Package 'ggeffects'

January 9, 2019

Type Package

Encoding UTF-8

Title Create Tidy Data Frames of Marginal Effects for 'ggplot' from Model Outputs

Version 0.8.0

Date 2019-01-09

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), graphics, stats, utils

Imports crayon, dplyr, ggplot2, lme4, magrittr, MASS, prediction, purrr, rlang, scales, sjlabelled (>= 1.0.14), sjmisc (>= 2.7.6), sjstats (>= 0.17.3), tidyr

Suggests brms, effects (>= 4.0-0), emmeans, glmmTMB, knitr, nlme, pscl, rmarkdown, rstanarm, rstantools, sandwich, survey, survival, testthat

URL https://strengejacke.github.io/ggeffects

BugReports https://github.com/strengejacke/ggeffects/issues

RoxygenNote 6.1.1

VignetteBuilder knitr

NeedsCompilation no

Author Daniel Lüdecke [aut, cre] (<https://orcid.org/0000-0002-8895-3206>)

Repository CRAN

Date/Publication 2019-01-09 16:20:03 UTC

R topics documented:

`efc` *Sample dataset from the EUROFAMCARE project*

Description

A SPSS sample data set, imported with the `read_spss` function.

Examples

```
# Attach EFC-data
data(efc)

# Show structure
str(efc)

# show first rows
head(efc)

# show variables
## Not run:
library(sjmisc)
library(sjPlot)
view_df(efc)

# show variable labels
get_label(efc)

# plot efc-data frame summary
sjt.df(efc, alternateRowColor = TRUE)
## End(Not run)
```

emm *Get marginal effects for model response*

Description

emm() is a convenient shortcut to compute the estimated marginal mean, resp. the marginal effect of the model's response variable, with all independent variables held constant (at their typical_value).

Usage

```
emm(model, ci.lvl = 0.95, type = c("fe", "re", "fe.zi", "re.zi", "sim",
  "surv", "cumhaz"), typical = "mean", condition = NULL, ...)
```

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 ggeffect(), any model that is supported by the **effects**-package should work.
ci.lvl	Numeric, the level of the confidence intervals. For ggpredict(), use ci.lvl = NA, if confidence intervals should not be calculated (for instance, due to computation time).
type	Character, only applies for survival models, mixed effects models and/or models with zero-inflation. **Note:** For brmsfit-models with zero-inflation component, there is no type = "fe.zi" nor type = "re.zi" (see 'Details').
	"fe" Predicted values are conditioned on the fixed effects or conditional model only (for mixed models: predicted values are on the population-level). 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").
	"re" This only applies to mixed models, and type = "re" does not condition on the zero-inflation component of the model. type = "re" still returns population-level predictions, however, unlike type = "fe", prediction intervals also consider the uncertainty in the variance parameters (the mean random effect variance, see re_var and *Johnson et al. 2014* for details). For models with zero-inflation component, this type calls predict(..., type = "link"). 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).
	"fe.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'.

"re.zi" 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'.

"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.

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

"debug" Only used internally.

typical — Character vector, naming the function to be applied to the covariates over which the effect is "averaged". The default is "mean". See 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'.

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

Details

For linear models, the predicted value is the estimated marginal mean. Else, the predicted value is on the scale of the inverse of link function.

Value

A data frame with the marginal effect of the response (predicted), std.error and the confidence intervals conf.low and conf.high. For cumulative link-models, the marginal effect for each level of the response variable is returned.

Examples

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

# Example from ?MASS::polr
library(MASS)
options(contrasts = c("contr.treatment", "contr.poly"))
house.plr <- polr(Sat ~ Infl + Type + Cont, weights = Freq, data = housing)
emm(house.plr)
```

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)

get_x_title(x, case = NULL)

get_y_title(x, case = NULL)

get_legend_title(x, case = NULL)

get_legend_labels(x, case = NULL)

get_x_labels(x, case = NULL)

get_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 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

```
data(efc)
efc$c172code <- sjmisc::to_factor(efc$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
get_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))
```

---

ggaverage *Get marginal effects from model terms*

Description

ggpredict() computes predicted (fitted) values for the response, at the margin of specific values from certain model terms, where additional model terms indicate the grouping structure. ggeffect() computes marginal effects by internally calling Effect, while ggemmeans() computes marginal effects by internally calling emmeans. ggaverage() computes the average predicted values. The result is returned as tidy data frame.

Usage

```
ggaverage(model, terms, ci.lvl = 0.95, type = c("fe", "re", "fe.zi",
  "re.zi"), typical = "mean", ppd = FALSE, x.as.factor = FALSE,
  condition = NULL, ...)

ggeffect(model, terms, ci.lvl = 0.95, x.as.factor = FALSE, ...)

ggemmeans(model, terms, ci.lvl = 0.95, type = c("fe", "re", "fe.zi",
  "re.zi"), typical = "mean", condition = NULL, x.as.factor = FALSE,
  x.cat, ...)

ggpredict(model, terms, ci.lvl = 0.95, type = c("fe", "re", "fe.zi",
  "re.zi", "sim", "surv", "cumhaz", "debug"), typical = "mean",
  condition = NULL, ppd = FALSE, x.as.factor = FALSE,
  full.data = FALSE, vcov.fun = NULL, vcov.type = NULL,
  vcov.args = NULL, x.cat, ...)
```

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 ggeffect(), any model that is supported by the **effects**-package should work.
terms	Character vector (or a formula) with the names of those terms from model, for which marginal effects should be displayed. At least one term is required to calculate effects for certain terms, maximum length is three terms, where the second and third term indicate the groups, i.e. predictions of first term are grouped by the values of the second (and third) term. If terms is missing or 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 terms are held constant (if full.data = FALSE, the default) or are set to the values from the observations (i.e. are kept as they happen to be; see 'Details'). See also argument condition and typical.
ci.lvl	Numeric, the level of the confidence intervals. For ggpredict(), use ci.lvl = NA, if confidence intervals should not be calculated (for instance, due to computation time).
type	Character, only applies for survival models, mixed effects models and/or models with zero-inflation. **Note:** For brmsfit-models with zero-inflation component, there is no type = "fe.zi" nor type = "re.zi" (see 'Details').

"fe" Predicted values are conditioned on the fixed effects or conditional model only (for mixed models: predicted values are on the population-level). 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").

"re" This only applies to mixed models, and type = "re" does not condition on the zero-inflation component of the model. type = "re" still returns population-level predictions, however, unlike type = "fe", prediction intervals also consider the uncertainty in the variance parameters (the mean random effect variance, see re_var and *Johnson et al. 2014* for details). For models with zero-inflation component, this type calls predict(..., type = "link").

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).

"fe.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'.

"re.zi" 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'.

"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.

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

"debug" Only used internally.

typical — Character vector, naming the function to be applied to the covariates over which the effect is "averaged". The default is "mean". See typical_value for options.

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

x.as.factor, x.cat — Logical, if TRUE, preserves factor-class as x-column in the returned data frame (only applies if first variable in terms is a factor). By default, the x-column is always numeric. This argument is useful when building own plots from the data, based on ggplot, so you don't need to coerce x to factor. The plot()-method, however, automatically uses continuous or discrete x-scales, depending on the variable-type. For more details, see the plot-vignette and the the vignette on package-basics. x.cat is an alias for x.as.factor.

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'.

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

full.data — Logical, if TRUE, the returned data frame contains predictions for all observations. This data frame also has columns for residuals and observed values, and can also be used to plot a scatter plot of all data points or fitted values. If FALSE (the default), the returned data frame only contains predictions for all combinations of unique values of the model's predictors. Residuals and observed values are set to NA. Usually, this argument is only used internally by ggaverage().

vcov.fun — String, indicating the name of the vcov*()-function from the **sandwich**-package, e.g. vcov.fun = "vcovCL", which is used to compute 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**-package.

vcov.type — Character vector, specifying the estimation type for the robust covariance matrix estimation (see vcovHC for details).

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

Details

Supported Models

Currently supported model-objects are (in alphabetical order): betareg, brglm, brmsfit, clm, clm2, clmm, coxph, gam (package **mgcv**), Gam (package **gam**), gamm, gamm4, gee, glm, glm.nb, glmer, glmer.nb, glmmTMB, glmmPQL, glmRob, gls, hurdle, lm, lm_robust, lme, lmer, lmRob, lrm, multinom, nlmer, plm, polr, rlm, stanreg, svyglm, svyglm.nb, truncreg, vgam, zeroinfl and zerotrunc. Other models not listed here are passed to a generic predict-function and might work as well, or maybe with ggeffect() or ggemmeans(), which effectively do the same as ggpredict(). The main difference is that ggpredict() calls predict(), while ggeffect() calls Effect and ggemmeans() calls emmeans to compute marginal effects.

Difference between ggpredict() and ggeffect() or ggemmeans()

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 terms-argument also supports the same shortcuts as the values-argument in rprs_values(). 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).

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 rprs_values) 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(), if full.data = FALSE, 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).

ggaverage() computes the average predicted values, by calling ggpredict() with full.data = TRUE, where argument newdata = model.frame(model) is used in predict(). Hence, predictions are made on the model data. In this case, all remaining covariates that are not specified in terms are *not* held constant, but vary between observations (and are kept as they happen to be). The predicted values are then averaged for each group (if any). Thus, ggpredict() can be considered as calculating marginal effects at the mean, while ggaverage() computes average marginal effects.

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 actually high density intervals, computed by hdi, unless ppd = TRUE. If ppd = TRUE, predictions are based on draws of the posterior predictive distribution and the uncertainty interval is computed using predictive_interval. By default (i.e. ppd = FALSE), the predictions are based on 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 (posterior_predict).#'

Zero-Inflated and Zero-Inflated Mixed Models with brms

Models of class brmsfit always condition on the zero-inflation component, if the model has such a component. Hence, there is no type = "fe.zi" nor type = "re.zi" for 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 glmmTMB

If `model` is of class `glmmTMB`, `hurdle`, `zeroinfl` or `zerotrunc`, simulations from a multivariate normal distribution (see `mvrnorm`) are drawn to calculate `mu*(1-p)`. Confidence intervals are then based on quantiles of these results. For `type = "re.zi"`, prediction intervals also take the uncertainty in the random-effect paramters into account (see also Brooks et al. 2017, pp.391-392 for details).

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

Value

A data frame (with `ggeffects` class attribute) with consistent data columns:

`x` the values of the first term in `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.

`observed` if `full.data = TRUE`, this columns contains the observed values (the response vector).

`residuals` if `full.data = TRUE`, this columns contains residuals.

`group` the grouping level from the second term in `terms`, used as grouping-aesthetics in plots.

`facet` the grouping level from the third term in `terms`, used to indicate facets in plots.

For proportional odds logistic regression (see `polr`) resp. cumulative link models (e.g., see `clm`), an additional column `response.level` is returned, which indicates the grouping of predictions based on the level of the model's response.

Note

Since data for `ggaverage()` comes from the model frame, not all possible combinations of values in `terms` might be present in the data, thus lines or confidence bands from `plot()` might not span over the complete x-axis-range.

`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.

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.

References

- Brooks ME, Kristensen K, Benthem KJ van, Magnusson A, Berg CW, Nielsen A, et al. glmmTMB Balances Speed and Flexibility Among Packages for Zero-inflated Generalized Linear Mixed Modeling. The R Journal. 2017;9: 378–400.
- Johnson PC, O'Hara RB. 2014. Extension of Nakagawa & Schielzeth's R2GLMM to random slopes models. Methods Ecol Evol, 5: 944-946. (doi: 10.1111/2041210X.12225)

Examples

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

ggpredict(fit, terms = "c12hour")
ggpredict(fit, terms = "c12hour", full.data = TRUE)
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)) +
  geom_line() +
  geom_ribbon(aes(ymin = conf.low, ymax = conf.high), alpha = .1)

# with "full.data = TRUE", remaining covariates vary between
# observations, so fitted values can be plotted
mydf <- ggpredict(fit, terms = "c12hour", full.data = TRUE)
ggplot(mydf, aes(x, predicted)) + geom_point()

# you can add a smoothing-geom to show the linear trend of fitted values
ggplot(mydf, aes(x, predicted)) +
  geom_smooth(method = "lm", se = FALSE) +
  geom_point()

# three variables, so we can use facets and groups
mydf <- ggpredict(
  fit,
```

```
  terms = c("c12hour", "c161sex", "c172code"),
  full.data = TRUE
)
ggplot(mydf, aes(x = x, y = predicted, colour = group)) +
  stat_smooth(method = "lm", se = FALSE) +
  facet_wrap(~facet, ncol = 2)

# average marginal effects
mydf <- ggaverage(fit, terms = c("c12hour", "c172code"))
ggplot(mydf, aes(x = x, y = predicted, colour = group)) +
  stat_smooth(method = "lm", se = FALSE)

# 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
library(sjlabelled)
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
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_continuous(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
dat <- ggpredict(fit, terms = c("c12hour", "barthtot [30,50,70]", "c161sex"))
ggplot(dat, aes(x = x, y = predicted, colour = group)) +
  stat_smooth(method = "lm", se = FALSE, fullrange = TRUE) +
  facet_wrap(~facet) +
```

```
  labs(
    colour = get_legend_title(dat),
    x = get_x_title(dat),
    y = get_y_title(dat),
    title = get_title(dat)
  )

# or with ggeffects' plot-method
## Not run:
plot(dat, ci = FALSE)
## End(Not run)

# use factor levels as x-column in returned data frame
data(efc)
efc$c161sex <- as_label(efc$c161sex)
fit <- lm(neg_c_7 ~ c12hour + c161sex, data = efc)
ggpredict(fit, terms = "c161sex", x.as.factor = TRUE)

# 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")
```

---

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, rawdata = FALSE, colors = "Set1",
  alpha = 0.15, dodge = 0.1, use.theme = TRUE, dot.alpha = 0.5,
  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, grid, ...)

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.For ggeffects-objects from ggpredict() with argument full.data = TRUE, ci is automatically set to FALSE.
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.
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 color brewer 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. • If colors is any valid color brewer palette name, the related palette will be used. Use display.brewer.all to view all available palette names. • 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 rawdata = TRUE.
jitter	Numeric, between 0 and 1. If not NULL and rawdata = 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. For binary outcomes, raw data is never jittered to avoid that data points exceed the axis limits.
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 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.
...	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

ggpredict() with argument full.data = FALSE computes marginal effects at the mean, where covariates are held constant. In this case, the slope between groups does not vary and the standard errors and confidence intervals have the same "trend" as the predicted values. Hence, plotting confidence bands or error bars is possible. However, ggpredict() with argument full.data = TRUE, covariates and standard errors vary between groups, so plotting confidence bands and error bars would follow a "winding" shape, while the predicted values are smoothened by geom_smooth. Predicted values and confidence bands or error bars would no longer match, thus, ci is automatically set to FALSE in such cases. You still may want to plot objects returned by ggpredict() with argument full.data = TRUE to additionally plot the raw data points, which is automatically done.

For ggaverage(), which computes average marginal effects, the same problem with standard errors and confidence bands would apply. However, the standard errors for the average marginal effects are taken from the marginal effects at the mean, and the predicted values from the average marginal effects are used to compute another regression on these values, to get the "smoothened" values that are used to compute standard errors and confidence intervals that match the predicted values of the average marginal effects (maybe, at this point, it is helpful to inspect the code to better understand what is happening...).

For proportional odds logistic regression (see 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 ~ c12hour + neg_c_7 + c161sex + c172code, data = efc)

dat <- ggpredict(fit, terms = "c12hour")
plot(dat)

dat <- ggpredict(fit, terms = "c12hour", full.data = TRUE)
plot(dat)

dat <- ggaverage(fit, terms = "neg_c_7")
plot(dat)
plot(dat, ci = "dash")

# 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 <- ggaverage(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)
```

---

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)
```

Arguments

x	A numeric vector.

n — Integer 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.

Value

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

Examples

```
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)
```

---

rprs_values *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

```
rprs_values(x, values = "meansd")
```

Arguments

x — A numeric vector.

values — Character vector, naming a pattern for which representative values should be calculcated.

"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

```
data(efc)
rprs_values(efc$c12hour)
rprs_values(efc$c12hour, "quart2")
```

Index