Package ‘easyViz’

July 22, 2025

Title Easy Visualization of Conditional Effects from Regression Models

Version 1.0.0

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(), and gam() (from 'mgcv'); nonlinear models via nls(); and generalized least squares via gls(). 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. Robust standard errors for rlm() are computed using the sandwich estimator (Zeileis 2004) <doi:10.18637/jss.v011.i10>. For mixed models using 'glmmTMB', see Brooks et al. (2017) <doi:10.32614/RJ-2017-066>. For linear mixed-effects models with 'lme4', see Bates et al. (2015) <doi:10.18637/jss.v067.i01>. Methods for generalized additive models follow Wood (2017) <doi:10.1201/9781315370279>.

Maintainer Luca Corlatti <lucac1980@yahoo.it>

Imports stats, utils, graphics, grDevices

Suggests nlme, lme4, MASS, glmmTMB, mgcv, numDeriv, sandwich

Encoding UTF-8

RoxygenNote 7.3.2

NeedsCompilation no

Author Luca Corlatti [aut, cre]

Repository CRAN

Date/Publication 2025-07-22 10:10:06 UTC

Contents

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,
  pred_type = "response",
  pred_range_limit = TRUE,
  pred_on_top = FALSE,
  num_conditioning = "median",
  cat_conditioning = "mode",
  fix_values = NULL,
  backtransform_response = NULL,
  re.form = NULL,
  xlim = NULL,
  ylim = NULL,
  xlab = NULL,
  ylab = NULL,
  font_family = "",
  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),
  pred_line_lwd = 2,
  ci_type = "polygon",
  ci_polygon_col = c("gray", "black", "lightgray", "darkgray"),
  ci_line_col = "black",
  ci_line_lty = c(1, 2, 3, 4),
  ci_line_lwd = 1,
  pred_point_col = c("black", "gray", "darkgray", "lightgray"),
```

```
  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,
  cat_labels = NULL,
  add_legend = FALSE,
  legend_position = "top",
  legend_text_size = 0.9,
  legend_labels = NULL
)
```

Arguments

`model` [required] A fitted model object (e.g., `model = your.model`). Supported models include a wide range of regression types, including linear, robust linear, non-linear, generalized least squares, generalized linear, mixed-effects, and generalized additive (mixed) models. Compatible model-fitting functions include: `stats::lm`, `MASS::rlm`, `stats::nls`, `nlme::gls`, `stats::glm`, `MASS::glm.nb`, `lme4::lmer`, `lme4::glmer`, `lme4::glmer.nb`, `glmmTMB::glmmTMB`, and `mgcv::gam`.

`data` [required] The data frame used to fit the model (e.g., `data = your.data`). This data frame is used internally for generating predictions. **All variables used in the model formula (including predictors, offsets, grouping variables, and interaction terms) must be present in this data frame**. If the model was fitted without using a `data` argument (e.g., using variables from the global environment), you must ensure that `data` 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., `predictor = "x1"`).

`by` The name of an interaction or additional variable for conditioning (e.g., `by = "x2"`). If a continuous variable is used, cross-sections are taken at the 10th, 50th, and 90th quantiles. If a categorical variable is used, a separate line or point will be plotted for each level. This can also be used to visualize group-level random effects all at once: namely, when `by` corresponds to a grouping variable used in a random effect term (e.g., if `by = "group"` when the random term is specified as `(1|group)` or `s(group, bs="re")`) and `re.form = NULL`, predictions are conditional on each group's estimated random effect. 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.

`pred_type` 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).

pred_range_limit
: 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, pred_range_limit is automatically set to FALSE, since numeric by values are treated as continuous rather than grouping factors.

pred_on_top
: 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.

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 — a common approach when visualizing effects in multivariable models. 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 (factors), provide the desired level as a character string or factor (e.g., fix_values = c(group = "levelA") or fix_values = list(group = levels(data$group)[1])). This overrides the default conditioning behavior specified via num_conditioning and cat_conditioning. **Note:** 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 useful for setting offsets, forcing predictions at specific values, or ensuring consistent conditioning across models. For example, it is particularly useful 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 description for details and an example of how to apply this approach.

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:** 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.

re.form
: A formula specifying which random effects to include when generating predictions. This argument is relevant for mixed-effects models only (e.g., from lme4, glmmTMB, or mgcv::gam). Use re.form = NULL (default) to include 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 that, when re.form = NULL, the prediction reflects the conditional estimate for that group level. However, you can explicitly fix the level of the grouping variable using the fix_values argument — this allows you to visualize group-specific predictions for a specific level of the random term (e.g., fix_values = c(group = "levelA")). If all levels are equally frequent and no value is specified via fix_values, the first level (in factor order) is used, which typically follows alphabetical order unless manually re-leveled. Use re.form = NA or re.form = ~0 to obtain population-level predictions based only on fixed effects — this means that random effects are part of the model fit but are excluded from the prediction, resulting in population-level (i.e., marginal) predictions based solely on fixed effects. This is equivalent to assuming the random effects are zero — i.e., an 'average' group or subject. For mgcv::gam() models, random effects are typically modeled using smooth terms such as 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 in the prediction, while re.form = NA or re.form = ~0 excludes them by internally using the exclude argument in predict.gam(). For all types of mixed models, when re.form = NULL and by corresponds to a grouping variable used in a random effect term, group-specific (i.e., conditional) predictions are visualized for all levels of the grouping variable. **Note:** For models fitted with lme4 (e.g., lmer(), glmer()), standard errors are not available when re.form = NULL.

xlim
: x-axis limits for the plot (e.g., xlim = c(0, 10)). Defaults to automatic scaling based on the data range.

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".

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, margin settings, font family, and tick mark style. 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
- Margins and layout: mar, oma, mgp, tcl, las, adj
- Box and axis rendering: bty, axes, frame.plot, ann
- Coordinate settings: xlim, ylim, xaxs, yaxs, asp, xlog, ylog

This is a flexible alternative to manually setting individual plot parameters in the function signature. For a full list of supported parameters, see ?par and ?plot.default. Example usage:
plot_args = list(main = "Title", cex.lab = 1.2, col.axis = "gray40", mar = c(5, 4, 4, 2), las = 1).

show_data_points — Logical. Whether to display raw data points (default: TRUE). For binomial models where the response is expressed in the formula as cbind(successes, failures) or as successes / trials, the raw data points plotted on the y-axis are based on the calculated proportions: successes / (successes + failures) or successes / trials, respectively. These proportions are computed internally from the original data and temporarily added to the data set for visualization purposes.

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 (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"). Dynamic: accepts multiple values when points are plotted for different values/levels of a variable. **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 setting a low alpha value (e.g., `rgb(1,0,0, alpha = 0.1)`, or by using `adjustcolor()` with the argument `alpha.f` (e.g., `adjustcolor("red", alpha.f = 0.1)`). In such cases, consider setting `pred_on_top = TRUE` to ensure that prediction lines and confidence intervals remain clearly visible above the dense cloud of raw points.

`point_pch` Point shape for raw data (default: `16`). Dynamic: accepts multiple values when points are plotted for different values/levels of a variable.

`point_cex` Point size for raw data (default: `0.75`). Dynamic: accepts multiple values when points are plotted for different values/levels of a variable.

`pred_line_col` Color of the predicted line for numerical predictors (default: `"black"`). Can be specified as a color name, number or RGB/hex string. Dynamic: accepts multiple values (e.g., `c("red", "green", "blue")`) when multiple lines are plotted (i.e., when `by` is specified).

`pred_line_lty` Type of the predicted line for numerical predictors (default: `1`). Dynamic: accepts multiple values (e.g., `c(1, 2, 3)`) when multiple lines are plotted (i.e., when `by` is specified).

`pred_line_lwd` Width of the predicted line for numerical predictors (default: `2`). Dynamic: accepts multiple values (e.g., `c(1, 2, 3)`) when multiple lines are plotted (i.e., when `by` is specified).

`ci_type` Type of 95 percent confidence intervals for numerical predictors. `"polygon"` (default) or `"lines"`.

`ci_polygon_col` Color for 95 percent confidence interval polygon (default: `"gray"`). Requires `ci_type = "polygon"`. Can be specified as a color name, number or RGB/hex string. Dynamic: accepts multiple values (e.g., `c("red", "green", "blue")`) when 95 percent CIs are plotted for multiple lines (i.e., when `by` is specified). **Tip:** To hide confidence bands entirely, set this to `rgb(0,0,0,0)`.

`ci_line_col` Color for 95 percent confidence interval lines (default: `"black"`). Requires `ci_type = "lines"`. Can be specified as a color name, number or RGB/hex string. Dynamic: accepts multiple values (e.g., `c("red", "green", "blue")`) when 95 percent CIs are plotted for multiple lines (i.e., when `by` is specified).

`ci_line_lty` Type for 95 percent confidence interval lines (default: `1`). Requires `ci_type = "lines"`. Dynamic: accepts multiple values (e.g., `c(1, 2, 3)`) when 95 percent CIs are plotted for multiple lines (i.e., when `by` is specified).

`ci_line_lwd` Width for 95 percent confidence interval lines (default: `1`). Requires `ci_type = "lines"`. Dynamic: accepts multiple values (e.g., `c(1, 2, 3)`) when 95 percent CIs are plotted for multiple lines (i.e., when `by` is specified).

`pred_point_col` Color for predicted point values of categorical predictors (default: `"black"`). Can be specified as a color name, number or RGB/hex string. Dynamic: accepts multiple values (e.g., `c("red", "green", "blue")`) when points are plotted for an interaction (i.e., when `by` is specified).

`pred_point_pch` Shape for predicted point values of categorical predictors (default: `16`). Dynamic: accepts multiple values (e.g., `c(1, 2, 3)`) when points are plotted for an interaction (i.e., when `by` is specified).

pred_point_cex
: Size for predicted point values of categorical predictors (default: 1). Dynamic: accepts multiple values (e.g., c(1, 2, 3)) when points are plotted for an interaction (i.e., when by is specified).

ci_bar_col
: Color for 95 percent confidence interval bars (default: "black"). Applies only when the predictor is categorical. Can be specified as a color name, number, or RGB/hex string.

ci_bar_lty
: Type for 95 percent confidence interval bars (default: 1). Applies only when the predictor is categorical.

ci_bar_lwd
: Width for 95 percent confidence interval bars (default: 1). Applies only when the predictor is categorical.

ci_bar_caps
: Size of the caps on 95 percent confidence interval bars (default: 0.1). Increase for more visible caps, set to 0 to remove caps and draw plain vertical bars.

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

add_legend
: Logical. Whether to add a legend for by variable levels (default: FALSE).

legend_position
: Legend position. Either a named position string ("top", "bottom", "left", "right", "topleft", "topright", "bottomleft", "bottomright") or a numeric vector c(x, y) specifying exact coordinates for the legend placement.

legend_text_size
: Legend text size (default: 0.9).

legend_labels
: Custom labels for the legend (e.g., legend_labels = c("Label 1", "Label 2", "Label 3")).

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, and mgcv::gam; nonlinear models via nls; and generalized least squares via gls. 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 argument. Users can pass any valid graphical parameters accepted by plot or par, enabling full control over axis labels, font styles, colors, margins, tick orientation, and more. 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 their 95 percent confidence intervals (lower, upper). 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(nlme)
library(MASS)
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
offset_var <- log(runif(n, 1, 10))
```

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

#------------------------------------------
# 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,
        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(1,0,0,0.5),
                           rgb(1,0.5,0,0.5),
                           rgb(1,1,0,0.5)),
        add_legend = TRUE,
        legend_position = "topleft",
        legend_labels = c("a", "b", "c"))

# Extract prediction data
pred_df <- easyViz(model = mod_lm, data = sim_data, predictor = "x1", by = "x4")
head(pred_df)

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,
        add_legend = 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_type = "lines",
        ci_line_lty = 2)

#------------------------------------------
# 2. Robust linear model (rlm)
```

```
#------------------------------------------
mod_rlm <- rlm(y ~ x1 + x4,
               data = sim_data)
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(1,0,0,0.5),
                           rgb(1,0.5,0,0.5),
                           rgb(1,1,0,0.5)),
        add_legend = TRUE,
        legend_position = c(1.25,-1),
        legend_labels = c("a", "b", "c"))

#------------------------------------------
# 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)
easyViz(model = mod_gls2, data = sim_data, predictor = "x4",
        by = "x5",
        jitter_data_points = TRUE,
        bty = "n",
        ylim = c(-15,20),
        xlab = "Predictor x4",
        ylab = "Response y",
        point_col = c(rgb(0,0,1,0.2), rgb(1,0,0,0.2)),
        pred_point_col = c("blue", "red"),
        ci_bar_caps = 0,
        cat_labels = c("group A", "group B", "group C"),
        add_legend = TRUE,
```

```
        legend_position = "topright",
        legend_labels = c("Category A", "Category B"))

#-----------------------------------------
# 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(offset_var)),
               family = binomial(link="cloglog"),
               data = sim_data)
easyViz(model = mod_glm, data = sim_data, predictor = "x1",
        fix_values = list(x4="b", offset_var=1),
        xlab = "Predictor x1",
        ylab = "Response y",
        binary_data_type = "binned",
        point_col = "black",
        ci_polygon_col = "red")

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,
                     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. Linear mixed-effects 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)
oldpar <- 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_polygon = rgb(0,0,0,0))
par(oldpar)

#------------------------------------------
```

```
# 8. 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)

#------------------------------------------
# 9. 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)

#------------------------------------------
# 10. GLMM using 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)

#------------------------------------------
# 11. GAM with random smooth for group
#------------------------------------------
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,
bty = "n",
xlab = "Predictor x3",
ylab = "Response y",
point_col = "black",
point_pch = 1,
ci_polygon_col = rgb(1,0,0,0.5))
```

Index