

# Package ‘ggeffects’

May 4, 2023

**Type** Package

**Encoding** UTF-8

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

**Version** 1.2.2

**Maintainer** Daniel Lüdtke <d.luedtke@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 (>= 0.19.0), stats

**Suggests** AER, aod, bayestestR, betareg, brms, clubSandwich, datawizard, effects (>= 4.1-2), emmeans (>= 1.4.1), gam, gamlss, gamm4, gee, geopack, ggplot2, GLMMadaptive, glmmTMB (>= 1.0.0), gridExtra, haven, httr, jsonlite, knitr, lme4, logistf, magrittr, margins, marginaleffects (>= 0.10.0), MASS, Matrix, mice, MCMCglmm, mgcv, 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, VGAM

**URL** <https://strengjacke.github.io/ggeffects/>

**BugReports** <https://github.com/strengjacke/ggeffects/issues/>

**RoxygenNote** 7.2.3

**VignetteBuilder** knitr

**Config/testthat/edition** 3

**NeedsCompilation** no

**Author** Daniel Lüdtke [aut, cre] (<<https://orcid.org/0000-0002-8895-3206>>),  
 Frederik Aust [ctb] (<<https://orcid.org/0000-0003-4900-788X>>),  
 Sam Crawley [ctb] (<<https://orcid.org/0000-0002-7847-0411>>),  
 Mattan S. Ben-Shachar [ctb] (<<https://orcid.org/0000-0002-4287-4801>>)

**Repository** CRAN

**Date/Publication** 2023-05-04 09:00:05 UTC

## R topics documented:

as.data.frame.ggeffects	2
collapse_by_group	11
efc	12
fish	12
get_title	13
hypothesis_test	14
install_latest	17
lung2	18
new_data	18
plot	19
pool_predictions	23
pretty_range	24
residualize_over_grid	25
values_at	26
vcov	27
<b>Index</b>	<b>30</b>

---

as.data.frame.ggeffects

*Marginal effects, adjusted predictions and estimated marginal means  
 from regression models*

---

## Description

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

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

**Usage**

```
## S3 method for class 'ggeffects'  
as.data.frame(  
  x,  
  row.names = NULL,  
  optional = FALSE,  
  ...,  
  stringsAsFactors = FALSE,  
  terms_to_colnames = FALSE  
)  
  
ggeffect(model, terms, ci.lvl = 0.95, verbose = TRUE, ...)  
  
ggemmeans(  
  model,  
  terms,  
  ci.lvl = 0.95,  
  type = "fe",  
  typical = "mean",  
  condition = NULL,  
  back.transform = TRUE,  
  interval = "confidence",  
  verbose = TRUE,  
  ...  
)  
  
ggpredict(  
  model,  
  terms,  
  ci.lvl = 0.95,  
  type = "fe",  
  typical = "mean",  
  condition = NULL,  
  back.transform = TRUE,  
  ppd = FALSE,  
  vcov.fun = NULL,  
  vcov.type = NULL,  
  vcov.args = NULL,  
  interval = "confidence",  
  verbose = TRUE,  
  ...  
)
```

**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 <code>make.names</code> ) is optional. Note that all of R's <code>base</code> package <code>as.data.frame()</code> methods use <code>optional</code> only for column names treatment, basically with the meaning of <code>data.frame(*, check.names = !optional)</code> . See also the <code>make.names</code> argument of the <code>matrix</code> method.
...	For <code>ggpredict()</code> , further arguments passed down to <code>predict()</code> ; for <code>ggeffect()</code> , further arguments passed down to <code>effects::Effect()</code> ; and for <code>ggemmeans()</code> , further arguments passed down to <code>emmeans::emmeans()</code> . If <code>type = "sim"</code> , ... may also be used to set the number of simulation, e.g. <code>nsim = 500</code> .
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 <code>terms</code> .
model	A fitted model object, or a list of model objects. Any model that supports common methods like <code>predict()</code> , <code>family()</code> or <code>model.frame()</code> should work. For <code>ggeffect()</code> , any model that is supported by <b>effects</b> should work, and for <code>ggemmeans()</code> , all models supported by <b>emmeans</b> should work.
terms	Character vector, (or a named list or a formula) with the names of those terms from <code>model</code> , for which predictions should be displayed. 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 the values or levels of the remaining terms. If <code>terms</code> is missing or NULL, adjusted predictions for each model term are calculated. It is also possible to define specific values for terms, at which adjusted predictions should be calculated (see 'Details'). All remaining covariates that are not specified in <code>terms</code> are held constant (see 'Details'). See also arguments <code>condition</code> and <code>typical</code> .
ci.lvl	Numeric, the level of the confidence intervals. For <code>ggpredict()</code> , use <code>ci.lvl = NA</code> , 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 <code>predict()</code> function are returned, assuming normal distribution (i.e. $\pm 1.96 * SE$ ). See introduction of <a href="#">this vignette</a> for more details.
verbose	Toggle messages or warnings.
type	Character, only applies for survival models, mixed effects models and/or models with zero-inflation. <b>Note:</b> For <code>brmsfit</code> -models with zero-inflation component, there is no <code>type = "zero_inflated"</code> nor <code>type = "zi_random"</code> ; predicted values for <code>MixMod</code> -models from <b>GLMMadaptive</b> with zero-inflation component <i>always</i> condition on the zero-inflation part of the model (see 'Details'). <ul style="list-style-type: none"> <li>"fixed" (or "fe" or "count") <p>Predicted values are conditioned on the fixed effects or conditional model only (for mixed models: predicted values are on the population-level and <i>confidence intervals</i> are returned). For instance, for models fitted with <code>zeroinfl</code> from <b>pscl</b>, this would return the predicted mean from the count</p> </li> </ul>

component (without zero-inflation). For models with zero-inflation component, this type calls `predict(..., type = "link")` (however, predicted values are back-transformed to the response scale).

- "random" (or "re")  
This only applies to mixed models, and `type = "random"` does not condition on the zero-inflation component of the model. `type = "random"` still returns population-level predictions, however, unlike `type = "fixed"`, intervals also consider the uncertainty in the variance parameters (the mean random effect variance, see *Johnson et al. 2014* for details) and hence can be considered as *prediction intervals*. For models with zero-inflation component, this type calls `predict(..., type = "link")` (however, predicted values are back-transformed to the response scale).  
To get predicted values for each level of the random effects groups, add the name of the related random effect term to the `terms`-argument (for more details, see [this vignette](#)).
- "zero\_inflated" (or "fe.zi" or "zi")  
Predicted values are conditioned on the fixed effects and the zero-inflation component. For instance, for models fitted with `zeroinfl` from **pscl**, this would return the predicted response ( $\mu \times (1-p)$ ) and for **glmmTMB**, this would return the expected value  $\mu \times (1-p)$  *without* conditioning on random effects (i.e. random effect variances are not taken into account for the confidence intervals). For models with zero-inflation component, this type calls `predict(..., type = "response")`. See 'Details'.
- "zi\_random" (or "re.zi" or "zero\_inflated\_random")  
Predicted values are conditioned on the zero-inflation component and take the random effects uncertainty into account. For models fitted with `glmmTMB()`, `hurdle()` or `zeroinfl()`, this would return the expected value  $\mu \times (1-p)$ . For **glmmTMB**, prediction intervals also consider the uncertainty in the random effects variances. This type calls `predict(..., type = "response")`. See 'Details'.
- "zi\_prob" (or "zi.prob")  
Predicted zero-inflation probability. For **glmmTMB** models with zero-inflation component, this type calls `predict(..., type = "zlink")`; models from **pscl** call `predict(..., type = "zero")` and for **GLMMadaptive**, `predict(..., type = "zero_part")` is called.
- "simulate" (or "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 objects of class `lm`, `glm`, `glmmTMB`, `wbm`, `MixMod` and `merMod`. See ... for details on number of simulations.
- "survival" and "cumulative\_hazard" (or "surv" and "cumhaz")  
Applies only to `coxph`-objects from the **survival**-package and calculates the survival probability or the cumulative hazard of an event.

typical

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

condition	Named character vector, which indicates covariates that should be held constant at specific values. Unlike <code>typical</code> , which applies a function to the covariates to determine the value that is used to hold these covariates constant, <code>condition</code> can be used to define exact values, for instance <code>condition = c(covariate1 = 20, covariate2 = 5)</code> . See 'Examples'.
back.transform	Logical, if TRUE (the default), predicted values for log- or log-log transformed responses will be back-transformed to original response-scale.
interval	Type of interval calculation, can either be "confidence" (default) or "prediction". May be abbreviated. Unlike <i>confidence intervals</i> , <i>prediction intervals</i> include the residual variance ( $\sigma^2$ ). This argument is ignored for mixed models, as <code>interval = "prediction"</code> is equivalent to <code>type = "random"</code> (and <code>interval = "confidence"</code> is equivalent to <code>type = "fixed"</code> ). Note that prediction intervals are not available for all models, but only for models that work with <code>insight::get_sigma()</code> .
ppd	Logical, if TRUE, predictions for Stan-models are based on the posterior predictive distribution <code>rstantools::posterior_predict()</code> . If FALSE (the default), predictions are based on posterior draws of the linear predictor <code>rstantools::posterior_linpred()</code> .
vcov.fun	String, indicating the name of the <code>vcov*()</code> -function from the <b>sandwich</b> or <b>clubSandwich</b> -package, e.g. <code>vcov.fun = "vcovCL"</code> , 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 <code>predict()</code> -function. <b>Note</b> that probably not all model objects that work with <code>ggpredict()</code> are also supported by the <b>sandwich</b> or <b>clubSandwich</b> -package.
vcov.type	Character vector, specifying the estimation type for the robust covariance matrix estimation (see <code>?sandwich::vcovHC</code> or <code>?clubSandwich::vcovCR</code> for details).
vcov.args	List of named vectors, used as additional arguments that are passed down to <code>vcov.fun</code> .

## Details

### Supported Models

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

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

`ggpredict()` calls `predict()`, while `ggeffect()` calls `effects::Effect()` and `ggemmeans()` calls `emmeans::emmeans()` to compute 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

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

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

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

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

You can take a random sample of any size with `sample=n`, e.g. `terms = "income [sample=8]"`, which will sample eight values from all possible values of the variable `income`. This option is especially useful for plotting 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 [this vignette](#).

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

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

### Holding covariates at constant values

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

`ggeffect()` and `ggemmeans()`, by default, set remaining numeric covariates to their mean value, while for factors, a kind of "average" value, which represents the proportions of each factor's category, is used. The same applies to character vectors: `ggemmeans()` averages over the distribution of unique values in a character vector, similar to how factors are treated. 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**-packages. The predicted values are the median value of all drawn posterior samples. The confidence intervals for Stan-models are Bayesian predictive intervals. By default (i.e. `ppd = FALSE`), the predictions are based on

`rstantools::posterior_linpred()` and hence have some limitations: the uncertainty of the error term is not taken into account. The recommendation is to use the posterior predictive distribution (`rstantools::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 = "zero_inflated"` nor `type = "zi_random"` 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 `?MASS::mvrnorm`) are drawn to calculate  $\mu \times (1-p)$ . 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 `?sjmisc::typical_value`).

## 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.

**Note****Multinomial Models**

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

**Printing Results**

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

**Limitations**

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

**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.

**Examples**

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

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

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

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

# 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 ggplot by hand.
```

```

## Not run:
# 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)

# 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("c172code", "c161sex"))
ggplot(dat, aes(x, predicted, colour = group)) +
  geom_point(position = position_dodge(.1)) +
  geom_errorbar(
    aes(ymin = conf.low, ymax = conf.high),
    position = position_dodge(.1)
  ) +
  scale_x_discrete(breaks = 1:3, labels = get_x_labels(dat))

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

```

```

# make categorical
efc$c161sex <- as_factor(efc$c161sex)
fit <- lm(neg_c_7 ~ c12hour * barthtot * c161sex, data = efc)
# select only levels 30, 50 and 70 from continuous variable Barthel-Index
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
plot(dat, ci = FALSE)
## End(Not run)

# predictions 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")

```

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)
```

**Arguments**

<code>grid</code>	A data frame representing the data grid, or an object of class <code>ggeffects</code> , as returned by <code>ggpredict()</code> and others.
<code>model</code>	The model for which to compute partial residuals. The data grid <code>grid</code> should match to predictors in the model.

<code>collapse.by</code>	Name of the (random effects) grouping factor. Data is collapsed by the levels of this factor.
<code>residuals</code>	Logical, if TRUE, collapsed partial residuals instead of raw data by the levels of the grouping factor.

**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**

```
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**

```
# 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.

---

<code>get_title</code>	<i>Get titles and labels from data</i>
------------------------	--

---

### 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

<code>x</code>	An object of class <code>ggeffects</code> , as returned by any <code>ggeffects</code> -function; for <code>get_complete_df()</code> , must be a list of <code>ggeffects</code> -objects.
<code>case</code>	Desired target case. Labels will automatically converted into the specified character case. See <code>?sjlabelled::convert_case</code> 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)
  efc$c172code <- to_factor(efc$c172code)
  fit <- lm(barthtot ~ c12hour + neg_c_7 + c161sex + c172code, data = efc)
```

```

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

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.

### Usage

```

hypothesis_test(model, ...)

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

```

```
## S3 method for class 'ggeffects'
hypothesis_test(
  model,
  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 <code>ggeffects</code> .
...	Arguments passed down to <code>data_grid()</code> when creating the reference grid and to <code>marginaleffects::predictions()</code> resp. <code>marginaleffects::slopes()</code> .
terms	Character vector with the names of the focal terms from <code>model</code> , 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 <i>slopes</i> of this numeric predictor are computed (possibly grouped by the levels of further categorical focal predictors).
test	Hypothesis to test. By default, pairwise-comparisons are conducted. See section <i>Introduction into contrasts and pairwise comparisons</i> .
equivalence	ROPE's lower and higher bounds. Should be "default" or a vector of length two (e.g., <code>c(-0.1, 0.1)</code> ). If "default", <code>bayestestR::rope_range()</code> is used. Instead of using the equivalence argument, it is also possible to call the <code>equivalence_test()</code> method directly. This requires the <b>parameters</b> package to be loaded. When using <code>equivalence_test()</code> , two more columns with information about the ROPE coverage and decision on $H_0$ are added. Furthermore, it is possible to <code>plot()</code> the results from <code>equivalence_test()</code> . See <code>bayestestR::equivalence_test()</code> resp. <code>parameters::equivalence_test.lm()</code> for details.
p_adjust	Character vector, if not NULL, indicates the method to adjust p-values. See <code>stats::p.adjust()</code> for details. Further possible adjustment methods are "tukey" or "sidak". Some caution is necessary when adjusting p-value for multiple comparisons. See also section <i>P-value adjustment</i> 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 <code>insight::get_df()</code> with <code>type = "wald"</code> .
ci.lvl	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.

**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:

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

**Examples**

```
## Not run:
if (requireNamespace("marginaleffects") && interactive()) {
  data(efc)
  efc$c172code <- as.factor(efc$c172code)
  efc$c161sex <- as.factor(efc$c161sex)
  levels(efc$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"))

# 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")

# 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"))
}

## End(Not run)

```

---

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

```

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

```

## Arguments

source	Character. Either "development" or "cran". If "cran", <i>ggeffects</i> will be installed from the default CRAN mirror returned by <code>getOption("repos")['CRAN']</code> . If "development" (the default), <i>ggeffects</i> is installed from the r-universe repository ( <a href="https://strengjacker.r-universe.dev/">https://strengjacker.r-universe.dev/</a> ).
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

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

---

lung2	<i>Sample data set</i>
-------	------------------------

---

### Description

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	<i>Create a data frame from all combinations of predictor values</i>
----------	--

---

### Description

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

### Usage

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

```
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.
typical	Character vector, naming the function to be applied to the covariates over which the effect is "averaged". The default is "mean". See <code>?sjmisc::typical_value</code> 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'.

`...`            Currently not used.

### 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

```
## S3 method for class 'ggeffects'
plot(
  x,
  ci = TRUE,
  ci.style = c("ribbon", "errorbar", "dash", "dot"),
  facets,
  add.data = FALSE,
  limit.range = FALSE,
  residuals = FALSE,
  residuals.line = FALSE,
  collapse.group = FALSE,
  colors = NULL,
  alpha = 0.15,
```

```

dodge = 0.25,
use.theme = TRUE,
dot.alpha = 0.35,
jitter = 0.2,
log.y = FALSE,
case = NULL,
show.legend = TRUE,
show.title = TRUE,
show.x.title = TRUE,
show.y.title = TRUE,
dot.size = NULL,
line.size = NULL,
connect.lines = FALSE,
grid,
one.plot = TRUE,
rawdata,
...
)

theme_ggeffects(base_size = 11, base_family = "")

show_pals()

```

### Arguments

<code>x</code>	An object of class <code>ggeffects</code> , as returned by the functions from this package.
<code>ci</code>	Logical, if TRUE, confidence bands (for continuous variables at x-axis) resp. error bars (for factors at x-axis) are plotted.
<code>ci.style</code>	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.
<code>facets, grid</code>	Logical, defaults to TRUE, if <code>x</code> has a column named <code>facet</code> , and defaults to FALSE, if <code>x</code> has no such column. Set <code>facets = TRUE</code> to wrap the plot into facets even for grouping variables (see 'Examples'). <code>grid</code> is an alias for <code>facets</code> .
<code>add.data, rawdata</code>	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.
<code>limit.range</code>	Logical, if TRUE, limits the range of the prediction bands to the range of the data.
<code>residuals</code>	Logical, if TRUE, a layer with partial residuals is added to the plot. See vignette <a href="#">Effect Displays with Partial Residuals</a> . from <b>effects</b> for more details on partial residual plots.
<code>residuals.line</code>	Logical, if TRUE, a loess-fit line is added to the partial residuals plot. Only applies if <code>residuals</code> is TRUE.
<code>collapse.group</code>	For mixed effects models, name of the grouping variable of random effects. If <code>collapse.group = TRUE</code> , 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.

colors	<p>Character vector with color values in hex-format, valid color value names (see <code>demo("colors")</code>) or a name of a <code>ggeffects-color-palette</code>.</p> <p>Following options are valid for colors:</p> <ul style="list-style-type: none"> <li>• If not specified, the color brewer palette "Set1" will be used.</li> <li>• If "gs", a greyscale will be used.</li> <li>• If "bw", the plot is black/white and uses different line types to distinguish groups.</li> <li>• There are some pre-defined color-palettes in this package that can be used, e.g. <code>colors = "metro"</code>. See <code>show_pals()</code> to show all available palettes.</li> <li>• Else specify own color values or names as vector (e.g. <code>colors = c("#f00000", "#00ff00")</code>).</li> </ul>
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 <code>ci.style = "errorbars"</code> .
use.theme	Logical, if TRUE, a slightly tweaked version of <code>ggplot's</code> minimal-theme, <code>theme_ggeffects()</code> , is applied to the plot. If FALSE, no theme-modifications are applied.
dot.alpha	Alpha value for data points, when <code>add.data = TRUE</code> .
jitter	Numeric, between 0 and 1. If not NULL and <code>add.data = TRUE</code> , 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.
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 <code>?sjlabelled::convert_case</code> for more details on this argument.
show.legend	Logical, shows or hides the plot legend.
show.title	Logical, shows or hides the plot title-
show.x.title	Logical, shows or hides the plot title for the x-axis.
show.y.title	Logical, shows or hides the plot title for the y-axis.
dot.size	Numeric, size of the point geoms.
line.size	Numeric, size of the line geoms.
connect.lines	Logical, if TRUE and plot has point-geoms with error bars (this is usually the case when the x-axis is discrete), points of same groups will be connected with a line.
one.plot	Logical, if TRUE and <code>x</code> has a panel column (i.e. when four terms were used), a single, integrated plot is produced.

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

`base_size` Base font size.

`base_family` Base font family.

### Details

For proportional odds logistic regression (see `?MASS::polr`) or cumulative link models in general, plots are automatically faceted 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** (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 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

```
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)

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

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

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

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

# show all color palettes
show_pals()
```

---

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

<code>x</code>	A list of <code>ggeffects</code> objects, as returned by <code>ggpredict()</code> , <code>ggemmeans()</code> or <code>ggeffect()</code> .
<code>...</code>	Currently not used.

## Details

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

## Value

A data frame with pooled predictions.

## References

Rubin, D.B. (1987). *Multiple Imputation for Nonresponse in Surveys*. New York: John Wiley and Sons.

**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")
  })
  pool_predictions(predictions)
}
```

---

```
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
range_n_5 <- pretty_range(n = 5)
range_n_5(1:1000)

```

---

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, pred_name, ...)

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

```

### Arguments

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

### 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 partialled out.

## References

Fox J, Weisberg S. Visualizing Fit and Lack of Fit in Complex Regression Models with Predictor Effect Plots and Partial Residuals. *Journal of Statistical Software* 2018;87.

## Examples

```
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

*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

```
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

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

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

---

vcov

---

*Calculate variance-covariance matrix for marginal effects*


---

### Description

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

**Usage**

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

**Arguments**

<code>object</code>	An object of class "ggeffects", as returned by <code>ggpredict()</code> .
<code>vcov.fun</code>	String, indicating the name of the <code>vcov*()</code> -function from the <b>sandwich</b> or <b>clubSandwich</b> -package, e.g. <code>vcov.fun = "vcovCL"</code> , which is used to compute (cluster) robust standard errors for predictions. If <code>NULL</code> , standard errors (and confidence intervals) for predictions are based on the standard errors as returned by the <code>predict()</code> -function. <b>Note</b> that probably not all model objects that work with <code>ggpredict()</code> are also supported by the <b>sandwich</b> or <b>clubSandwich</b> -package.
<code>vcov.type</code>	Character vector, specifying the estimation type for the robust covariance matrix estimation (see <code>?sandwich::vcovHC</code> or <code>?clubSandwich::vcovCR</code> for details).
<code>vcov.args</code>	List of named vectors, used as additional arguments that are passed down to <code>vcov.fun</code> .
<code>...</code>	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  $12*12$  dimension. In short, `nrow(object)` is always equal to `nrow(vcov(object))`. See also 'Examples'.

**Value**

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

**Examples**

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

vcov(result)

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

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

# 2 levels for c161sex multiplied by 3 levels for c172code
```

```
# result in 6 combinations of predicted values
# thus vcov() returns a 6x6 matrix
result <- ggpredict(model, c("c161sex", "c172code"))
vcov(result)
```

# Index

- \* **data**
  - efc, 12
  - fish, 12
  - lung2, 18
- as.data.frame.ggeffects, 2
- bayestestR::equivalence\_test(), 15
- bayestestR::rope\_range(), 15
- collapse\_by\_group, 11
- collapse\_by\_group(), 21
- data.frame, 4
- data\_grid(new\_data), 18
- data\_grid(), 15
- efc, 12
- efc\_test(efc), 12
- effects::Effect(), 2
- emmeans::emmeans(), 2
- fish, 12
- get\_complete\_df(get\_title), 13
- get\_legend\_labels(get\_title), 13
- get\_legend\_title(get\_title), 13
- get\_title, 13
- get\_x\_labels(get\_title), 13
- get\_x\_title(get\_title), 13
- get\_y\_title(get\_title), 13
- ggeffect(as.data.frame.ggeffects), 2
- ggenmeans(as.data.frame.ggeffects), 2
- ggpredict(as.data.frame.ggeffects), 2
- ggpredict(), 23
- hypothesis\_test, 14
- insight::get\_df(), 15
- insight::get\_sigma(), 6
- install\_latest, 17
- lung2, 18
- make.names, 4
- marginaleffects::predictions(), 15
- marginaleffects::slopes(), 15
- mice::pool(), 23
- new\_data, 18
- new\_data(), 25
- parameters::equivalence\_test.lm(), 15, 16
- plot, 19
- pool\_predictions, 23
- predict(), 2
- pretty\_range, 24
- pretty\_range(), 7
- representative\_values(values\_at), 26
- residualize\_over\_grid, 25
- rstantools::posterior\_linpred(), 6, 8
- rstantools::posterior\_predict(), 6, 8
- show\_pals(plot), 19
- show\_pals(), 21, 22
- stats::p.adjust(), 15
- theme\_ggeffects(plot), 19
- values\_at, 26
- values\_at(), 7
- vcov, 27