1 + exp(-logit_p))
binary_y <- rbinom(n, size = 1, prob = p)

# Binomial response: number of successes and failures
y3 <- sample(10:30, n, replace = TRUE)
logit_p_prop <- -1.5 * scale(x1)
p_prop <- 1 / (1 + exp(-logit_p_prop))
y1 <- rbinom(n, size = y3, prob = p_prop) # successes
y2 <- y3 - y1 # failures

# Count response with group effect in log(mu)
mu_count <- exp(1 + 0.8 * x2 - 0.5 * (x4 == "b") + group_intercept)
size <- 1.2
count_y <- rnbinom(n, size = size, mu = mu_count)
# Offset variable
exposure <- runif(n, 1, 10)

# Assemble dataset
sim.data <- data.frame(x1, x2, x3, x4, group, y, binary_y, y1, y2, y3, count_y, exposure)

```

```

#-----
# 1. Linear model (lm)
#-----
mod.lm <- lm(y ~ x1 + x4,
              data = sim.data)
easyViz(model = mod.lm, data = sim.data, predictor = "x1",
        by = "x4",
        pred_range_limit = FALSE,
        pred_on_top = TRUE,
        ylim = c(-12,18),
        xlab = "Predictor x1",
        ylab = "Response y",
        point_col = ifelse(sim.data$x4=="a", "red",
                           ifelse(sim.data$x4=="b", "orange",
                                 "yellow")),
        point_cex = 0.5,
        pred_line_col = c("red", "orange", "yellow"),
        pred_line_lty = 1,
        ci_polygon_col = c(rgb(1,0,0,0.5),
                           rgb(1,0.5,0,0.5),
                           rgb(1,1,0,0.5)))

mod.lm2 <- lm(sqrt(x3) ~ x1 * x4,
              data = sim.data)
easyViz(model = mod.lm2, data = sim.data, predictor = "x1",
        by="x4",
        backtransform_response = function(x) x^2,
        ylim = c(0,8),
        show_data_points = FALSE,
        show_conditioning = TRUE)

mod.lm3 <- lm(y ~ poly(x3, 3),
              data = sim.data)
easyViz(model = mod.lm3, data = sim.data, predictor = "x3",
        pred_on_top = TRUE,
        font_family = "mono",
        point_col = rgb(1,0,0,0.3),
        point_pch = "+",
        ci_level = 0.85,
        ci_type = "lines",
        ci_line_lty = 2)

# Extract prediction data
pred.df <- easyViz(model = mod.lm, data = sim.data, predictor = "x1",
                     by = "x4", ci_level = 0.85, plot = FALSE)
head(pred.df)

#-----
# 2. Robust linear model (rlm)
#-----
mod.rlm <- rlm(y ~ x1 + x4,
                 data = sim.data)

```

```

old_xpd_mar <- par(xpd = TRUE, mar = c(5.1, 4.1, 4.1, 5.1))
easyViz(model = mod.rlm, data = sim.data, predictor = "x1",
        by = "x4",
        pred_on_top = TRUE,
        bty = "n",
        xlab = "Predictor x1",
        ylab = "Response y",
        point_col = ifelse(sim.data$x4=="a", "red",
                           ifelse(sim.data$x4=="b", "orange",
                                 "yellow")),
        point_cex = 0.5,
        pred_line_col = c("red", "orange", "yellow"),
        pred_line_lty = 1,
        ci_polygon_col = c(rgb(1,0,0),
                           rgb(1,0.5,0),
                           rgb(1,1,0)),
        ci_polygon_alpha = 0.4,
        legend_position = c(2.25,13),
        legend_title = "Predictor x4",
        legend_title_size = 0.9,
        legend_args = list(legend = c("a", "b", "c",
                                      "a", "b", "c"),
                           col = c("red", "orange", "yellow",
                                   "red", "orange", "yellow"),
                           lty = c(1, 1, 1, NA, NA, NA),
                           lwd = c(2, 2, 2, NA, NA, NA),
                           pch = c(NA, NA, NA, 16, 16, 15),
                           cex = 0.75,
                           bty = "n"))
par(old_xpd_mar)

#-----
# 3. Generalized least squares (gls)
#-----
mod.gls <- gls(y ~ x1 + x2 + x4,
                 correlation = corAR1(form = ~1|group),
                 data = sim.data)
easyViz(model = mod.gls, data = sim.data, predictor = "x4",
        jitter_data_points = TRUE,
        bty = "n",
        xlab = "Predictor x4",
        ylab = "Response y",
        point_col = rgb(0,0,1,0.2),
        pred_point_col = "blue",
        cat_labels = c("group A", "group B", "group C"))

sim.data$x5 <- sample(c(rep("CatA", 50), rep("CatB", 50)))
mod.gls2 <- gls(y ~ x1 + x2 + x4 * x5,
                  correlation = corAR1(form = ~1|group),
                  data = sim.data)
old_xpd_mar <- par(xpd = TRUE, mar = c(5.1, 4.1, 4.1, 5.1))
easyViz(model = mod.gls2, data = sim.data, predictor = "x4",
        by = "x5",

```

```

jitter_data_points = TRUE,
bty = "n",
ylim = c(-15,15),
xlab = "Predictor x4",
ylab = "Response y",
cat_labels = c("group A", "group B", "group C"),
point_col = c(rgb(0,0,1,0.2), rgb(1,0,0,0.2)),
pred_point_col = c("blue", "red"),
ci_bar_col = c("blue", "blue", "blue", "red", "red", "red"),
ci_bar_caps = 0,
legend_position = "topright",
legend_args = list(title = "Predictor x5",
                    title.cex = 1,
                    legend = c("A", "B"),
                    pt.cex = 1.25,
                    horiz = TRUE,
                    inset = c(-0.2, 0)))
par(old_xpd_mar)

#-----
# 4. Nonlinear least squares (nls)
#-----
mod.nls <- nls(y ~ a * sin(b * x3) + c,
                 data = sim.data,
                 start = list(a = 5, b = 1, c = 0))
summary(mod.nls)
easyViz(model = mod.nls, data = sim.data, predictor = "x3",
        pred_on_top = TRUE,
        font_family = "serif",
        bty = "n",
        xlab = "Predictor x3",
        ylab = "Response y",
        point_col = rgb(0,1,0,0.7),
        point_pch = 1,
        ci_type = "lines",
        ci_line_col = "black",
        ci_line_lty = 2)
text(x = 2.5, y = 11,
      labels = expression(Y %~% 5.31584 %*% sin(1.08158 %*% X[3]) + 0.51338),
      cex = 0.7)

#-----
# 5. Generalized linear model (glm)
#-----
mod.glm <- glm(binary_y ~ x1 + x4 + offset(log(exposure)),
                 family = binomial(link="cloglog"),
                 data = sim.data)
easyViz(model = mod.glm, data = sim.data, predictor = "x1",
        fix_values = list(x4="b", exposure=1),
        xlab = "Predictor x1",
        ylab = "Response y",
        binary_data_type = "binned",
        point_col = "black",

```

```
ci_polygon_col = "red",
ci_polygon_alpha = 1)

easyViz(model = mod.glm, data = sim.data, predictor = "x4",
        bty = "n",
        xlab = "Predictor x4",
        ylab = "Response y",
        binary_data_type = "plain",
        jitter_data_points = TRUE,
        point_col = "black",
        point_pch = "|",
        point_cex = 0.5)

mod.glm2 <- glm(y1/y3 ~ x1 + x4, weights = y3,
                  family = binomial(link="logit"),
                  data = sim.data)
easyViz(model = mod.glm2, data = sim.data, predictor = "x1",
        pred_on_top = TRUE,
        xlab = "Predictor x1",
        ylab = "Response y",
        point_col = "black",
        ci_polygon_col = "red")

#-----
# 6. Negative binomial GLM (glm.nb)
#-----
mod.glm.nb <- glm.nb(count_y ~ x2 + offset(log(exposure)),
                      data = sim.data)
easyViz(model = mod.glm.nb, data = sim.data, predictor = "x2",
        font_family = "mono",
        bty = "L",
        plot_args = list(main = "NB model"),
        xlab = "Predictor x2",
        ylab = "Response y",
        ci_polygon_col = "blue")

#-----
# 7. Beta regression (betareg)
#-----
sim.data$prop <- y1/y3
mod.betareg <- betareg(prop ~ x1 * x4, offset = log(exposure),
                        data = sim.data, link= "cloglog")
easyViz(model = mod.betareg, data = sim.data, predictor = "x1",
        fix_values = c(exposure = 6),
        xlab = "Predictor x1",
        ylab = "Response y",
        ci_polygon_col = "forestgreen",
        show_conditioning = TRUE)

#-----
# 8. Survival model (coxph)
#-----
mod.surviv <- coxph(Surv(y3, binary_y) ~ poly(x1,2) + x4, data=sim.data)
```

```

easyViz(model = mod.surviv, data = sim.data, predictor = "x1",
        pred_type = "link",
        xlab = "Predictor x1",
        ci_polygon_col = "orange2",
        ci_polygon_alpha = 1,
        show_conditioning = TRUE)

#-----
# 9. Linear mixed model (lmer)
#-----
mod.lmer <- lmer(y ~ x1 + x4 + (1 | group),
                  data = sim.data)
easyViz(model = mod.lmer, data = sim.data, predictor = "x1",
        by="group",
        re_form = NULL,
        bty = "n",
        plot_args = list(xaxp = c(round(min(sim.data$x1),1),
                                   round(max(sim.data$x1),1), 5)),
        ylim = c(-15, 15),
        xlab = "Predictor x1",
        ylab = "Response y",
        pred_line_col = "green",
        pred_line_lty = 1,
        pred_line_lwd = 1,
        add_legend = FALSE)
old_new <- par(new = TRUE)
easyViz(model = mod.lmer, data = sim.data, predictor = "x1",
        re_form = NA,
        bty = "n",
        plot_args = list(xaxp = c(round(min(sim.data$x1),1),
                                   round(max(sim.data$x1),1), 5)),
        show_data_points = FALSE,
        xlab = "Predictor x1",
        ylab = "Response y",
        ylim = c(-15, 15),
        pred_line_col = "red",
        pred_line_lty = 1,
        pred_line_lwd = 2,
        ci_type = NULL)
par(old_new)

#-----
# 10. Generalized linear mixed model (glmer)
#-----
mod.glmer <- glmer(binary_y ~ x1 + x4 + (1 | group),
                     family = binomial,
                     data = sim.data)
easyViz(model = mod.glmer, data = sim.data, predictor = "x1",
        by = "group",
        re_form = NULL,
        cat_conditioning = "reference",
        font_family = "serif",
        xlab = "Predictor x1",

```

```

ylab = "Response y",
binary_data_type = "binned",
pred_range_limit = FALSE,
pred_line_col = "blue",
pred_line_lty = 1,
pred_line_lwd = 1,
add_legend = FALSE)

#-----
# 11. GLMM with negative binomial (glmer.nb)
#-----
mod.glmer.nb <- glmer.nb(count_y ~ x2 + x4 + (1 | group),
                           data = sim.data)
easyViz(model = mod.glmer.nb, data = sim.data, predictor = "x2",
        re_form = NA,
        bty = "n",
        xlab = "Predictor x2",
        ylab = "Response y",
        ylim = c(0, 120),
        point_pch = 1)

#-----
# 12. GLMM (glmmTMB)
#-----
mod.glmmTMB <- glmmTMB(count_y ~ x2 + x4 + (1 | group),
                          ziformula = ~ x2,
                          family = nbinom2,
                          data = sim.data)
easyViz(model = mod.glmmTMB, data = sim.data, predictor = "x2",
        re_form = NA,
        bty = "n",
        xlab = "Predictor x2",
        ylab = "Response y",
        ylim = c(0, 120),
        point_pch = 1,
        ci_type = NULL)

#-----
# 13. GAM (mgcv:::gam) with random smooth
#-----
mod.gam <- gam(y ~ s(x3) + s(group, bs = "re"),
                 data = sim.data)
easyViz(model = mod.gam, data = sim.data, predictor = "x3",
        re_form = NA,
        las = 0,
        plot_args = list(xlab = "", ylab = "", axes = FALSE),
        bty = "n",
        xlab = "Predictor x3",
        ylab = "Response y",
        point_col = "black",
        point_pch = 1,
        ci_level = 0.99,
        ci_polygon_alpha = 0.25,

```

```

    ci_polygon_col = "red")
old_new <- par(new = TRUE)
easyViz(model = mod.gam, data = sim.data, predictor = "x3",
        re_form = NA,
        las = 0,
        plot_args = list(xlab = "", ylab = "", axes = FALSE),
        bty = "n",
        xlab = "Predictor x3",
        ylab = "Response y",
        point_col = "black",
        point_pch = 1,
        ci_level = 0.95,
        ci_polygon_alpha = 0.5,
        ci_polygon_col = "red")
par(old_new)
old_new <- par(new = TRUE)
easyViz(model = mod.gam, data = sim.data, predictor = "x3",
        re_form = NA,
        las = 0,
        bty = "n",
        xlab = "Predictor x3",
        ylab = "Response y",
        point_col = "black",
        point_pch = 1,
        ci_polygon_alpha = 1,
        ci_level = 0.8,
        ci_polygon_col = "red")
par(old_new)
rect(3.5,9,4,9.5, col=adjustcolor("red", alpha.f = 0.25), border=FALSE)
rect(3.5,7.5,4,8, col=adjustcolor("red", alpha.f = 0.5), border=FALSE)
rect(3.5,6,4,6.5, col=adjustcolor("red", alpha.f = 1), border=FALSE)
text(c(4.4, 4.4, 4.4), c(9.25, 7.75, 6.25), c("99% CI", "95% CI", "80% CI"), cex=0.75)

#-----
# 14. Plotting 3-way interaction
#-----
mod.lm.int <- lm(y ~ x1 * x2 * x3,
                  data = sim.data)

# Check conditional values to use for plotting
quantile(x2, c(0.1,0.5, 0.9))
quantile(x3, c(0.1,0.5, 0.9))

# (optional) Generate a customizable function to add a strip label at the top
add_strip_label <- function(label, bg = "grey90", cex = 1, font = 2, height_mult = 2.5) {
  usr <- par("usr")
  x_left <- usr[1]
  x_right <- usr[2]
  y_top <- usr[4]
  # Estimate strip height using text height
  h <- strheight(label, cex = cex) * height_mult
  # Strip coordinates (extending above the plotting region)
  y_bottom <- y_top + 0.2 * h
}

```

```

y_top_box <- y_bottom + h
# Draw the full-width strip
rect(x_left, y_bottom, x_right, y_top_box, col = bg, border = "black", xpd = NA)
# Add centered text
text(x = mean(c(x_left, x_right)),
      y = mean(c(y_bottom, y_top_box)),
      labels = label, cex = cex, font = font, xpd = NA)
}

# par settings for multi-panel plot
old_mfrow <- par(mfrow = c(1, 3))
old_oma <- par(oma = c(4, 4, 2, 1))
old_mar <- par(mar = c(0, 0, 2, 0))

# Panel 1
easyViz(model = mod.lm.int, data = sim.data, predictor = "x1",
        by = "x2",
        fix_values = c(x3 = 0.5750978),
        plot_args = list(xlab = "", ylab = ""),
        show_data_points = FALSE,
        pred_line_col = c(2, 3, 4),
        ci_polygon_col = c(2, 3, 4),
        legend_position = "topleft",
        legend_title = NULL,
        legend_labels = c("x2 = -1.3", "x2 = -0.2", "x2 = 1.5"))
add_strip_label("x3 = 0.6")
mtext("Response y", side = 2, outer = TRUE, line = 2.5)

# Panel 2
easyViz(model = mod.lm.int, data = sim.data, predictor = "x1",
        by = "x2",
        fix_values = c(x3 = 2.3095046),
        plot_args = list(yaxt = "n", xlab = "", ylab = ""),
        show_data_points = FALSE,
        pred_line_col = c(2, 3, 4),
        ci_polygon_col = c(2, 3, 4),
        add_legend = FALSE)
add_strip_label("x3 = 2.3")

# Panel 3
easyViz(model = mod.lm.int, data = sim.data, predictor = "x1",
        by = "x2",
        fix_values = c(x3 = 4.4509078),
        plot_args = list(yaxt = "n", xlab = "", ylab = ""),
        show_data_points = FALSE,
        pred_line_col = c(2, 3, 4),
        ci_polygon_col = c(2, 3, 4),
        add_legend = FALSE)
add_strip_label("x3 = 4.5")
mtext("Predictor x1", side = 1, outer = TRUE, line = 2.5)

# Restore original settings
par(old_mfrow)

```

```
par(old oma)
par(old mar)

#-----END OF EXAMPLES-----
```

Index

easyViz, [2](#)