# Package ‘ggeffects’

**Type** Package  
**Encoding** UTF-8  
**Title** Create Tidy Data Frames of Marginal Effects for 'ggplot' from Model Outputs  
**Version** 0.15.1  
**Maintainer** Daniel Lüdecke <d.luedecke@uke.de>  
**Description** Compute marginal effects from statistical models and returns the result as tidy data frames. These data frames are ready to use with the 'ggplot2'-package. Marginal effects can be calculated for many different models. Interaction terms, splines and polynomial terms are also supported. The main functions are ggpredict(), ggemmeans() and ggeffect(). There is a generic plot()-method to plot the results using 'ggplot2'.  
**License** GPL-3  
**Depends** R (>= 3.2)  
**Imports** graphics, insight (>= 0.8.1), MASS, sjlabelled (>= 1.1.2), stats  
**Suggests** AER, aod, betareg, brms, clubSandwich, effects (>= 4.1-2), emmeans (>= 1.4.1), gam, gamm4, gee, geepack, ggplot2, GLMMadaptive, glmmTMB (>= 1.0.0), htr, knitr, lme4, logistf, magrittr, Matrix, MCMCglmm, mgcv, nlme, ordinal, prediction, pscl, quantreg, rmarkdown, rms, robustbase, rstanarm, rstantools, sandwich, see, sjstats, sjmisc (>= 2.8.2), survey, survival, testthat, VGAM  
**URL** https://strengejacke.github.io/ggeffects  
**BugReports** https://github.com/strengejacke/ggeffects/issues  
**RoxygenNote** 7.1.1  
**VignetteBuilder** knitr  
**NeedsCompilation** no  
**Author** Daniel Lüdecke [aut, cre] (<https://orcid.org/0000-0002-8895-3206>), Frederik Aust [ctb] (<https://orcid.org/0000-0003-4900-788X>), Sam Crawley [ctb] (<https://orcid.org/0000-0002-7847-0411>)
R topics documented:

efc .......................................................... 2
fish ......................................................... 2
get_title ............................................... 3
ggeffect .............................................. 4
new_data ........................................... 12
plot .................................................... 13
pretty_range ..................................... 16
values_at ......................................... 17
vcov ................................................... 18

Description

A SPSS sample data set, imported with the sjlabelled::read_spss() function.

Examples

# Attach EFC-data
data(efc)

# Show structure
str(efc)

# show first rows
head(efc)

fish  Sample data set

Description

A sample data set, used in tests and some examples.
get_title  Get titles and labels from data

Description
Get variable and value labels from ggeffects-objects. Functions like ggpredict() or ggeffect() save information on variable names and value labels as additional attributes in the returned data frame. This is especially helpful for labelled data (see sjlabelled), since these labels can be used to set axis labels and titles.

Usage
get_title(x, case = NULL)
g_x_title(x, case = NULL)
g_y_title(x, case = NULL)
g_legend_title(x, case = NULL)
g_legend_labels(x, case = NULL)
g_x_labels(x, case = NULL)
g_complete_df(x, case = NULL)

Arguments
x An object of class ggeffects, as returned by any ggeffects-function; for get_complete_df(), must be a list of ggeffects-objects.
case Desired target case. Labels will automatically converted into the specified character case. See ?sjlabelled::convert_case for more details on this argument.

Value
The titles or labels as character string, or NULL, if variables had no labels; get_complete_df() returns the input list x as single data frame, where the grouping variable indicates the marginal effects for each term.

Examples
library(sjmisc)
data(efc)
efc$c172code <- to_factor(efc$c172code)
fit <- lm(barthtot ~ c12hour + neg_c_7 + c161sex + c172code, data = efc)

mydf <- ggpredict(fit, terms = c("c12hour", "c161sex", "c172code"))
library(ggplot2)
ggplot(mydf, aes(x = x, y = predicted, colour = group)) +
  stat_smooth(method = "lm") +
  facet_wrap(~facet, ncol = 2) +
  labs(
    x = get_x_title(mydf),
    y = get_y_title(mydf),
    colour = get_legend_title(mydf)
  )

# get marginal effects, a list of data frames (one data frame per term)
eff <- ggeffect(fit)
eff
gget_complete_df(eff)

# get marginal effects for education only, and get x-axis-labels
mydat <- eff["c172code"]
ggplot(mydat, aes(x = x, y = predicted, group = group)) +
  stat_summary(fun.y = sum, geom = "line") +
  scale_x_discrete(labels = get_x_labels(mydat))

ggeffect 

Marginal effects and estimated marginal means from regression models

Description

The ggeffects package computes estimated marginal means (predicted values) for the response, at the margin of specific values or levels from certain model terms, i.e. it generates predictions by a model by holding the non-focal variables constant and varying the focal variable(s).

ggpredict() uses predict() for generating predictions, while ggeffect() computes marginal effects by internally calling effects::Effect() and ggemmeans() uses emmeans::emmeans(). The result is returned as consistent data frame.

Usage

  ggeffect(model, terms, ci.lvl = 0.95, ...)

  ggemmeans(
    model,
    terms,
    ci.lvl = 0.95,
    type = "fe",
    typical = "mean",
    condition = NULL,
    back.transform = TRUE,
ggpredict(
  model,
  terms,
  ci.lvl = 0.95,
  type = "fe",
  typical = "mean",
  condition = NULL,
  back.transform = TRUE,
  ppd = FALSE,
  vcov.fun = NULL,
  vcov.type = NULL,
  vcov.args = NULL,
  interval = "confidence",
  ...
)

Arguments

model A fitted model object, or a list of model objects. Any model that supports
common methods like predict(), family() or model.frame() should work. For
\code{ggeffect()}, any model that is supported by \code{effects} should work, and for
\code{ggemmeans()}, all models supported by \code{emmeans} should work.

terms Character vector (or a formula) with the names of those terms from \code{model}, for
which marginal effects should be displayed. At least one term is required to calcu-
late effects for certain terms, maximum length is four terms, where the second
to fourth term indicate the groups, i.e. predictions of first term are grouped at the
values or levels of the remaining terms. If \code{terms} is missing or \code{NULL}, marginal
effects for each model term are calculated. It is also possible to define specific
values for terms, at which marginal effects should be calculated (see 'Details').
All remaining covariates that are not specified in \code{terms} are held constant (see
'Details'). See also arguments condition and typical.

ci.lvl Numeric, the level of the confidence intervals. For \code{ggpredict()}, use \code{ci.lvl}
= \code{NA}, if confidence intervals should not be calculated (for instance, due to com-
putation time). Typically, confidence intervals based on the standard errors as
returned by the \code{predict()} function are returned, assuming normal distribution
(i.e. +/- 1.96 * SE). See introduction of \code{this vignette} for more details.

... For \code{ggpredict()}, further arguments passed down to \code{predict()}; for \code{ggeffect()},
further arguments passed down to \code{effects::Effect()}; and for \code{ggemmeans()},
further arguments passed down to \code{emmeans::emmeans()}. If \code{type = "sim"}, ...
may also be used to set the number of simulation, e.g. \code{nsim = 500}.

type Character, only applies for survival models, mixed models and/or models
with zero-inflation. \code{Note:} For \pkg{brmsfit}-models with zero-inflation component,
there is no \code{type = "zero_inflated"} nor \code{type = "zi_random"}; predicted values
for \pkg{MixMod}-models from \pkg{GLMMadaptive} with zero-inflation component
\emph{always} condition on the zero-inflation part of the model (see 'Details').
"fixed" (or "fe" or "count") Predicted values are conditioned on the fixed effects or conditional model only (for mixed models: predicted values are on the population-level and confidence intervals are returned). For instance, for models fitted with zeroInfl from pscl, this would return the predicted mean from the count component (without zero-inflation). For models with zero-inflation component, this type calls predict(..., type = "link") (however, predicted values are back-transformed to the response scale).

"random" (or "re") This only applies to mixed models, and type = "random" does not condition on the zero-inflation component of the model. type = "random" still returns population-level predictions, however, unlike type = "fixed", intervals also consider the uncertainty in the variance parameters (the mean random effect variance, see Johnson et al. 2014 for details) and hence can be considered as prediction intervals. For models with zero-inflation component, this type calls predict(..., type = "link") (however, predicted values are back-transformed to the response scale).

To get predicted values for each level of the random effects groups, add the name of the related random effect term to the terms-argument (for more details, see this vignette).

"zero_inflated" (or "fe.zi" or "zi") Predicted values are conditioned on the fixed effects and the zero-inflation component. For instance, for models fitted with zeroInfl from pscl, this would return the predicted response (\(\mu*(1-p)\)) and for glmmTMB, this would return the expected value \(\mu*(1-p)\) without conditioning on random effects (i.e. random effect variances are not taken into account for the confidence intervals). For models with zero-inflation component, this type calls predict(..., type = "response"). See 'Details'.

"zi_random" (or "re.zi" or "zero_inflated_random") Predicted values are conditioned on the zero-inflation component and take the random effects uncertainty into account. For models fitted with glmmTMB(), hurdle() or zeroInfl(), this would return the expected value \(\mu*(1-p)\). For glmmTMB, prediction intervals also consider the uncertainty in the random effects variances. This type calls predict(..., type = "response"). See 'Details'.

"zi_prob" (or "zi.prob") Predicted zero-inflation probability. For glmmTMB models with zero-inflation component, this type calls predict(..., type = "zlink"); models from pscl call predict(..., type = "zero") and for GLMMadaptive, predict(..., type = "zero_part") is called.

"sim" Predicted values and confidence resp. prediction intervals are based on simulations, i.e. calls to simulate(). This type of prediction takes all model uncertainty into account, including random effects variances. Currently supported models are glmmTMB and merMod. See ... for details on number of simulations.

"survival" and "cumulative_hazard" (or "surv" and "cumhaz") Applies only to coxph-objects from the survival-package and calculates the survival probability or the cumulative hazard of an event.

typical Character vector, naming the function to be applied to the covariates over which the effect is "averaged". The default is "mean". See ?sjmisc::typical_value
condition
Named character vector, which indicates covariates that should be held constant at specific values. Unlike typical, which applies a function to the covariates to determine the value that is used to hold these covariates constant, condition can be used to define exact values, for instance `condition = c(covariate1 = 20, covariate2 = 5)`. See 'Examples'.

back.transform
Logical, if `TRUE` (the default), predicted values for log- or log-log transformed responses will be back-transformed to original response-scale.

ppd
Logical, if `TRUE`, predictions for Stan-models are based on the posterior predictive distribution (`rstantools::posterior_predict()`). If `FALSE` (the default), predictions are based on posterior draws of the linear predictor (`rstantools::posterior_linpred()`).

vcov.fun
String, indicating the name of the `vcov*()`-function from the `sandwich` or `clubSandwich`-package, e.g. `vcov.fun = "vcovCL"`, which is used to compute (cluster) robust standard errors for predictions. If `NULL`, standard errors (and confidence intervals) for predictions are based on the standard errors as returned by the `predict()`-function. Note that probably not all model objects that work with `ggpredict()` are also supported by the `sandwich` or `clubSandwich`-package.

vcov.type
Character vector, specifying the estimation type for the robust covariance matrix estimation (see `?sandwich::vcovHC` or `?clubSandwich::vcovCR` for details).

vcov.args
List of named vectors, used as additional arguments that are passed down to `vcov.fun`.

interval
Type of interval calculation, can either be "confidence" (default) or "prediction". May be abbreviated. Unlike `confidence intervals`, `prediction intervals` include the residual variance (sigma^2). This argument is ignored for mixed models, as interval = "prediction" is equivalent to type = "random" (and interval = "confidence" is equivalent to type = "fixed"). Note that prediction intervals are not available for all models.

Details

**Supported Models:** A list of supported models can be found at [https://github.com/strengejacke/ggeffects](https://github.com/strengejacke/ggeffects). Support for models varies by function, i.e. although `ggpredict()`, `ggemmeans()` and `ggeffect()` support most models, some models are only supported exclusively by one of the three functions.

**Difference between `ggpredict()` and `ggeffect()` or `ggemmeans()`:** `ggpredict()` calls `predict()`, while `ggeffect()` calls `effects::Effect()` and `ggemmeans()` calls `emmeans::emmeans()` to compute marginal effects. Therefore, `ggpredict()` and `ggeffect()` resp. `ggemmeans()` differ in how factors are held constant: `ggpredict()` uses the reference level, while `ggeffect()` and `ggemmeans()` compute a kind of "average" value, which represents the proportions of each factor's category. Use `condition` to set a specific level for factors in `ggemmeans()`, so factors are not averaged over their categories, but held constant at a given level.

**Marginal Effects at Specific Values:** Specific values of model terms can be specified via the `terms`-argument. Indicating levels in square brackets allows for selecting only specific groups or values resp. value ranges. Term name and the start of the levels in brackets must be separated by a whitespace character, e.g. `terms = c("age", "education [1, 3]")`. Numeric ranges, separated
with colon, are also allowed: terms = c("education", "age [30:60]"). The stepsize for range can be adjusted using 'by', e.g. terms = "age [30:60 by=5]."

The terms-argument also supports the same shortcuts as the values-argument in values_at(). So terms = "age [meansd]" would return predictions for the values one standard deviation below the mean age, the mean age and one SD above the mean age. terms = "age [quart2]" would calculate predictions at the value of the lower, median and upper quartile of age.

Furthermore, it is possible to specify a function name. Values for predictions will then be transformed, e.g. terms = "income [exp]". This is useful when model predictors were transformed for fitting the model and should be back-transformed to the original scale for predictions. It is also possible to define own functions (see this vignette).

Instead of a function, it is also possible to define the name of a variable with specific values, e.g. to define a vector v = c(1000, 2000, 3000) and then use terms = "income [v]".

You can take a random sample of any size with sample=n, e.g terms = "income [sample=8]", which will sample eight values from all possible values of the variable income. This option is especially useful for plotting marginal effects at certain levels of random effects group levels, where the group factor has many levels that can be completely plotted. For more details, see this vignette.

Finally, numeric vectors for which no specific values are given, a "pretty range" is calculated (see pretty_range), to avoid memory allocation problems for vectors with many unique values. If a numeric vector is specified as second or third term (i.e. if this vector represents a grouping structure), representative values (see values_at) are chosen (unless other values are specified). If all values for a numeric vector should be used to compute predictions, you may use e.g. terms = "age [all]". See also package vignettes.

To create a pretty range that should be smaller or larger than the default range (i.e. if no specific values would be given), use the n-tag, e.g. terms="age [n=5]" or terms="age [n=12]". Larger values for n return a larger range of predicted values.

**Holding covariates at constant values:** For ggpredict(), expand.grid() is called on all unique combinations of model.frame(model)[, terms] and used as newdata-argument for predict(). In this case, all remaining covariates that are not specified in terms are held constant: Numeric values are set to the mean (unless changed with the condition or typical-argument), factors are set to their reference level (may also be changed with condition) and character vectors to their mode (most common element).

ggeffect() and ggemmeans(), by default, set remaining numeric covariates to their mean value, while for factors, a kind of "average" value, which represents the proportions of each factor's category, is used. For ggemmeans(), use condition to set a specific level for factors so that these are not averaged over their categories, but held constant at the given level.

**Bayesian Regression Models:** ggpredict() also works with Stan-models from the rstanarm or brms-package. The predicted values are the median value of all drawn posterior samples. The confidence intervals for Stan-models are Bayesian predictive intervals. By default (i.e. ppd = FALSE), the predictions are based on rstantools::posterior_linpred() and hence have some limitations: the uncertainty of the error term is not taken into account. The recommendation is to
use the posterior predictive distribution (\texttt{rstantools::posterior_predict()}).

**Zero-Inflated and Zero-Inflated Mixed Models with \texttt{brms}:** Models of class \texttt{brmsfit} always condition on the zero-inflation component, if the model has such a component. Hence, there is no \texttt{type = "zero_inflated"} nor \texttt{type = "zi_random"} for \texttt{brmsfit}-models, because predictions are based on draws of the posterior distribution, which already account for the zero-inflation part of the model.

**Zero-Inflated and Zero-Inflated Mixed Models with \texttt{glmmTMB}:** If model is of class \texttt{glmmTMB}, \texttt{hurdle}, \texttt{zeroinfl} or \texttt{zetrunc}, simulations from a multivariate normal distribution (see ?\texttt{MASS::mvrnorm}) are drawn to calculate $\mu \times (1-p)$. Confidence intervals are then based on quantiles of these results. For \texttt{type = "zi_random"}, prediction intervals also take the uncertainty in the random-effect parameters into account (see also Brooks et al. 2017, pp.391-392 for details).

An alternative for models fitted with \texttt{glmmTMB} that take all model uncertainties into account are simulations based on \texttt{simulate()}, which is used when \texttt{type = "sim"} (see Brooks et al. 2017, pp.392-393 for details).

**MixMod-models from \texttt{GLMMadaptive}:** Predicted values for the fixed effects component (\texttt{type = "fixed"} or \texttt{type = "zero_inflated"}) are based on \texttt{predict(..., type = "mean_subject")}, while predicted values for random effects components (\texttt{type = "random"} or \texttt{type = "zi_random"}) are calculated with \texttt{predict(..., type = "subject_specific")} (see \texttt{GLMMadaptive::predict.MixMod} for details). The latter option requires the response variable to be defined in the \texttt{newdata}-argument of \texttt{predict()}, which will be set to its typical value (see \texttt{sjmisc::typical_value}).

### Value

A data frame (with \texttt{ggeffects} class attribute) with consistent data columns:

- **x** the values of the first term in \texttt{terms}, used as x-position in plots.
- **predicted** the predicted values of the response, used as y-position in plots.
- **std.error** the standard error of the predictions.
- **conf.low** the lower bound of the confidence interval for the predicted values.
- **conf.high** the upper bound of the confidence interval for the predicted values.
- **group** the grouping level from the second term in \texttt{terms}, used as grouping-aesthetics in plots.
- **facet** the grouping level from the third term in \texttt{terms}, used to indicate facets in plots.

The estimated marginal means (predicted values) are always on the response scale!

For proportional odds logistic regression (see ?\texttt{MASS::polr}) resp. cumulative link models (e.g., see \texttt{ordinal::clm}), an additional column \texttt{response.level} is returned, which indicates the grouping of predictions based on the level of the model’s response.

Note that for convenience reasons, the columns for the intervals are always named \texttt{conf.low} and \texttt{conf.high}, even though for Bayesian models credible or highest posterior density intervals are returned.
Note

**Multinomial Models:** polr-, clm-models, or more generally speaking, models with ordinal or multinominal outcomes, have an additional column `response.level`, which indicates with which level of the response variable the predicted values are associated.

**Printing Results:** The `print()`-method gives a clean output (especially for predictions by groups), and indicates at which values covariates were held constant. Furthermore, the `print()`-method has the arguments `digits` and `n` to control number of decimals and lines to be printed, and an argument `x.lab` to print factor-levels instead of numeric values if `x` is a factor.

**Limitations:** The support for some models, for example from package `MCMCglmm`, is rather experimental and may fail for certain models. If you encounter any errors, please file an issue at https://github.com/strengejacke/ggeffects/issues.

References


Examples

```r
library(sjlabelled)
data(efc)
fit <- lm(barthtot ~ c12hour + neg_c_7 + c161sex + c172code, data = efc)

ggpredict(fit, terms = "c12hour")
ggpredict(fit, terms = c("c12hour", "c172code"))
ggpredict(fit, terms = c("c12hour", "c172code", "c161sex"))

# specified as formula
ggpredict(fit, terms = ~ c12hour + c172code + c161sex)

# only range of 40 to 60 for variable 'c12hour'
ggpredict(fit, terms = "c12hour [40:60]")

# using "summary()" shows that covariate "neg_c_7" is held
# constant at a value of 11.84 (its mean value). To use a
# different value, use "condition"
ggpredict(fit, terms = "c12hour [40:60]", condition = c(neg_c_7 = 20))

# to plot ggeffects-objects, you can use the 'plot()' -function.
# the following examples show how to build your ggplot by hand.

# plot predicted values, remaining covariates held constant
library(ggplot2)
mydf <- ggpredict(fit, terms = "c12hour")
ggplot(mydf, aes(x, predicted)) +
```
```r
geom_line() +
geom_ribbon(aes(ymin = conf.low, ymax = conf.high), alpha = .1)

# three variables, so we can use facets and groups
mydf <- ggpredict(fit, terms = c("c12hour", "c161sex", "c172code"))
ggplot(mydf, aes(x = x, y = predicted, colour = group)) +
  stat_smooth(method = "lm", se = FALSE) +
  facet_wrap(~facet, ncol = 2)

# select specific levels for grouping terms
mydf <- ggpredict(fit, terms = c("c12hour", "c172code [1,3]", "c161sex"))
ggplot(mydf, aes(x = x, y = predicted, colour = group)) +
  stat_smooth(method = "lm", se = FALSE) +
  facet_wrap(~facet) +
  labs(
    y = get_y_title(mydf),
    x = get_x_title(mydf),
    colour = get_legend_title(mydf)
  )

# level indication also works for factors with non-numeric levels
# and in combination with numeric levels for other variables
data(efc)
efc$c172code <- as_label(efc$c172code)
fit <- lm(barthtot ~ c12hour + neg_c_7 + c161sex + c172code, data = efc)
ggpredict(fit, terms = c("c12hour",
  "c172code [low level of education, high level of education]",
  "c161sex [1]"))

# use categorical value on x-axis, use axis-labels, add error bars
data(efc)
dat <- ggpredict(fit, terms = c("c172code", "c161sex"))
ggplot(dat, aes(x, predicted, colour = group)) +
  geom_point(position = position_dodge(.1)) +
  geom_errorbar(
    aes(ymin = conf.low, ymax = conf.high),
    position = position_dodge(.1)
  ) +
  scale_x_discrete(breaks = 1:3, labels = get_x_labels(dat))

# 3-way-interaction with 2 continuous variables
data(efc)

# make categorical
efc$c161sex <- as_factor(efc$c161sex)
fit <- lm(neg_c_7 ~ c12hour * barthtot + c161sex, data = efc)

# select only levels 30, 50 and 70 from continuous variable Barthel-Index
data <- ggpredict(fit, terms = c("c12hour", "barthtot [30,50,70]", "c161sex"))
ggplot(data, aes(x = x, y = predicted, colour = group)) +
  stat_smooth(method = "lm", se = FALSE, fullrange = TRUE) +
  facet_wrap(~facet) +
  labs(
    colour = get_legend_title(data),
    x = get_x_title(data),
    y = get_y_title(data),
  )
```

title = get_title(dat)

# or with ggeffects' plot-method
plot(dat, ci = FALSE)

# marginal effects for polynomial terms
data(efc)
fit <- glm(
  tot_sc_e ~ c12hour + e42dep + e17age + I(e17age^2) + I(e17age^3),
  data = efc,
  family = poisson()
)
ggeffect(fit, terms = "e17age")

---

new_data Create a data frame from all combinations of predictor values

**Description**

Create a data frame for the "newdata"-argument that contains all combinations of values from the terms in questions. Similar to `expand.grid()`. The `terms`-argument accepts all shortcuts for representative values as in `ggpredict()`.

**Usage**

`new_data(model, terms, typical = "mean", condition = NULL)`

**Arguments**

- `model` A fitted model object.
- `terms` Character vector with the names of those terms from `model` for which all combinations of values should be created.
- `typical` Character vector, naming the function to be applied to the covariates over which the effect is "averaged". The default is "mean". See `sjmisc::typical_value` for options.
- `condition` Named character vector, which indicates covariates that should be held constant at specific values. Unlike `typical`, which applies a function to the covariates to determine the value that is used to hold these covariates constant, `condition` can be used to define exact values, for instance `condition = c(covariate1 = 20, covariate2 = 5)`. See 'Examples'.

**Value**

A data frame containing one row for each combination of values of the supplied variables.
Examples

data(efc)
fit <- lm(barthtot ~ c12hour + neg_c_7 + c161sex + c172code, data = efc)
new_data(fit, c("c12hour [meansd]", "c161sex"))

nd <- new_data(fit, c("c12hour [meansd]", "c161sex"))
pr <- predict(fit, type = "response", newdata = nd)
nd$predicted <- pr
don

# compare to
ggpredict(fit, c("c12hour [meansd]", "c161sex"))

---

plot

Plot ggeffects-objects

Description

A generic plot-method for ggeffects-objects.

Usage

## S3 method for class 'ggeffects'
plot(
x,
ci = TRUE,
ci.style = c("ribbon", "errorbar", "dash", "dot"),
facets,
add.data = FALSE,
colors = "Set1",
alpha = 0.15,
dodge = 0.25,
use.theme = TRUE,
dot.alpha = 0.35,
jitter = 0.2,
log.y = FALSE,
case = NULL,
show.legend = TRUE,
show.title = TRUE,
show.x.title = TRUE,
show.y.title = TRUE,
dot.size = NULL,
line.size = NULL,
connect.lines = FALSE,
grid,
one.plot = TRUE,
rawdata,
...
)

theme_ggeffects(base_size = 11, base_family = "")

show_pals()

Arguments

x
  An object of class ggeffects, as returned by the functions from this package.

ci
  Logical, if TRUE, confidence bands (for continuous variables at x-axis) resp. error bars (for factors at x-axis) are plotted.

ci.style
  Character vector, indicating the style of the confidence bands. May be either "ribbon", "errorbar", "dash" or "dot", to plot a ribbon, error bars, or dashed or dotted lines as confidence bands.

facets, grid
  Logical, defaults to TRUE, if x has a column named facet, and defaults to FALSE, if x has no such column. Set facets = TRUE to wrap the plot into facets even for grouping variables (see 'Examples'). grid is an alias for facets.

add.data, rawdata
  Logical, if TRUE, a layer with raw data from response by predictor on the x-axis, plotted as point-geoms, is added to the plot.

colors
  Character vector with color values in hex-format, valid color value names (see demo("colors")) or a name of a ggeffects-color-palette. Following options are valid for colors:
  • If not specified, the color brewer palette "Set1" will be used.
  • If "gs", a greyscale will be used.
  • If "bw", the plot is black/white and uses different line types to distinguish groups.
  • There are some pre-defined color-palettes in this package that can be used, e.g. colors = "metro". See show_pals() to show all available palettes.
  • Else specify own color values or names as vector (e.g. colors = c("#f00000","#00ff00")).

alpha
  Alpha value for the confidence bands.

dodge
  Value for offsetting or shifting error bars, to avoid overlapping. Only applies, if a factor is plotted at the x-axis (in such cases, the confidence bands are replaced by error bars automatically), or if ci.style = "errorbars".

use.theme
  Logical, if TRUE, a slightly tweaked version of ggplot's minimal-theme, theme_ggeffects(), is applied to the plot. If FALSE, no theme-modifications are applied.

dot.alpha
  Alpha value for data points, when add.data = TRUE.

jitter
  Numeric, between 0 and 1. If not NULL and add.data = TRUE, adds a small amount of random variation to the location of data points dots, to avoid overplotting. Hence the points don't reflect exact values in the data. May also be a numeric vector of length two, to add different horizontal and vertical jittering. For binary outcomes, raw data is not jittered by default to avoid that data points exceed the axis limits.
plot

log.y  Logical, if TRUE, the y-axis scale is log-transformed. This might be useful for binomial models with predicted probabilities on the y-axis.

case  Desired target case. Labels will automatically converted into the specified character case. See ?sjlabelled::convert_case for more details on this argument.

show.legend  Logical, shows or hides the plot legend.

show.title  Logical, shows or hides the plot title.

show.x.title  Logical, shows or hides the plot title for the x-axis.

show.y.title  Logical, shows or hides the plot title for the y-axis.

dot.size  Numeric, size of the point geoms.

line.size  Numeric, size of the line geoms.

connect.lines  Logical, if TRUE and plot has point-geoms with error bars (this is usually the case when the x-axis is discrete), points of same groups will be connected with a line.

one.plot  Logical, if TRUE and x has a panel column (i.e. when four terms were used), a single, integrated plot is produced.

...  Further arguments passed down to ggplot::scale_y*, to control the appearance of the y-axis.

base.size  Base font size.

base.family  Base font family.

Details

For proportional odds logistic regression (see ?MASS::polr) or cumulative link models in general, plots are automatically facetted by response.level, which indicates the grouping of predictions based on the level of the model’s response.

Value

A ggplot2-object.

Note

Load library(ggplot2) and use theme_set(theme_ggeffects()) to set the ggeffects-theme as default plotting theme. You can then use further plot-modifiers from sjPlot, like legend_style() or font_size() without losing the theme-modifications.

There are pre-defined colour palettes in this package. Use show_pals() to show all available colour palettes.

Examples

library(sjlabelled)
data(efc)
efc$c172code <- as_label(efc$c172code)
fit <- lm(barthtot ~ cl2hour + neg_c_7 + c161sex + c172code, data = efc)
dat <- ggpredict(fit, terms = "c12hour")
plot(dat)

# facet by group, use pre-defined color palette
dat <- ggpredict(fit, terms = c("c12hour", "c172code"))
plot(dat, facet = TRUE, colors = "hero")

# don't use facets, b/w figure, w/o confidence bands
dat <- ggpredict(fit, terms = c("c12hour", "c172code"))
plot(dat, colors = "bw", ci = FALSE)

# factor at x axis, plot exact data points and error bars
dat <- ggpredict(fit, terms = c("c172code", "c161sex"))
plot(dat)

# for three variables, automatic facetting
dat <- ggpredict(fit, terms = c("c12hour", "c172code", "c161sex"))
plot(dat)

# show all color palettes
show_pals()

---

pretty_range

Create a pretty sequence over a range of a vector

Description

Creates an evenly spaced, pretty sequence of numbers for a range of a vector.

Usage

pretty_range(x, n = NULL, length = NULL)

Arguments

x A numeric vector.

n Numeric value, indicating the size of how many values are used to create a pretty sequence. If x has a large value range (> 100), n could be something between 1 to 5. If x has a rather small amount of unique values, n could be something between 10 to 20. If n = NULL, pretty_range() automatically tries to find a pretty sequence.

length Integer value, as alternative to n, defines the number of intervals to be returned.

Value

A numeric vector with a range corresponding to the minimum and maximum values of x.
Examples

```r
library(sjmisc)
data(efc)
x <- std(efc$c12hour)
x
# pretty range for vectors with decimal points
pretty_range(x)

# pretty range for large range, increasing by 50
pretty_range(1:1000)

# increasing by 20
pretty_range(1:1000, n = 7)

# return 10 intervals
pretty_range(1:1000, length = 10)

# same result
pretty_range(1:1000, n = 2.5)
```

---

values_at

**Calculate representative values of a vector**

Description

This function calculates representative values of a vector, like minimum/maximum values or lower, median and upper quartile etc., which can be used for numeric vectors to plot marginal effects at these representative values.

Usage

```r
values_at(x, values = "meansd")
representative_values(x, values = "meansd")
```

Arguments

- `x`: A numeric vector.
- `values`: Character vector, naming a pattern for which representative values should be calculated.
  - "minmax" (default) minimum and maximum values (lower and upper bounds) of the moderator are used to plot the interaction between independent variable and moderator.
  - "meansd" uses the mean value of the moderator as well as one standard deviation below and above mean value to plot the effect of the moderator on the independent variable.
"zeromax" is similar to the "minmax" option, however, 0 is always used as minimum value for the moderator. This may be useful for predictors that don’t have an empirical zero-value, but absence of moderation should be simulated by using 0 as minimum.

"quart" calculates and uses the quartiles (lower, median and upper) of the moderator value, including minimum and maximum value.

"quart2" calculates and uses the quartiles (lower, median and upper) of the moderator value, excluding minimum and maximum value.

"all" uses all values of the moderator variable. Note that this option only applies to type = "eff", for numeric moderator values.

Value
A numeric vector of length two or three, representing the required values from x, like minimum/maximum value or mean and +/- 1 SD.

Examples

```r
data(efc)
values_at(efc$c12hour)
values_at(efc$c12hour, "quart2")
```

---

**vcov**

*Calculate variance-covariance matrix for marginal effects*

Description

Returns the variance-covariance matrix for the predicted values from object.

Usage

```r
## S3 method for class 'ggeffects'
vcov(object, vcov.fun = NULL, vcov.type = NULL, vcov.args = NULL, ...)
```

Arguments

- **object**: An object of class "ggeffects", as returned by `ggpredict()`.
- **vcov.fun**: String, indicating the name of the `vcov*()`-function from the `sandwich` or `clubSandwich`-package, e.g. `vcov.fun = "vcovCL"`, which is used to compute (cluster) robust standard errors for predictions. If NULL, standard errors (and confidence intervals) for predictions are based on the standard errors as returned by the `predict()`-function. **Note** that probably not all model objects that work with `ggpredict()` are also supported by the `sandwich` or `clubSandwich`-package.
- **vcov.type**: Character vector, specifying the estimation type for the robust covariance matrix estimation (see `?sandwich::vcovHC` or `?clubSandwich::vcovCR` for details).
vcov.args  List of named vectors, used as additional arguments that are passed down to vcov.fun.

...  Currently not used.

Details

The returned matrix has as many rows (and columns) as possible combinations of predicted values from the ggpredict() call. For example, if there are two variables in the terms-argument of ggpredict() with 3 and 4 levels each, there will be 3*4 combinations of predicted values, so the returned matrix has a 12x12 dimension. In short, nrow(object) is always equal to nrow(vcov(object)). See also 'Examples'.

Value

The variance-covariance matrix for the predicted values from object.

Examples

data(efc)
model <- lm(barthtot ~ c12hour + neg_c_7 + c161sex + c172code, data = efc)
result <- ggpredict(model, c("c12hour [meansd]", "c161sex"))
vcov(result)

# compare standard errors
sqrt(diag(vcov(result)))
as.data.frame(result)

# only two predicted values, no further terms
# vcov() returns a 2x2 matrix
result <- ggpredict(model, "c161sex")
vcov(result)

# 2 levels for c161sex multiplied by 3 levels for c172code
# result in 6 combinations of predicted values
# thus vcov() returns a 6x6 matrix
result <- ggpredict(model, c("c161sex", "c172code"))
vcov(result)
Index

* data  
  - efc, 2
  - fish, 2
  - efc, 2
  - efc_test (efc), 2
  - fish, 2

  - get_complete_df (get_title), 3
  - get_legend_labels (get_title), 3
  - get_legend_title (get_title), 3
  - get_title, 3
  - get_x_labels (get_title), 3
  - get_x_title (get_title), 3
  - get_y_title (get_title), 3
  - ggeffect, 4
  - ggemmeans (ggeffect), 4
  - ggpredict (ggeffect), 4

new_data, 12

plot, 13

pretty_range, 8, 16

representative_values (values_at), 17

show_pals (plot), 13

show_pals(), 14

theme_ggeffects (plot), 13

values_at, 8, 17

vcov, 18