Package ‘ggeffects’

October 17, 2023

Type Package

Encoding UTF-8

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

Version 1.3.2

Maintainer Daniel Lüdecke <d.luedecke@uke.de>

Description Compute marginal effects and adjusted predictions from statistical models and returns the result as tidy data frames. These data frames are ready to use with the 'ggplot2'-package. Effects and predictions 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 (>= 3.6)

Imports graphics, insight (>= 0.19.5), stats

Suggests AER, aod, bayestestR, betareg, brglm2, brms, broom, car, carData, clubSandwich, datawizard (>= 0.9.0), effects (>= 4.2-2), emmeans (>= 1.8.8), gam, gamlss, gamm4, gee, geepack, ggplot2, ggrepel, GLMMadaptive, glmmTMB (>= 1.1.7), gridExtra, haven, htr, jsonlite, knitr, lme4, logistf, magrittr, margins, marginaleffects (>= 0.15.1), MASS, Matrix, mice, MCMCglmm, mgcv, nestedLogit (>= 0.3.0), nlme, ordinal, parameters, patchwork, pscl, quantreg, rmarkdown, rms, robustbase, rstanarm, rstantools, sandwich, see, sjlabelled (>= 1.1.2), sjstats, sjmisc (>= 2.8.2), survey, survival, testthat, tibble, VGAM

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

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

RoxygenNote 7.2.3

VignetteBuilder knitr
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).

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

```r
## S3 method for class 'ggeffects'
as.data.frame(
  x,
  row.names = NULL,
  optional = FALSE,
  ..., 
  stringsAsFactors = FALSE,
  terms_to_colnames = FALSE
)

ggeffect(model, terms, ci_level = 0.95, verbose = TRUE, ci.lvl = ci_level, ...) 

ggemmeans(
  model,
  terms,
  ci_level = 0.95,
  type = "fixed",
  typical = "mean",
  condition = NULL,
  back_transform = TRUE,
  interval = "confidence",
  verbose = TRUE,
  ci.lvl = ci_level,
  back.transform = back_transform,
  ...
)

ggpredict(
  model,
  terms,
  ci_level = 0.95,
  type = "fixed",
  typical = "mean",
  condition = NULL,
  back_transform = TRUE,
  ppd = FALSE,
  vcov_fun = NULL,
  vcov_type = NULL,
  vcov_args = NULL,
  interval,
  verbose = TRUE,
  ci.lvl = ci_level,
  back.transform = back_transform,
  vcov.fun = vcov_fun,
  vcov.type = vcov_type,
  vcov.args = vcov_args,
  ...
)
Arguments

x
An object of class ggeffects, as returned by ggpredict(), ggeffect() or ggemmeans().

row.names
NULL or a character vector giving the row names for the data frame. Missing values are not allowed.

optional
logical. If TRUE, setting row names and converting column names (to syntactic names: see make.names) is optional. Note that all of R’s base package as.data.frame() methods use optional only for column names treatment, basically with the meaning of data.frame(*, check.names = !optional). See also the make.names argument of the matrix method.

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

stringsAsFactors
logical: should the character vector be converted to a factor?

terms_to_colnames
Logical, if TRUE, standardized column names (like "x", "group" or "facet") are replaced by the variable names of the focal predictors specified in terms.

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 effects::Effect() should work, and for ggemmeans(), all models supported by emmeans::emmeans() shall work.

terms
Character vector, (or a named list or a formula) with the names of those terms from model, for which predictions should be displayed (so called focal terms). At least one term is required to calculate 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 meaningful values or levels of the remaining terms (see values_at()). If terms is missing or NULL, adjusted predictions for each model term are calculated (i.e. each model term is used as single focal term). It is also possible to define specific values for focal terms, at which adjusted predictions should be calculated (see 'Details'). All remaining covariates that are not specified in terms are held constant (see 'Details'). See also arguments condition and typical.

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

verbose
Toggle messages or warnings.

ci.lvl, vcov.fun, vcov.type, vcov.args, back.transform
Deprecated arguments. Please use ci_level, vcov_fun, vcov_type, vcov_args and back_transform instead.
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 = "zero_inflated" nor type = "zi_random"; predicted values for MixMod-models from GLMMadaptive with zero-inflation component
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, i.e. `re.form = NA` when calling `predict()`).
  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, conditioned on random
  effects and considering individual level predictions, i.e. `re.form = NULL
  when calling predict()`. This may affect the returned predicted values,
  depending on whether `REML = TRUE` or `REML = FALSE` was used for model
  fitting. Furthermore, 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 pre-
  diction 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 confi-
  dence 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")`; mod-
els from \texttt{pscl} call \texttt{predict(..., type = "zero")} and for \texttt{GLMMAdaptive}, \texttt{predict(..., type = "zero_part")} is called.

- "simulate" (or "sim")
  Predicted values and confidence resp. prediction intervals are based on simulations, i.e. calls to \texttt{simulate()}. This type of prediction takes all model uncertainty into account, including random effects variances. Currently supported models are objects of class \texttt{lm, glm, glmmTMB, wbm, MixMod} and \texttt{merMod}. See ... for details on number of simulations.

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

\textbf{typical}
Character vector, naming the function to be applied to the covariates (non-focal terms) over which the effect is "averaged". The default is "mean". Can be "mean", "weighted.mean", "median", "mode" or "zero", which call the corresponding R functions (except "mode", which calls an internal function to compute the most common value); "zero" simply returns 0. By default, if the covariate is a factor, only "mode" is applicable; for all other values (including the default, "mean") the reference level is returned. For character vectors, only the mode is returned. You can use a named vector to apply different functions to integer, numeric and categorical covariates, e.g. \texttt{typical = c(numeric = "median", factor = "mode")}. If \texttt{typical} is "weighted.mean", weights from the model are used. If no weights are available, the function falls back to "mean".

\textbf{condition}
Named character vector, which indicates covariates that should be held constant at specific values. Unlike \texttt{typical}, which applies a function to the covariates to determine the value that is used to hold these covariates constant, \texttt{condition} can be used to define exact values, for instance \texttt{condition = c(covariate1 = 20, covariate2 = 5)}. See 'Examples'.

\textbf{back_transform}
Logical, if \texttt{TRUE} (the default), predicted values for log- or log-log transformed responses will be back-transformed to original response-scale.

\textbf{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) to account for the uncertainty of predicted values. For mixed models, \texttt{interval = "prediction"} is the default for \texttt{type = "random"}. When \texttt{type = "fixed"}, the default is \texttt{interval = "confidence"}. Note that prediction intervals are not available for all models, but only for models that work with \texttt{insight::get_sigma()}.

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

\textbf{vcov_fun}
Variance-covariance matrix used to compute uncertainty estimates (e.g., for confidence intervals based on robust standard errors). This argument accepts a variance-covariance matrix, a function which returns a covariance matrix, or a string which identifies the function to be used to compute the covariance matrix.

- A (variance-covariance) matrix
- A function which returns a covariance matrix (e.g., \texttt{stats::vcov()})
• A string which indicates the name of the `vcov*()`-function from the *sandwich* or *clubSandwich* packages, 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* packages.

See details in this vignette.

**vcov_type**
Character vector, specifying the estimation type for the robust covariance matrix estimation (see `?sandwich::vcovHC` or `?clubSandwich::vcovCR` for details). Only used when `vcov_fun` is a character string indicating on of the function from those packages.

**vcov_args**
List of named vectors, used as additional arguments that are passed down to `vcov_fun`.

**Details**

**Supported Models**
A list of supported models can be found at the package website. 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 predicted values. Thus, effects returned by `ggpredict()` can be described as conditional effects (i.e. these are conditioned on certain (reference) levels of factors), while `ggemmeans()` and `ggeffect()` return marginal means, since the effects are "marginalized" (or "averaged") over the levels of factors (or values of character vectors). Therefore, `ggpredict()` and `ggeffect()` resp. `ggemmeans()` differ in how factors and character vectors are held constant: `ggpredict()` uses the reference level (or "lowest" value in case of character vectors), 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 and Adjusted Predictions at Specific Values**

Meaningful values of focal terms can be specified via the `terms` argument. Specifying meaningful or representative values as string pattern is the preferred way in the `ggeffects` package. However, it is also possible to use a `list()` for the focal terms if prefer the "classical" R way, which is described in this vignette.

Indicating levels in square brackets allows for selecting only certain groups or values resp. value ranges. The 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 ranges 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. \texttt{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 \texttt{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 \( v = \{1000, 2000, 3000\} \) and then use \texttt{terms = "income \[v\]}".

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

Finally, numeric vectors for which no specific values are given, a "pretty range" is calculated (see \texttt{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 \texttt{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. \texttt{terms = "age \[all\]}". See also package vignettes.

Holding covariates at constant values

For \texttt{ggpredict()}, a data grid is constructed, roughly comparable to \texttt{expand.grid()} on all unique combinations of \texttt{model.frame(model)[, terms]}. This data grid (see \texttt{data_grid()}) as newdata argument for \texttt{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 \texttt{condition} or \texttt{typical} argument), integer values are set to their median, factors are set to their reference level (may also be changed with \texttt{condition}) and character vectors to their mode (most common element).

\texttt{ggeffect()} and \texttt{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. The same applies to character vectors: \texttt{ggemmeans()} averages over the distribution of unique values in a character vector, similar to how factors are treated. For \texttt{ggemmeans()}, use \texttt{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

\texttt{ggpredict()} also works with \texttt{Stan}-models from the \texttt{rstanarm} or \texttt{brms}-packages. The predicted values are the median value of all drawn posterior samples. The confidence intervals for \texttt{Stan}-models are Bayesian predictive intervals. By default (i.e. \texttt{ppd = FALSE}), the predictions are based on \texttt{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 type = "zero_inflated" nor 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 glmmTMB, hurdle, zeroInfl or zeroTrunc, simulations from a multivariate normal distribution (see ?MASS::mvrnorm) are drawn to calculate $mu(1-p)$. Confidence intervals are then based on quantiles of these results. For 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 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).

**MixMod-models from GLMMadaptive**

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

**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. *Note that the standard errors are always on the link-scale, and not back-transformed for non-Gaussian models!*
- "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 terms, used as grouping-aesthetics in plots.
- "facet": the grouping level from the third term in terms, used to indicate facets in plots.

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

For proportional odds logistic regression (see ?MASS::polr) resp. cumulative link models (e.g., see ?ordinal::clm), an additional column "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 "conf.low" and "conf.high", even though for Bayesian models credible or highest posterior density intervals are returned.

There is an as.data.frame() method for objects of class ggeffects, which has an terms_to_colnames argument, to use the term names as column names instead of the standardized names "x" etc.

**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.

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 Github.

References


Examples

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

# ggpredict() call

# specified as formula
ggpredict(fit, terms = "c12hour")

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

# terms as named list
ggpredict(fit, terms = list(c12hour = 40:60))

# 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 ggpplot by hand.

# plot predicted values, remaining covariates held constant
library(ggplot2)
mydf <- ggpredict(fit, terms = "c12hour")
ggplot(mydf, aes(x, predicted)) +
ggplot2::geom_line() +
ggplot2::geom_ribbon(aes(ymin = conf.low, ymax = conf.high), alpha = 0.1)
```
# 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$c172code <- sjlabelled::as_label(efc$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]")
)

# when "terms" is a named list
ggpredict(fit, terms = list(  
c12hour = seq(0, 170, 30),
  c172code = c("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("c12hour", "c161sex"))
ggplot(dat, aes(x, predicted, colour = group)) +
  geom_point(position = position_dodge(0.1)) +
  geom_errorbar(aes(ymin = conf.low, ymax = conf.high),
                position = position_dodge(0.1)) +
  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
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) +
collapse_by_group

Collapse raw data by random effect groups

Description

This function extracts the raw data points (i.e., the data that was used to fit the model) and "averages" (i.e., "collapses") the response variable over the levels of the grouping factor given in collapse_by. Only works with mixed models.

Usage

collapse_by_group(
  grid,
  model,
  collapse_by = NULL,
  residuals = FALSE,
  collapse.by = collapse_by
)

Arguments

grid A data frame representing the data grid, or an object of class ggeffects, as returned by ggpredict() and others.
model The model for which to compute partial residuals. The data grid grid should match to predictors in the model.
collapse_by Name of the (random effects) grouping factor. Data is collapsed by the levels of this factor.
Logical, if TRUE, collapsed partial residuals instead of raw data by the levels of
the grouping factor.

Deprecated. Use collapse_by instead.

Value

A data frame with raw data points, averaged over the levels of the given grouping factor from the
random effects. The group level of the random effect is saved in the column "random".

Examples

```r
library(ggeffects)
if (require("lme4", quietly = TRUE)) {
  data(efc)
  efc$e15relat <- as.factor(efc$e15relat)
  efc$c161sex <- as.factor(efc$c161sex)
  levels(efc$c161sex) <- c("male", "female")
  model <- lmer(neg_c_7 ~ c161sex + (1 | e15relat), data = efc)
  me <- ggpredict(model, terms = "c161sex")
  head(attributes(me)$rawdata)
  collapse_by_group(me, model, "e15relat")
}
```

efc

Sample dataset from the EUROFAMCARE project

Description

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

Examples

```r
# 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)
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 ?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 predicted values for each term.

Examples

if (require("sjmisc", quietly = TRUE) && require("ggplot2", quietly = TRUE) && require("effects", quietly = TRUE)) {
data(efc)
tfc$c172code <- to_factor(efc$c172code)
fit <- lm(barthtot ~ c12hour + neg_c_7 + c161sex + c172code, data = efc)
mydf <- ggpredict(fit, terms = c("c12hour", "c161sex", "c172code"))

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)
  )

# adjusted predictions, a list of data frames (one data frame per term)
eff <- ggeffect(fit)
eff
get_complete_df(eff)

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

hypothesis_test  
(Pairwise) comparisons between predictions

Description

Function to test differences of adjusted predictions for statistical significance. This is usually called contrasts or (pairwise) comparisons. test_predictions() is an alias.

Usage

hypothesis_test(model, ...)
test_predictions(model, ...)

## Default S3 method:
hypothesis_test(
  model,
  terms = NULL,
  by = NULL,
  test = "pairwise",
  equivalence = NULL,
  scale = "response",
  p_adjust = NULL,
  df = NULL,
  ci_level = 0.95,
collapse_levels = FALSE,
verbose = TRUE,
ci.lvl = ci_level,
...
)

## S3 method for class 'ggeffects'
hypothesis_test(
  model,
  by = NULL,
  test = "pairwise",
  equivalence = NULL,
  p_adjust = NULL,
  df = NULL,
  collapse_levels = FALSE,
  verbose = TRUE,
  ...
)

### Arguments

- **model**: A fitted model object, or an object of class ggeffects.
- **terms**: Character vector with the names of the focal terms from model, for which contrasts or comparisons should be displayed. At least one term is required, maximum length is three terms. If the first focal term is numeric, contrasts or comparisons for the slopes of this numeric predictor are computed (possibly grouped by the levels of further categorical focal predictors).
- **by**: Character vector specifying the names of predictors to condition on. Hypothesis test is then carried out for focal terms by each level of by variables. This is useful especially for interaction terms, where we want to test the interaction within "groups". by is only relevant for categorical predictors.
- **test**: Hypothesis to test. By default, pairwise-comparisons are conducted. See section *Introduction into contrasts and pairwise comparisons*.
- **equivalence**: ROPE’s lower and higher bounds. Should be "default" or a vector of length two (e.g., c(-0.1, 0.1)). If "default", bayestestR::rope_range() is used. Instead of using the equivalence argument, it is also possible to call the equivalence_test() method directly. This requires the parameters package to be loaded. When using equivalence_test(), two more columns with information about the ROPE coverage and decision on H0 are added. Furthermore, it is possible to plot() the results from equivalence_test(). See bayestestR::equivalence_test() resp. parameters::equivalence_test.lm() for details.
**scale** Character string, indicating the scale on which the contrasts or comparisons are represented. Can be one of:

- "response" (default), which would return contrasts on the response scale (e.g. for logistic regression, as probabilities);
- "link" to return contrasts on scale of the linear predictors (e.g. for logistic regression, as log-odds);
- "probability" (or "probs") returns contrasts on the probability scale, which is required for some model classes, like MASS::polr();
- "oddsratios" to return contrasts on the odds ratio scale (only applies to logistic regression models);
- "irr" to return contrasts on the odds ratio scale (only applies to count models);
- or a transformation function like "exp" or "log", to return transformed (exponentiated respectively logarithmic) contrasts; note that these transformations are applied to the *response scale*.

**Note:** If the scale argument is not supported by the provided model, it is automatically changed to a supported scale-type (a message is printed when verbose = TRUE).

**p_adjust** Character vector, if not NULL, indicates the method to adjust p-values. See stats::p.adjust() for details. Further possible adjustment methods are "tukey" or "sidak". Some caution is necessary when adjusting p-value for multiple comparisons. See also section *P-value adjustment* below.

**df** Degrees of freedom that will be used to compute the p-values and confidence intervals. If NULL, degrees of freedom will be extracted from the model using insight::get_df() with type = "wald".

**ci_level** Numeric, the level of the confidence intervals.

**collapse_levels** Logical, if TRUE, term labels that refer to identical levels are no longer separated by ",", but instead collapsed into a unique term label (e.g., "level a-level a" becomes "level a"). See 'Examples'.

**verbose** Toggle messages and warnings.

**ci.lvl** Deprecated, please use ci_level.

**Value**

A data frame containing predictions (e.g. for test = NULL), contrasts or pairwise comparisons of adjusted predictions or estimated marginal means.

**Introduction into contrasts and pairwise comparisons**

There are many ways to test contrasts or pairwise comparisons. A detailed introduction with many (visual) examples is shown in this vignette.
P-value adjustment for multiple comparisons

Note that p-value adjustment for methods supported by `p.adjust()` (see also `p.adjust.methods`), each row is considered as one set of comparisons, no matter which test was specified. That is, for instance, when `hypothesis_test()` returns eight rows of predictions (when `test = NULL`), and `p_adjust = "bonferroni"`, the p-values are adjusted in the same way as if we had a test of pairwise comparisons (`test = "pairwise"`) where eight rows of comparisons are returned. For methods “tukey” or “sidak”, a rank adjustment is done based on the number of combinations of levels from the focal predictors in terms. Thus, the latter two methods may be useful for certain tests only, in particular pairwise comparisons.

See Also

There is also an `equivalence_test()` method in the `parameters` package (`parameters::equivalence_test.lm()`), which can be used to test contrasts or comparisons for practical equivalence. This method also has a `plot()` method, hence it is possible to do something like:

```r
library(parameters)
ggpredict(model, focal_terms) |> equivalence_test() |> plot()
```

Examples

```r
data(efc)
ecf$c172code <- as.factor(ecf$c172code)
ecf$c161sex <- as.factor(ecf$c161sex)
levels(ecf$c161sex) <- c("male", "female")
m <- lm(barthtot ~ c12hour + neg_c_7 + c161sex + c172code, data = efc)

# direct computation of comparisons
hypothesis_test(m, "c172code")

# passing a `ggeffects` object
pred <- ggpredict(m, "c172code")
hypothesis_test(pred)

# test for slope
hypothesis_test(m, "c12hour")

# interaction - contrasts by groups
m <- lm(barthtot ~ c12hour + c161sex * c172code + neg_c_7, data = efc)
hypothesis_test(m, c("c161sex", "c172code"), test = NULL)

# interaction - pairwise comparisons by groups
hypothesis_test(m, c("c161sex", "c172code"))

# equivalence testing
hypothesis_test(m, c("c161sex", "c172code"), equivalence = c(-2.96, 2.96))
```
# equivalence testing, using the parameters package
pr <- ggpredict(m, c("c161sex", "c172code"))
parameters::equivalence_test(pr)

# interaction - collapse unique levels
hypothesis_test(m, c("c161sex", "c172code"), collapse_levels = TRUE)

# p-value adjustment
hypothesis_test(m, c("c161sex", "c172code"), p_adjust = "tukey")

# not all comparisons, only by specific group levels
hypothesis_test(m, "c172code", by = "c161sex")

# specific comparisons
hypothesis_test(m, c("c161sex", "c172code"), test = "b2 = b1")

# interaction - slope by groups
m <- lm(barthtot ~ c12hour + neg_c_7 * c172code + c161sex, data = efc)
hypothesis_test(m, c("neg_c_7", "c172code"))

---

**install_latest**

Update latest ggeffects-version from R-universe (GitHub) or CRAN

### Description

This function can be used to install the latest package version of *ggeffects*, either the development version (from R-universe/GitHub) or the current version from CRAN.

### Usage

```r
install_latest(
  source = c("development", "cran"),
  force = FALSE,
  verbose = TRUE
)
```

### Arguments

- **source** Character. Either "development" or "cran". If "cran", *ggeffects* will be installed from the default CRAN mirror returned by `getOption("repos")["CRAN"]`. If "development" (the default), *ggeffects* is installed from the r-universe repository (https://strengejacke.r-universe.dev/).
- **force** Logical, if FALSE, the update will only be installed if a newer version is available. Use force=TRUE to force installation, even if the version number for the locally installed package is identical to the latest development-version. Only applies when source="development".
- **verbose** Toggle messages.
Value

Invisible NULL.

Examples

```r
# install latest development-version of ggeffects from the # r-universe repository
install_latest()
```

---

**johnson_neyman**

*Spotlight-analysis: Create Johnson-Neyman confidence intervals and plots*

---

Description

Function conduct a spotlight-analysis to create so-called Johnson-Neyman intervals. The `plot()` method can be used to visualize the results of the Johnson-Neyman test.

Usage

```r
johnson_neyman(x, precision = 500, ...)
spotlight_analysis(x, precision = 500, ...)
```

```r
## S3 method for class 'ggjohnson_neyman'
plot(
x,
   colors = c("#f44336", "#2196F3"),
   show_association = TRUE,
   show_rug = FALSE,
   verbose = TRUE,
   ...
)
```

Arguments

- **x**: An object of class `ggeffects`, as returned by the functions from this package.
- **precision**: Number of values used for the range of the moderator variable to calculate the Johnson-Neyman interval. This argument is passed down to `pretty(...)`, namely `n = precision`). Usually, the default value of 500 is sufficient. Increasing this value will result in a smoother plot and more accurate values for the interval bounds, but can also slightly increase the computation time.
- **...**: Arguments passed down to `hypothesis_test()` (and then probably further to `marginaleffects::slopes()`).
The Johnson-Neyman intervals help to understand where slopes are significant in the context of interactions in regression models. Thus, the interval is only useful if the model contains at least one interaction term. The function accepts the results of a call to `ggpredict()`, `ggeffect()` or `ggemmeans()`. The first and the last focal term used in the terms argument of `ggpredict()` etc. must be numeric. The function will then test the slopes of the first focal terms against zero, for different moderator values of the last focal term. Use `plot()` to create a plot of the results.

To avoid misleading interpretations of the plot, we speak of "positive" and "negative" associations, respectively, and "no clear" associations (instead of "significant" or "non-significant"). This should prevent the user from considering a non-significant range of values of the moderator as "accepting the null hypothesis".

**Value**

A Johnson-Neyman plot.

**References**


**Examples**

```r
# Not run:
data(efc)
efc$c172code <- as.factor(efc$c172code)
```
m <- lm(neg_c_7 ~ c12hour * barthtot * c172code, data = efc)

pr <- ggpredict(m, c("c12hour", "barthtot"))
johnson_neyman(pr)
plot(johnson_neyman(pr))

pr <- ggpredict(m, c("c12hour", "c172code", "barthtot"))
johnson_neyman(pr)
plot(johnson_neyman(pr))

# robust standard errors
if (requireNamespace("sandwich")) {
  johnson_neyman(pr, vcov = sandwich::vcovHC)
}

## End(Not run)

---

### lung2

**Sample data set**

A sample data set, used in tests and examples for survival models. This dataset is originally included in the `survival` package, but for convenience reasons it is also available in this package.

---

### new_data

**Create a data frame from all combinations of predictor values**

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, ...)
```

```
data_grid(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. This argument works in the same way as the terms argument in ggpredict(). See also this vignette.

typical  
Character vector, naming the function to be applied to the covariates (non-focal terms) over which the effect is "averaged". The default is "mean". Can be "mean", "weighted.mean", "median", "mode" or "zero", which call the corresponding R functions (except "mode", which calls an internal function to compute the most common value); "zero" simply returns 0. By default, if the covariate is a factor, only "mode" is applicable; for all other values (including the default, "mean") the reference level is returned. For character vectors, only the mode is returned. You can use a named vector to apply different functions to integer, numeric and categorical covariates, e.g. typical = c(numeric = "median", factor = "mode"). If typical is "weighted.mean", weights from the model are used. If no weights are available, the function falls back to "mean".

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
nd

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

plot  
Plot ggeffects-objects
Description

A generic plot-method for ggeffects-objects.

Usage

```r
## S3 method for class 'ggeffects'
plot(
  x,
  show_ci = TRUE,
  ci_style = c("ribbon", "errorbar", "dash", "dot"),
  show_data = FALSE,
  show_residuals = FALSE,
  show_residuals_line = FALSE,
  data_labels = FALSE,
  limit_range = FALSE,
  collapse_group = FALSE,
  show_legend = TRUE,
  show_title = TRUE,
  show_x_title = TRUE,
  show_y_title = TRUE,
  case = NULL,
  colors = NULL,
  alpha = 0.15,
  dot_alpha = 0.35,
  jitter = NULL,
  dodge = 0.25,
  dot_size = NULL,
  line_size = NULL,
  use_theme = TRUE,
  log_y = FALSE,
  connect_lines = FALSE,
  facets,
  grid,
  one_plot = TRUE,
  verbose = TRUE,
  ci = show_ci,
  ci.style = ci_style,
  rawdata = show_data,
  add.data = show_data,
  residuals = show_residuals,
  residuals.line = show_residuals_line,
  label.data = data_labels,
  limit.range = limit_range,
  collapse.group = collapse_group,
  dot.alpha = dot_alpha,
  dot.size = dot_size,
  line.size = line_size,
  connect.lines = connect_lines,
```
show.title = show_title,
show.x.title = show_x_title,
show.y.title = show_y_title,
use.theme = use_theme,
show.legend = show_legend,
one.plot = one_plot,
log.y = log_y,
...

theme_ggeffects(base_size = 11, base_family = "")
show_pals()

Arguments

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

show.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.

show.data
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.

show_residuals
Logical, if TRUE, a layer with partial residuals is added to the plot. See vignette Effect Displays with Partial Residuals. from effects for more details on partial residual plots.

show_residuals_line
Logical, if TRUE, a loess-fit line is added to the partial residuals plot. Only applies if residuals is TRUE.

data_labels
Logical, if TRUE and row names in data are available, data points will be labelled by their related row name.

limit_range
Logical, if TRUE, limits the range of the prediction bands to the range of the data.

collapse_group
For mixed effects models, name of the grouping variable of random effects. If collapse_group = TRUE, data points "collapsed" by the first random effect groups are added to the plot. Else, if collapse_group is a name of a group factor, data is collapsed by that specific random effect. See collapse_by_group() for further details.

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.

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

dot_alpha  Alpha value for data points, when show_data = TRUE.

jitter  Numeric, between 0 and 1. If not NULL and show_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.

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".

dot_size  Numeric, size of the point geoms.

line_size  Numeric, size of the line geoms.

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.

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.

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.

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.

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

verbose  Logical, toggle warnings and messages.

...  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.

### Partial Residuals

For **generalized linear models** (glm), residualized scores are computed as `inv.link(link(Y) + r)` where `Y` are the predicted values on the response scale, and `r` are the _working_ residuals.

For **(generalized) linear mixed models**, the random effect are also partialled out.

### 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, e.g. 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

```r
library(sjlabelled)
data(efc)
ecf$c172code <- as_label(efc$c172code)
fit <- lm(barthtot ~ c12hour + 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", show_ci = FALSE)

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

Pool Predictions or Estimated Marginal Means

Description

This function "pools" (i.e. combines) multiple ggeffects objects, in a similar fashion as mice::pool().

Usage

pool_predictions(x, ...)

Arguments

x

A list of ggeffects objects, as returned by ggpredict(). ggemmeans() or ggeffect().

...

Currently not used.

Details

Averaging of parameters follows Rubin's rules (Rubin, 1987, p. 76).

Value

A data frame with pooled predictions.

References


Examples

# example for multiple imputed datasets
if (require("mice")) {
  data("nhanes2")
  imp <- mice(nhanes2, printFlag = FALSE)
  predictions <- lapply(1:5, function(i) {
    m <- lm(bmi ~ age + hyp + chl, data = complete(imp, action = i))
    ggpredict(m, "age")
    })
}
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. If x is missing, a function, pre-programmed with n and length is returned. See examples.

Examples

data(iris)
# pretty range for vectors with decimal points
pretty_range(iris$Petal.Length)

# 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)

# function factory
residualize_over_grid
Compute partial residuals from a data grid

Description
This function computes partial residuals based on a data grid, where the data grid is usually a data frame from all combinations of factor variables or certain values of numeric vectors. This data grid is usually used as newdata argument in predict(), and can be created with new_data().

Usage
residualize_over_grid(grid, model, ...)  
## S3 method for class 'data.frame'  
residualize_over_grid(grid, model, predictor_name, ...)  
## S3 method for class 'ggeffects'  
residualize_over_grid(grid, model, protect_names = TRUE, ...)

Arguments
grid A data frame representing the data grid, or an object of class ggeffects, as returned by ggpredict() and others.
model The model for which to compute partial residuals. The data grid grid should match to predictors in the model.
... Currently not used.
predictor_name The name of the focal predictor, for which partial residuals are computed.
protect_names Logical, if TRUE, preserves column names from the ggeffects objects that is used as grid.

Value
A data frame with residuals for the focal predictor.

Partial Residuals
For generalized linear models (glms), residualized scores are computed as inv.link(link(Y) + r) where Y are the predicted values on the response scale, and r are the working residuals.

For (generalized) linear mixed models, the random effect are also partialed out.

References
Examples

```r
library(ggeffects)
set.seed(1234)
x <- rnorm(200)
z <- rnorm(200)
# quadratic relationship
y <- 2 * x + x^2 + 4 * z + rnorm(200)
d <- data.frame(x, y, z)
model <- lm(y ~ x + z, data = d)
pr <- ggpredict(model, c("x [all]", "z"))
head(residualize_over_grid(pr, model))
```

values_at

`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.
  - "fivenum": calculates and uses the Tukey’s five number summary (minimum, lower-hinge, median, upper-hinge, maximum) of the moderator value.
• "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.
• "terciles": calculates and uses the terciles (lower and upper third) of the moderator value, including minimum and maximum value.
• "terciles2": calculates and uses the terciles (lower and upper third) 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. If x is missing, a function, pre-programmed with n and length is returned. See examples.

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

mean_sd <- values_at(values = "meansd")
mean_sd(efc$c12hour)
```

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,
  vcov.fun = vcov_fun,
  vcov.type = vcov_type,
  vcov.args = vcov_args,
  ...
)
```
vcov

Arguments

object An object of class "ggeffects", as returned by ggpredict().

vcov_fun Variance-covariance matrix used to compute uncertainty estimates (e.g., for confidence intervals based on robust standard errors). This argument accepts a covariance matrix, a function which returns a covariance matrix, or a string which identifies the function to be used to compute the covariance matrix.

- A (variance-covariance) matrix
- A function which returns a covariance matrix (e.g., stats::vcov())
- A string which indicates the name of the vcov\*()-function from the sandwich or clubSandwich packages, 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 packages.

See details in this vignette.

vcov_type Character vector, specifying the estimation type for the robust covariance matrix estimation (see ?sandwich::vcovHC or ?clubSandwich::vcovCR for details). Only used when vcov_fun is a character string indicating on of the function from those packages.

vcov_args List of named vectors, used as additional arguments that are passed down to vcov_fun.

vcov.fun, vcov.type, vcov.args Deprecated. Use vcov_fun, vcov_type and vcov_args instead.

... 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")
v cov(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"))
v cov(result)
Index

* data
  - efec, 13
  - fish, 13
  - lung2, 22
  - as.data.frame(), 9
  - as.data.frame.ggeffects, 2
  - bayestestR::equivalence_test(), 16
  - bayestestR::rope_range(), 16
  - collapse_by_group, 12
  - collapse_by_group(), 25
  - data.frame, 4
  - data_grid(new_data), 22
  - data_grid(), 8, 16
  - efec, 13
  - efec_test(efc), 13
  - effects::Effect(), 2
  - emmeans::emmeans(), 2
  - fish, 13
  - get_complete_df(get_title), 14
  - get_legend_labels(get_title), 14
  - get_legend_title(get_title), 14
  - get_title, 14
  - get_x_labels(get_title), 14
  - get_x_title(get_title), 14
  - get_y_title(get_title), 14
  - ggeffect(as.data.frame.ggeffects), 2
  - ggemmeans(as.data.frame.ggeffects), 2
  - ggformula(as.data.frame.ggeffects), 2
  - ggpredict(as.data.frame.ggeffects), 2
  - ggpredict(), 28
  - hypothesis_test, 15
  - insight::get_df(), 17
  - insight::get_sigma(), 6
  - install_latest, 19
  - johnson_neyman, 20
  - lung2, 22
  - make.names, 4
  - marginaleffects::predictions(), 16
  - marginaleffects::slopes(), 16, 20
  - mice::pool(), 28
  - new_data, 22
  - new_data(), 30
  - parameters::equivalence_test.lm(), 16, 18
  - plot, 23
  - plot.ggjohnson_neyman(johnson_neyman), 20
  - pool_predictions, 28
  - predict(), 2
  - pretty_range, 29
  - pretty_range(), 8
  - representative_values(values_at), 31
  - residualize_over_grid, 30
  - rstantools::posterior_linpred(), 6, 8
  - rstantools::posterior_predict(), 6, 8
  - show_pals(plot), 23
  - show_pals(), 26, 27
  - spotlight_analysis(johnson_neyman), 20
  - stats::p.adjust(), 17
  - test_predictions(hypothesis_test), 15
  - theme_ggeffect, 23
  - values_at, 31
  - values_at(), 4, 8, 9
  - vcov, 32

35