Package 'marginaleffects'

November 23, 2022

Title Marginal Effects, Marginal Means, Predictions, and Contrasts

Version 0.8.1

Description Compute and plot adjusted predictions, contrasts, marginal effects, and marginal means for over 70 classes of statistical models in R. Conduct linear and non-linear hypothesis tests using the delta method.

License GPL (>= 3)

Copyright inst/COPYRIGHTS

Encoding UTF-8

URL https://vincentarelbundock.github.io/marginaleffects/

BugReports https://github.com/vincentarelbundock/marginaleffects/issues

RoxygenNote 7.2.1

VignetteBuilder knitr

Depends R (>= 3.5.0)

Imports checkmate, data.table, generics, insight (>= 0.18.5), methods

Suggests AER, afex, aod, bench, betareg, BH, bife, biglm, brglm2, brms, brmsmargins, broom, collapse, conflicted, covr, crch, datawizard, dplyr, emmeans, estimatr, fixest (>= 0.10.1), future, future.apply, gam, gamlss, gamlss.dist, geepack, ggbeeswarm, ggdag, ggdist, ggplot2, ggrepel, glmmTMB, glmx, haven, here, itsadug, ivreg, kableExtra, knitr, lme4, lmerTest, magrittr, margins, MASS, MatchIt, mclogit, mgcv, mhurdle, mice, mlogit, modelbased, modelsummary, nlme, nnet, ordinal, patchwork, pkgdown, plm, polspline, posterior, prediction, pscl, quantreg, RcppEigen, remotes, rlang, rmarkdown, rms, robust, robustbase, robustlmm, rstanarm, rstantools, sampleSelection, sandwich, scam, speedglm, spelling, survey, survival, systemfonts, tidymodels, tidyr, tidyverse, tinytest, titanic, truncreg, tsModel, vdiffr, withr

Collate 'align_J_V.R' 'arg_name_change.R' 'attributes.R' 'backtransform.R' 'by.R' 'comparisons.R' 'complete_levels.R' 'datagrid.R' 'deltamethod.R' 'deprecate_arg.R'

'find_categorical.R' 'find_variable_class.R' 'format_msg.R'
'get_ci.R' 'get_coef.R' 'get_contrast_data.R'
'get_contrast_data_character.R' 'get_contrast_data_factor.R'
'get_contrast_data_logical.R' 'get_contrast_data_numeric.R'
'get_contrasts.R' 'get_eti.R' 'get_group_names.R' 'get_hdi.R'
'get_hypothesis.R' 'get_jacobian.R' 'get_predict.R'
'get_se_delta.R' 'get_vcov.R' 'github_issue.R' 'glance.R'
'hush.R' 'marginaleffects.R' 'marginalmeans.R' 'mean_or_mode.R'
'set_coef.R' 'methods_MASS.R' 'methods_afex.R' 'methods_aod.R'
'methods_betareg.R' 'methods_bife.R' 'methods_biglm.R'
'methods_nnet.R' 'methods_brglm2.R' 'sanity_model.R'
'methods_brms.R' 'methods_crch.R' 'methods_fixest.R'
'methods_gamlss.R' 'methods_glmmTMB.R' 'methods_glmx.R'
'methods_lme4.R' 'methods_mclogit.R' 'methods_mgcv.R'
'methods_mhurdle.R' 'methods_mlm.R' 'methods_mlogit.R'
'methods_ordinal.R' 'methods_plm.R' 'methods_pscl.R'
'methods_quantreg.R' 'methods_rms.R' 'methods_robustlmm.R'
'methods_rstanarm.R' 'methods_sampleSelection.R'
'methods_scam.R' 'methods_stats.R' 'methods_survival.R'
'methods_tobit1.R' 'modelarchive.R' 'myTryCatch.R' 'package.R'
'plot.R' 'plot_cap.R' 'plot_cco.R' 'plot_cme.R'
'posteriordraws.R' 'predictions.R' 'sanitize_conf_level.R'
'sanitize_hypothesis.R' 'sanitize_interaction.R'
'sanitize_newdata.R' 'sanitize_transform_pre.R'
'sanitize_type.R' 'sanitize_variables.R' 'sanity.R'
'sanity_by.R' 'sanity_dots.R' 'settings.R' 'summary.R' 'tidy.R'
'tidy.comparisons.R' 'tidy.predictions.R' 'type_dictionary.R'
'unpack_matrix_cols.R' 'utils.R' 'utils_cjdt.R' 'warn_once.R'

Language en-US

NeedsCompilation no

Author Vincent Arel-Bundock [aut, cre, cph]
(<https://orcid.org/0000-0003-2042-7063>),
Marcio Augusto Diniz [ctb] (<https://orcid.org/0000-0002-2427-7843>),
Noah Greifer [ctb] (<https://orcid.org/0000-0003-3067-7154>)

Maintainer Vincent Arel-Bundock <vincent.arel-bundock@umontreal.ca>

Repository CRAN

Date/Publication 2022-11-23 22:20:06 UTC

R topics documented:

`comparisons` *Contrasts Between Adjusted Predictions*

Description

Difference, ratio, or function of adjusted predictions, calculated for meaningfully different predictor values. The `tidy()` and `summary()` functions can be used to aggregate and summarize the output of `comparisons()`. To learn more, read the contrasts vignette, visit the package website, or scroll down this page for a full list of vignettes:

- https://vincentarelbundock.github.io/marginaleffects/articles/contrasts.html
- https://vincentarelbundock.github.io/marginaleffects/

Usage

```
comparisons(
  model,
  newdata = NULL,
  variables = NULL,
  type = NULL,
  vcov = TRUE,
  conf_level = 0.95,
  transform_pre = "difference",
  transform_post = NULL,
  cross = FALSE,
  by = NULL,
  wts = NULL,
  hypothesis = NULL,
  eps = NULL,
```

```
  ...
)
```

Arguments

`model`	Model object
`newdata`	`NULL`, data frame, string, or `datagrid()` call. Determines the predictor values for which to compute contrasts. • `NULL` (default): Unit-level contrasts for each observed value in the original dataset. • data frame: Unit-level contrasts for each row of the `newdata` data frame. • string: – "mean": Contrasts at the Mean. Contrasts when each predictor is held at its mean or mode. – "median": Contrasts at the Median. Contrasts when each predictor is held at its median or mode. – "marginalmeans": Contrasts at Marginal Means. – "tukey": Contrasts at Tukey's 5 numbers. – "grid": Contrasts on a grid of representative numbers (Tukey's 5 numbers and unique values of categorical predictors). • `datagrid()` call to specify a custom grid of regressors. For example: – `newdata = datagrid(cyl = c(4, 6))`: `cyl` variable equal to 4 and 6 and other regressors fixed at their means or modes. – `newdata = datagrid(mpg = fivenum)`: `mpg` variable held at Tukey's five numbers (using the `fivenum` function), and other regressors fixed at their means or modes. – See the Examples section and the datagrid documentation.
`variables`	`NULL`, character vector, or named list. The subset of variables for which to compute contrasts. • `NULL`: compute contrasts for all the variables in the model object (can be slow). • Character vector: subset of variables (usually faster). • Named list: names identify the subset of variables of interest, and values define the type of contrast to compute. Acceptable values depend on the variable type: – Factor or character variables: ∗ "reference": Each factor level is compared to the factor reference (base) level ∗ "all": All combinations of observed levels ∗ "sequential": Each factor level is compared to the previous factor level ∗ "pairwise": Each factor level is compared to all other levels ∗ Vector of length 2 with the two values to compare. – Logical variables:

    - NULL: contrast between TRUE and FALSE
  - Numeric variables:
    - Numeric of length 1: Contrast for a gap of `x`, computed at the observed value plus and minus `x / 2`. For example, estimating a `+1` contrast compares adjusted predictions when the regressor is equal to its observed value minus 0.5 and its observed value plus 0.5.
    - Numeric vector of length 2: Contrast between the 2nd element and the 1st element of the `x` vector.
    - Function which accepts a numeric vector and returns a data frame with two columns of "low" and "high" values to compare. See examples below.
    - "iqr": Contrast across the interquartile range of the regressor.
    - "sd": Contrast across one standard deviation around the regressor mean.
    - "2sd": Contrast across two standard deviations around the regressor mean.
    - "minmax": Contrast between the maximum and the minimum values of the regressor.
  - Examples:
    - `variables = list(gear = "pairwise", hp = 10)`
    - `variables = list(gear = "sequential", hp = c(100, 120))`
    - See the Examples section below for more.

`type` string indicates the type (scale) of the predictions used to compute marginal effects or contrasts. This can differ based on the model type, but will typically be a string such as: "response", "link", "probs", or "zero". When an unsupported string is entered, the model-specific list of acceptable values is returned in an error message. When `type` is `NULL`, the default value is used. This default is the first model-related row in the `marginaleffects:::type_dictionary` dataframe.

`vcov` Type of uncertainty estimates to report (e.g., for robust standard errors). Acceptable values:

- FALSE: Do not compute standard errors. This can speed up computation considerably.
- TRUE: Unit-level standard errors using the default `vcov(model)` variance-covariance matrix.
- String which indicates the kind of uncertainty estimates to return.
  - Heteroskedasticity-consistent: `"HC"`, `"HC0"`, `"HC1"`, `"HC2"`, `"HC3"`, `"HC4"`, `"HC4m"`, `"HC5"`. See `?sandwich::vcovHC`
  - Heteroskedasticity and autocorrelation consistent: `"HAC"`
  - Mixed-Models degrees of freedom: "satterthwaite", "kenward-roger"
  - Other: `"NeweyWest"`, `"KernHAC"`, `"OPG"`. See the `sandwich` package documentation.
- One-sided formula which indicates the name of cluster variables (e.g., `~unit_id`). This formula is passed to the `cluster` argument of the `sandwich::vcovCL` function.

- Square covariance matrix
- Function which returns a covariance matrix (e.g., `stats::vcov(model)`)

`conf_level` numeric value between 0 and 1. Confidence level to use to build a confidence interval.

`transform_pre` string or function. How should pairs of adjusted predictions be contrasted?

- string: shortcuts to common contrast functions.
  - Supported shortcuts strings: difference, differenceavg, differenceavgwts, dydx, eyex, eydx, dyex, dydxavg, eyexavg, eydxavg, dyexavg, dydxavgwts, eyexavgwts, eydxavgwts, dyexavgwts, ratio, ratioavg, ratioavgwts, lnratio, lnratioavg, lnratioavgwts, lnor, lnoravg, lnoravgwts, expdydx, expdydxavg, expdydxavgwts
  - See the Transformations section below for definitions of each transformation.
- function: accept two equal-length numeric vectors of adjusted predictions (`hi` and `lo`) and returns a vector of contrasts of the same length, or a unique numeric value.
  - See the Transformations section below for examples of valid functions.

`transform_post` string or function. Transformation applied to unit-level estimates and confidence intervals just before the function returns results. Functions must accept a vector and return a vector of the same length. Support string shortcuts: "exp", "ln"

`cross` TRUE or FALSE

- `FALSE`: Contrasts represent the change in adjusted predictions when one predictor changes and all other variables are held constant.
- `TRUE`: Contrasts represent the changes in adjusted predictions when all the predictors specified in the `variables` argument are manipulated simultaneously (a "cross-contrast").

`by` Compute group-wise average estimates. Valid inputs:

- Character vector of column names in `newdata` or in the data frame produced by calling the function without the `by` argument.
- Data frame with a `by` column of group labels, and merging columns shared by `newdata` or the data frame produced by calling the same function without the `by` argument.
- See examples below.

`wts` string or numeric: weights to use when computing average contrasts or marginal-effects. These weights only affect the averaging in `tidy()` or `summary()`, and not the unit-level estimates themselves.

- string: column name of the weights variable in `newdata`. When supplying a column name to `wts`, it is recommended to supply the original data (including the weights variable) explicitly to `newdata`.
- numeric: vector of length equal to the number of rows in the original data or in `newdata` (if supplied).

`hypothesis` specify a hypothesis test or custom contrast using a vector, matrix, string, or string formula.

- String:

  - "pairwise": pairwise differences between estimates in each row.
  - "reference": differences between the estimates in each row and the estimate in the first row.
  - "sequential": difference between an estimate and the estimate in the next row.
  - "revpairwise", "revreference", "revsequential": inverse of the corresponding hypotheses, as described above.
- String formula to specify linear or non-linear hypothesis tests. If the `term` column uniquely identifies rows, terms can be used in the formula. Otherwise, use `b1`, `b2`, etc. to identify the position of each parameter. Examples:
  - `hp = drat`
  - `hp + drat = 12`
  - `b1 + b2 + b3 = 0`
- Numeric vector: Weights to compute a linear combination of (custom contrast between) estimates. Length equal to the number of rows generated by the same function call, but without the `hypothesis` argument.
- Numeric matrix: Each column is a vector of weights, as describe above, used to compute a distinct linear combination of (contrast between) estimates. The column names of the matrix are used as labels in the output.
- See the Examples section below and the vignette: https://vincentarelbundock.github.io/marginaleffect

`eps` NULL or numeric value which determines the step size to use when calculating numerical derivatives: (f(x+eps)-f(x))/eps. When `eps` is `NULL`, the step size is 0.0001 multiplied by the difference between the maximum and minimum values of the variable with respect to which we are taking the derivative. Changing `eps` may be necessary to avoid numerical problems in certain models.

`...` Additional arguments are passed to the `predict()` method supplied by the modeling package.These arguments are particularly useful for mixed-effects or bayesian models (see the online vignettes on the `marginaleffects` website). Available arguments can vary from model to model, depending on the range of supported arguments by each modeling package. See the "Model-Specific Arguments" section of the `?marginaleffects` documentation for a non-exhaustive list of available arguments.

Details

A "contrast" is a difference, ratio of function of adjusted predictions, calculated for meaningfully different predictor values (e.g., College graduates vs. Others). Uncertainty estimates are computed using the delta method.

The `newdata` argument can be used to control the kind of contrasts to report:

- Average Contrasts
- Adjusted Risk Ratios
- Adjusted Risk Differences
- Group-Average Contrasts
- Contrasts at the Mean
- Contrasts at User-Specified values (aka Contrasts at Representative values, MER).
- Custom contrasts using arbitrary functions

Vignettes and documentation

Vignettes:

- Adjusted Predictions
- Contrasts
- Marginal Effects
- Marginal Means
- Hypothesis Tests and Custom Contrasts using the Delta Method

Case studies:

- Bayesian Analyses with `brms`
- Causal Inference with the g-Formula
- Elasticity
- Experiments
- Generalized Additive Models
- Mixed effects models
- Multinomial Logit and Discrete Choice Models
- Multiple Imputation
- Plots: interactions, predictions, contrasts, and slopes
- Python NumPyro models in `marginaleffects`
- Unit-level contrasts in logistic regressions

Tips and technical notes:

- 71 Supported Classes of Models
- Index of Functions and Documentation
- Extending `marginaleffects`: add new models or modify existing ones
- Standard Errors and Confidence Intervals
- Tables and Plots
- Performance
- Alternative Software
- Frequently Asked Questions

Model-Specific Arguments

Some model types allow model-specific arguments to modify the nature of marginal effects, predictions, marginal means, and contrasts.

Package	Class	Argument	Documentation
`brms`	`brmsfit`	`ndraws`	brms::posterior_predict
		`re_formula`	
`lme4`	`merMod`	`include_random`	insight::get_predicted
		`re.form`	lme4::predict.merMod

		allow.new.levels	lme4::predict.merMod
glmmTMB	glmmTMB	re.form	glmmTMB::predict.glmmTMB
		allow.new.levels	glmmTMB::predict.glmmTMB
		zitype	glmmTMB::predict.glmmTMB
mgcv	bam	exclude	mgcv::predict.bam
robustlmm	rlmerMod	re.form	robustlmm::predict.rlmerMod
		allow.new.levels	robustlmm::predict.rlmerMod

Transformations

The following transformations can be applied by supplying one of the shortcut strings to the `transform_pre` argument. `hi` is a vector of adjusted predictions for the "high" side of the contrast. `lo` is a vector of adjusted predictions for the "low" side of the contrast. `y` is a vector of adjusted predictions for the original data. `x` is the predictor in the original data. `eps` is the step size to use to compute derivatives and elasticities.

Shortcut	Function
difference	\(hi, lo) hi - lo
differenceavg	\(hi, lo) mean(hi) - mean(lo)
differenceavgwts	\(hi, lo, w) wmean(hi, w) - wmean(lo, w)
dydx	\(hi, lo, eps) (hi - lo)/eps
eyex	\(hi, lo, eps, y, x) (hi - lo)/eps * (x/y)
eydx	\(hi, lo, eps, y, x) ((hi - lo)/eps)/y
dyex	\(hi, lo, eps, x) ((hi - lo)/eps) * x
dydxavg	\(hi, lo, eps) mean((hi - lo)/eps)
eyexavg	\(hi, lo, eps, y, x) mean((hi - lo)/eps * (x/y))
eydxavg	\(hi, lo, eps, y, x) mean(((hi - lo)/eps)/y)
dyexavg	\(hi, lo, eps, x) mean(((hi - lo)/eps) * x)
dydxavgwts	\(hi, lo, eps, w) wmean((hi - lo)/eps, w)
eyexavgwts	\(hi, lo, eps, y, x, w) wmean((hi - lo)/eps * (x/y), w)
eydxavgwts	\(hi, lo, eps, y, x, w) wmean(((hi - lo)/eps)/y, w)
dyexavgwts	\(hi, lo, eps, x, w) wmean(((hi - lo)/eps) * x, w)
ratio	\(hi, lo) hi/lo
ratioavg	\(hi, lo) mean(hi)/mean(lo)
ratioavgwts	\(hi, lo) wmean(hi)/wmean(lo)
lnratio	\(hi, lo) log(hi/lo)
lnratioavg	\(hi, lo) log(mean(hi)/mean(lo))
lnratioavgwts	\(hi, lo) log(wmean(hi)/wmean(lo))
lnor	\(hi, lo) log((hi/(1 - hi))/(lo/(1 - lo)))
lnoravg	\(hi, lo) log((mean(hi)/(1 - mean(hi)))/(mean(lo)/(1 - mean(lo))))
lnoravgwts	\(hi, lo, w) log((wmean(hi, w)/(1 - wmean(hi, w)))/(wmean(lo, w)/(1 - wmean(lo, w))))
expdydx	\(hi, lo, eps) ((exp(hi) - exp(lo))/exp(eps))/eps
expdydxavg	\(hi, lo, eps) mean(((exp(hi) - exp(lo))/exp(eps))/eps)
expdydxavgwts	\(hi, lo, eps, w) wmean(((exp(hi) - exp(lo))/exp(eps))/eps, w)

Examples

```
library(marginaleffects)
library(magrittr)

# Linear model
tmp <- mtcars
tmp$am <- as.logical(tmp$am)
mod <- lm(mpg ~ am + factor(cyl), tmp)
comparisons(mod, variables = list(cyl = "reference")) %>% tidy()
comparisons(mod, variables = list(cyl = "sequential")) %>% tidy()
comparisons(mod, variables = list(cyl = "pairwise")) %>% tidy()

# GLM with different scale types
mod <- glm(am ~ factor(gear), data = mtcars)
comparisons(mod, type = "response") %>% tidy()
comparisons(mod, type = "link") %>% tidy()

# Contrasts at the mean
comparisons(mod, newdata = "mean")

# Contrasts between marginal means
comparisons(mod, newdata = "marginalmeans")

# Contrasts at user-specified values
comparisons(mod, newdata = datagrid(am = 0, gear = tmp$gear))
comparisons(mod, newdata = datagrid(am = unique, gear = max))

m <- lm(mpg ~ hp + drat + factor(cyl) + factor(am), data = mtcars)
comparisons(m, variables = "hp", newdata = datagrid(FUN_factor = unique, FUN_numeric = median))


# Numeric contrasts
mod <- lm(mpg ~ hp, data = mtcars)
comparisons(mod, variables = list(hp = 1)) %>% tidy()
comparisons(mod, variables = list(hp = 5)) %>% tidy()
comparisons(mod, variables = list(hp = c(90, 100))) %>% tidy()
comparisons(mod, variables = list(hp = "iqr")) %>% tidy()
comparisons(mod, variables = list(hp = "sd")) %>% tidy()
comparisons(mod, variables = list(hp = "minmax")) %>% tidy()

# using a function to specify a custom difference in one regressor
dat <- mtcars
dat$new_hp <- 49 * (dat$hp - min(dat$hp)) / (max(dat$hp) - min(dat$hp)) + 1
modlog <- lm(mpg ~ log(new_hp) + factor(cyl), data = dat)
fdiff <- \(x) data.frame(x, x + 10)
comparisons(modlog, variables = list(new_hp = fdiff)) %>% summary()

# Adjusted Risk Ratio: see the contrasts vignette
mod <- glm(vs ~ mpg, data = mtcars, family = binomial)
cmp <- comparisons(mod, transform_pre = "lnratioavg")
summary(cmp, transform_avg = exp)
```

```
# Adjusted Risk Ratio: Manual specification of the `transform_pre`
cmp <- comparisons(mod, transform_pre = function(hi, lo) log(mean(hi) / mean(lo)))
summary(cmp, transform_avg = exp)
# cross contrasts
mod <- lm(mpg ~ factor(cyl) * factor(gear) + hp, data = mtcars)
cmp <- comparisons(mod, variables = c("cyl", "gear"), cross = TRUE)
summary(cmp)

# variable-specific contrasts
cmp <- comparisons(mod, variables = list(gear = "sequential", hp = 10))
summary(cmp)

# hypothesis test: is the `hp` marginal effect at the mean equal to the `drat` marginal effect
mod <- lm(mpg ~ wt + drat, data = mtcars)

comparisons(
    mod,
    newdata = "mean",
    hypothesis = "wt = drat")

# same hypothesis test using row indices
comparisons(
    mod,
    newdata = "mean",
    hypothesis = "b1 - b2 = 0")

# same hypothesis test using numeric vector of weights
comparisons(
    mod,
    newdata = "mean",
    hypothesis = c(1, -1))

# two custom contrasts using a matrix of weights
lc <- matrix(c(
    1, -1,
    2, 3),
    ncol = 2)
comparisons(
    mod,
    newdata = "mean",
    hypothesis = lc)


# `by` argument
mod <- lm(mpg ~ hp * am * vs, data = mtcars)
cmp <- comparisons(mod, variables = "hp", by = c("vs", "am"))
summary(cmp)

library(nnet)
mod <- multinom(factor(gear) ~ mpg + am * vs, data = mtcars, trace = FALSE)
by <- data.frame(
    group = c("3", "4", "5"),
```

```
    by = c("3,4", "3,4", "5"))
comparisons(mod, type = "probs", by = by)
```

`datagrid` *Generate a data grid of "typical," "counterfactual," or user-specified values for use in the* `newdata` *argument of the* `marginaleffects` *or* `predictions` *functions.*

Description

Generate a data grid of "typical," "counterfactual," or user-specified values for use in the `newdata` argument of the `marginaleffects` or `predictions` functions.

Usage

```
datagrid(
  ...,
  model = NULL,
  newdata = NULL,
  grid_type = "typical",
  FUN_character = Mode,
  FUN_factor = Mode,
  FUN_logical = Mode,
  FUN_numeric = function(x) mean(x, na.rm = TRUE),
  FUN_integer = function(x) round(mean(x, na.rm = TRUE)),
  FUN_other = function(x) mean(x, na.rm = TRUE)
)
```

Arguments

`...`	named arguments with vectors of values or functions for user-specified variables. • Functions are applied to the variable in the `model` dataset or `newdata`, and must return a vector of the appropriate type. • Character vectors are automatically transformed to factors if necessary. +The output will include all combinations of these variables (see Examples below.)
`model`	Model object
`newdata`	data.frame (one and only one of the `model` and `newdata` arguments
`grid_type`	character • "typical": variables whose values are not explicitly specified by the user in `...` are set to their mean or mode, or to the output of the functions supplied to `FUN_type` arguments.

- "counterfactual": the entire dataset is duplicated for each combination of the variable values specified in `...`. Variables not explicitly supplied to `datagrid()` are set to their observed values in the original dataset.

`FUN_character`	the function to be applied to character variables.
`FUN_factor`	the function to be applied to factor variables.
`FUN_logical`	the function to be applied to factor variables.
`FUN_numeric`	the function to be applied to numeric variables.
`FUN_integer`	the function to be applied to integer variables.
`FUN_other`	the function to be applied to other variable types.

Details

If `datagrid` is used in a `marginaleffects` or `predictions` call as the `newdata` argument, the model is automatically inserted in the function call, and users do not need to specify either the `model` or `newdata` arguments. Note that only the variables used to fit the models will be attached to the results. If a user wants to attach other variables as well (e.g., weights or grouping variables), they can supply a data.frame explicitly to the `newdata` argument inside `datagrid()`.

If users supply a model, the data used to fit that model is retrieved using the `insight::get_data` function.

Value

A `data.frame` in which each row corresponds to one combination of the named predictors supplied by the user via the `...` dots. Variables which are not explicitly defined are held at their mean or mode.

See Also

Other grid: `datagridcf()`

Examples

```
# The output only has 2 rows, and all the variables except `hp` are at their
# mean or mode.
datagrid(newdata = mtcars, hp = c(100, 110))

# We get the same result by feeding a model instead of a data.frame
mod <- lm(mpg ~ hp, mtcars)
datagrid(model = mod, hp = c(100, 110))

# Use in `marginaleffects` to compute "Typical Marginal Effects". When used
# in `marginaleffects()` or `predictions()` we do not need to specify the
#`model` or `newdata` arguments.
marginaleffects(mod, newdata = datagrid(hp = c(100, 110)))

# datagrid accepts functions
datagrid(hp = range, cyl = unique, newdata = mtcars)
comparisons(mod, newdata = datagrid(hp = fivenum))
```

```
# The full dataset is duplicated with each observation given counterfactual
# values of 100 and 110 for the `hp` variable. The original `mtcars` includes
# 32 rows, so the resulting dataset includes 64 rows.
dg <- datagrid(newdata = mtcars, hp = c(100, 110), grid_type = "counterfactual")
nrow(dg)

# We get the same result by feeding a model instead of a data.frame
mod <- lm(mpg ~ hp, mtcars)
dg <- datagrid(model = mod, hp = c(100, 110), grid_type = "counterfactual")
nrow(dg)
```

`datagridcf` *A "counterfactual" version of the* `datagrid()` *function.*

Description

For each combination of the variable values specified, this function duplicates the entire data frame supplied to `newdata`, or the entire dataset used to fit `model`. This is a convenience shortcut to call the `datagrid()` function with argument `grid_type="counterfactual"`.

Usage

```
datagridcf(..., model = NULL, newdata = NULL)
```

Arguments

`...`	named arguments with vectors of values or functions for user-specified variables. • Functions are applied to the variable in the `model` dataset or `newdata`, and must return a vector of the appropriate type. • Character vectors are automatically transformed to factors if necessary. +The output will include all combinations of these variables (see Examples below.)
`model`	Model object
`newdata`	data.frame (one and only one of the `model` and `newdata` arguments

See Also

Other grid: `datagrid()`

Examples

```
# Fit a model with 32 observations from the `mtcars` dataset.
nrow(mtcars)

mod <- lm(mpg ~ hp + am, data = mtcars)

# We specify two values for the `am` variable and obtain a counterfactual
```

```
# dataset with 64 observations (32 x 2).
dat <- datagridcf(model = mod, am = 0:1)
head(dat)
nrow(dat)

# We specify 2 values for the `am` variable and 3 values for the `hp` variable
# and obtained a dataset with 192 observations (2x3x32), corresponding to the
# full original data, with each possible combination of `hp` and `am`.
dat <- datagridcf(am = 0:1, hp = c(100, 110, 120), newdata = mtcars)
head(dat)
dim(dat)
```

`deltamethod` *Estimate and Standard Error of a Non-Linear Function of Estimated Model Parameters*

Description

`deltamethod` is a function to get a first-order approximate standard error for a nonlinear function of a vector of random variables with known or estimated covariance matrix. `deltamethod` emulates the behavior of the excellent and well-established car::deltaMethod and car::linearHypothesis functions, but it supports more models, requires fewer dependencies, and offers some convenience features like shortcuts for robust standard errors.

Usage

```
deltamethod(
  model,
  hypothesis = NULL,
  FUN = NULL,
  vcov = NULL,
  conf_level = 0.95,
  ...
)
```

Arguments

`model`	Model object or object generated by the `comparisons()`, `marginaleffects()`, `predictions()`, or `marginalmeans()` functions.
`hypothesis`	specify a hypothesis test or custom contrast using a vector, matrix, string, or string formula.

- String:
  - "pairwise": pairwise differences between estimates in each row.
  - "reference": differences between the estimates in each row and the estimate in the first row.
  - "sequential": difference between an estimate and the estimate in the next row.

  - "revpairwise", "revreference", "revsequential": inverse of the corresponding hypotheses, as described above.
- String formula to specify linear or non-linear hypothesis tests. If the `term` column uniquely identifies rows, terms can be used in the formula. Otherwise, use `b1`, `b2`, etc. to identify the position of each parameter. Examples:
  - `hp = drat`
  - `hp + drat = 12`
  - `b1 + b2 + b3 = 0`
- Numeric vector: Weights to compute a linear combination of (custom contrast between) estimates. Length equal to the number of rows generated by the same function call, but without the `hypothesis` argument.
- Numeric matrix: Each column is a vector of weights, as describe above, used to compute a distinct linear combination of (contrast between) estimates. The column names of the matrix are used as labels in the output.
- See the Examples section below and the vignette: https://vincentarelbundock.github.io/marginaleffect

`FUN` `NULL` or function.

- `NULL` (default): hypothesis test on the model's coefficients.
- Function which accepts a model object and returns a numeric vector or a data.frame with two columns called `term` and `estimate`. This argument can be useful when users want to conduct a hypothesis test on an arbitrary function of quantities held in a model object.

`vcov` Type of uncertainty estimates to report (e.g., for robust standard errors). Acceptable values:

- FALSE: Do not compute standard errors. This can speed up computation considerably.
- TRUE: Unit-level standard errors using the default `vcov(model)` variance-covariance matrix.
- String which indicates the kind of uncertainty estimates to return.
  - Heteroskedasticity-consistent: `"HC"`, `"HC0"`, `"HC1"`, `"HC2"`, `"HC3"`, `"HC4"`, `"HC4m"`, `"HC5"`. See `?sandwich::vcovHC`
  - Heteroskedasticity and autocorrelation consistent: `"HAC"`
  - Mixed-Models degrees of freedom: "satterthwaite", "kenward-roger"
  - Other: `"NeweyWest"`, `"KernHAC"`, `"OPG"`. See the `sandwich` package documentation.
- One-sided formula which indicates the name of cluster variables (e.g., `~unit_id`). This formula is passed to the `cluster` argument of the `sandwich::vcovCL` function.
- Square covariance matrix
- Function which returns a covariance matrix (e.g., `stats::vcov(model)`)

`conf_level` numeric value between 0 and 1. Confidence level to use to build a confidence interval.

`...` Additional arguments are passed to the `predict()` method supplied by the modeling package.These arguments are particularly useful for mixed-effects or bayesian models (see the online vignettes on the `marginaleffects` website).

Available arguments can vary from model to model, depending on the range of supported arguments by each modeling package. See the "Model-Specific Arguments" section of the ?marginaleffects documentation for a non-exhaustive list of available arguments.

Details

Warning: For hypothesis tests on objects produced by the marginaleffects package, it is safer to use the hypothesis argument of the original function. Using deltamethod() may not work in certain environments, or when called programmatically.

Examples

```
library(marginaleffects)
mod <- lm(mpg ~ hp + wt + factor(cyl), data = mtcars)

# When `FUN` and `hypothesis` are `NULL`, `deltamethod()` returns a data.frame of parameters
deltamethod(mod)

# Test of equality between coefficients
deltamethod(mod, hypothesis = "hp = wt")

# Non-linear function
deltamethod(mod, hypothesis = "exp(hp + wt) = 0.1")

# Robust standard errors
deltamethod(mod, hypothesis = "hp = wt", vcov = "HC3")

# b1, b2, ... shortcuts can be used to identify the position of the
# parameters of interest in the output of FUN
deltamethod(mod, hypothesis = "b2 = b3")

# term names with special characters have to be enclosed in backticks
deltamethod(mod, hypothesis = "`factor(cyl)6` = `factor(cyl)8`")

mod2 <- lm(mpg ~ hp * drat, data = mtcars)
deltamethod(mod2, hypothesis = "`hp:drat` = drat")

# predictions(), comparisons(), and marginaleffects()
mod <- glm(am ~ hp + mpg, data = mtcars, family = binomial)
cmp <- comparisons(mod, newdata = "mean")
deltamethod(cmp, hypothesis = "b1 = b2")

mfx <- marginaleffects(mod, newdata = "mean")
deltamethod(cmp, hypothesis = "b2 = 0.2")

pre <- predictions(mod, newdata = datagrid(hp = 110, mpg = c(30, 35)))
deltamethod(pre, hypothesis = "b1 = b2")

# The `FUN` argument can be used to compute standard errors for fitted values
mod <- glm(am ~ hp + mpg, data = mtcars, family = binomial)
```

```
f <- function(x) predict(x, type = "link", newdata = mtcars)
p <- deltamethod(mod, FUN = f)
head(p)

f <- function(x) predict(x, type = "response", newdata = mtcars)
p <- deltamethod(mod, FUN = f)
head(p)
```

`glance.marginaleffects` *Glance at key characteristics of an object*

Description

Glance at key characteristics of an object

Usage

```
## S3 method for class 'marginaleffects'
glance(x, ...)
```

Arguments

`x`	An object produced by the `marginaleffects` function.
`...`	Additional arguments are passed to the `predict()` method supplied by the modeling package.These arguments are particularly useful for mixed-effects or bayesian models (see the online vignettes on the `marginaleffects` website). Available arguments can vary from model to model, depending on the range of supported arguments by each modeling package. See the "Model-Specific Arguments" section of the ?`marginaleffects` documentation for a non-exhaustive list of available arguments.

See Also

Other summary: `reexports`, `summary.comparisons()`, `summary.marginaleffects()`, `summary.marginalmeans()`, `summary.predictions()`, `tidy.comparisons()`, `tidy.deltamethod()`, `tidy.marginaleffects()`, `tidy.marginalmeans()`, `tidy.predictions()`

`marginaleffects` *Marginal Effects (Slopes)*

Description

Partial derivative (slope) of the regression equation with respect to a regressor of interest. The `tidy()` and `summary()` functions can be used to aggregate and summarize the output of `marginaleffects()`. To learn more, read the marginal effects vignette, visit the package website, or scroll down this page for a full list of vignettes:

- https://vincentarelbundock.github.io/marginaleffects/articles/marginaleffects.html
- https://vincentarelbundock.github.io/marginaleffects/

Usage

```
marginaleffects(
  model,
  newdata = NULL,
  variables = NULL,
  vcov = TRUE,
  conf_level = 0.95,
  type = NULL,
  slope = "dydx",
  by = NULL,
  wts = NULL,
  hypothesis = NULL,
  eps = NULL,
  ...
)
```

Arguments

`model`	Model object
`newdata`	`NULL`, data frame, string, or `datagrid()` call. Determines the predictor values for which to compute marginal effects. • `NULL` (default): Unit-level marginal effects for each observed value in the original dataset. • data frame: Unit-level marginal effects for each row of the `newdata` data frame. • string: – "mean": Marginal Effects at the Mean. Marginal effects when each predictor is held at its mean or mode. – "median": Marginal Effects at the Median. Marginal effects when each predictor is held at its median or mode.

  - "marginalmeans": Marginal Effects at Marginal Means. See Details section below.
  - "tukey": Marginal Effects at Tukey's 5 numbers.
  - "grid": Marginal Effects on a grid of representative numbers (Tukey's 5 numbers and unique values of categorical predictors).
- `datagrid()` call to specify a custom grid of regressors. For example:
  - `newdata = datagrid(cyl = c(4, 6))`: `cyl` variable equal to 4 and 6 and other regressors fixed at their means or modes.
  - See the Examples section and the `datagrid()` documentation.

`variables` `NULL` or character vector. The subset of variables for which to compute marginal effects.

- `NULL`: compute contrasts for all the variables in the model object (can be slow).
- Character vector: subset of variables (usually faster).

`vcov` Type of uncertainty estimates to report (e.g., for robust standard errors). Acceptable values:

- FALSE: Do not compute standard errors. This can speed up computation considerably.
- TRUE: Unit-level standard errors using the default `vcov(model)` variance-covariance matrix.
- String which indicates the kind of uncertainty estimates to return.
  - Heteroskedasticity-consistent: `"HC"`, `"HC0"`, `"HC1"`, `"HC2"`, `"HC3"`, `"HC4"`, `"HC4m"`, `"HC5"`. See `?sandwich::vcovHC`
  - Heteroskedasticity and autocorrelation consistent: `"HAC"`
  - Mixed-Models degrees of freedom: "satterthwaite", "kenward-roger"
  - Other: `"NeweyWest"`, `"KernHAC"`, `"OPG"`. See the `sandwich` package documentation.
- One-sided formula which indicates the name of cluster variables (e.g., `~unit_id`). This formula is passed to the `cluster` argument of the `sandwich::vcovCL` function.
- Square covariance matrix
- Function which returns a covariance matrix (e.g., `stats::vcov(model)`)

`conf_level` numeric value between 0 and 1. Confidence level to use to build a confidence interval.

`type` string indicates the type (scale) of the predictions used to compute marginal effects or contrasts. This can differ based on the model type, but will typically be a string such as: "response", "link", "probs", or "zero". When an unsupported string is entered, the model-specific list of acceptable values is returned in an error message. When `type` is `NULL`, the default value is used. This default is the first model-related row in the `marginaleffects::type_dictionary` dataframe.

`slope` string indicates the type of slope or (semi-)elasticity to compute:

- "dydx": dY/dX
- "eyex": dY/dX * Y / X

- "eydx": dY/dX * Y
- "dyex": dY/dX / X

`by` Character vector of variable names over which to compute group-wise estimates.

`wts` string or numeric: weights to use when computing average contrasts or marginal-effects. These weights only affect the averaging in `tidy()` or `summary()`, and not the unit-level estimates themselves.

- string: column name of the weights variable in `newdata`. When supplying a column name to `wts`, it is recommended to supply the original data (including the weights variable) explicitly to `newdata`.
- numeric: vector of length equal to the number of rows in the original data or in `newdata` (if supplied).

`hypothesis` specify a hypothesis test or custom contrast using a vector, matrix, string, or string formula.

- String:
  - "pairwise": pairwise differences between estimates in each row.
  - "reference": differences between the estimates in each row and the estimate in the first row.
  - "sequential": difference between an estimate and the estimate in the next row.
  - "revpairwise", "revreference", "revsequential": inverse of the corresponding hypotheses, as described above.
- String formula to specify linear or non-linear hypothesis tests. If the `term` column uniquely identifies rows, terms can be used in the formula. Otherwise, use `b1`, `b2`, etc. to identify the position of each parameter. Examples:
  - `hp = drat`
  - `hp + drat = 12`
  - `b1 + b2 + b3 = 0`
- Numeric vector: Weights to compute a linear combination of (custom contrast between) estimates. Length equal to the number of rows generated by the same function call, but without the `hypothesis` argument.
- Numeric matrix: Each column is a vector of weights, as describe above, used to compute a distinct linear combination of (contrast between) estimates. The column names of the matrix are used as labels in the output.
- See the Examples section below and the vignette: https://vincentarelbundock.github.io/marginaleffect

`eps` NULL or numeric value which determines the step size to use when calculating numerical derivatives: (f(x+eps)-f(x))/eps. When `eps` is `NULL`, the step size is 0.0001 multiplied by the difference between the maximum and minimum values of the variable with respect to which we are taking the derivative. Changing `eps` may be necessary to avoid numerical problems in certain models.

`...` Additional arguments are passed to the `predict()` method supplied by the modeling package.These arguments are particularly useful for mixed-effects or bayesian models (see the online vignettes on the `marginaleffects` website). Available arguments can vary from model to model, depending on the range of supported arguments by each modeling package. See the "Model-Specific Arguments" section of the `?marginaleffects` documentation for a non-exhaustive list of available arguments.

Details

A "marginal effect" is the partial derivative of the regression equation with respect to a variable in the model. This function uses automatic differentiation to compute marginal effects for a vast array of models, including non-linear models with transformations (e.g., polynomials). Uncertainty estimates are computed using the delta method.

The `newdata` argument can be used to control the kind of marginal effects to report:

- Average Marginal Effects (AME)
- Group-Average Marginal Effects (G-AME)
- Marginal Effects at the Mean (MEM) or
- Marginal Effects at User-Specified values (aka Marginal Effects at Representative values, MER).

See the marginaleffects vignette for worked-out examples of each kind of marginal effect.

Numerical derivatives for the `marginaleffects` function are calculated using a simple epsilon difference approach: $\partial Y/\partial X = (f(X+\varepsilon)-f(X))/\varepsilon$, where f is the `predict()` method associated with the model class, and ε is determined by the `eps` argument.

Warning: Some models are particularly sensitive to `eps`, so it is good practice to try different values of this argument.

Standard errors for the marginal effects are obtained using the Delta method. See the "Standard Errors" vignette on the package website for details (link above).

Value

A `data.frame` with one row per observation (per term/group) and several columns:

- `rowid`: row number of the `newdata` data frame
- `type`: prediction type, as defined by the `type` argument
- `group`: (optional) value of the grouped outcome (e.g., categorical outcome models)
- `term`: the variable whose marginal effect is computed
- `dydx`: marginal effect of the term on the outcome for a given combination of regressor values
- `std.error`: standard errors computed by via the delta method.

Vignettes and documentation

Vignettes:

- Adjusted Predictions
- Contrasts
- Marginal Effects
- Marginal Means
- Hypothesis Tests and Custom Contrasts using the Delta Method

Case studies:

- Bayesian Analyses with `brms`

- Causal Inference with the g-Formula
- Elasticity
- Experiments
- Generalized Additive Models
- Mixed effects models
- Multinomial Logit and Discrete Choice Models
- Multiple Imputation
- Plots: interactions, predictions, contrasts, and slopes
- Python NumPyro models in `marginaleffects`
- Unit-level contrasts in logistic regressions

Tips and technical notes:

- 71 Supported Classes of Models
- Index of Functions and Documentation
- Extending `marginaleffects`: add new models or modify existing ones
- Standard Errors and Confidence Intervals
- Tables and Plots
- Performance
- Alternative Software
- Frequently Asked Questions

Model-Specific Arguments

Some model types allow model-specific arguments to modify the nature of marginal effects, predictions, marginal means, and contrasts.

Package	Class	Argument	Documentation
`brms`	`brmsfit`	`ndraws`	brms::posterior_predict
		`re_formula`	
`lme4`	`merMod`	`include_random`	insight::get_predicted
		`re.form`	lme4::predict.merMod
		`allow.new.levels`	lme4::predict.merMod
`glmmTMB`	`glmmTMB`	`re.form`	glmmTMB::predict.glmmTMB
		`allow.new.levels`	glmmTMB::predict.glmmTMB
		`zitype`	glmmTMB::predict.glmmTMB
`mgcv`	`bam`	`exclude`	mgcv::predict.bam
`robustlmm`	`rlmerMod`	`re.form`	robustlmm::predict.rlmerMod
		`allow.new.levels`	robustlmm::predict.rlmerMod

Examples

```
mod <- glm(am ~ hp * wt, data = mtcars, family = binomial)
mfx <- marginaleffects(mod)
head(mfx)

# Average Marginal Effect (AME)
summary(mfx)
tidy(mfx)
plot(mfx)


# Marginal Effect at the Mean (MEM)
marginaleffects(mod, newdata = datagrid())

# Marginal Effect at User-Specified Values
# Variables not explicitly included in `datagrid()` are held at their means
marginaleffects(mod,
                newdata = datagrid(hp = c(100, 110)))

# Group-Average Marginal Effects (G-AME)
# Calculate marginal effects for each observation, and then take the average
# marginal effect within each subset of observations with different observed
# values for the `cyl` variable:
mod2 <- lm(mpg ~ hp * cyl, data = mtcars)
mfx2 <- marginaleffects(mod2, variables = "hp", by = "cyl")
summary(mfx2)

# Marginal Effects at User-Specified Values (counterfactual)
# Variables not explicitly included in `datagrid()` are held at their
# original values, and the whole dataset is duplicated once for each
# combination of the values in `datagrid()`
mfx <- marginaleffects(mod,
                       newdata = datagrid(hp = c(100, 110),
                                          grid_type = "counterfactual"))
head(mfx)

# Heteroskedasticity robust standard errors
marginaleffects(mod, vcov = sandwich::vcovHC(mod))

# hypothesis test: is the `hp` marginal effect at the mean equal to the `drat` marginal effect
mod <- lm(mpg ~ wt + drat, data = mtcars)

marginaleffects(
    mod,
    newdata = "mean",
    hypothesis = "wt = drat")

# same hypothesis test using row indices
marginaleffects(
    mod,
    newdata = "mean",
    hypothesis = "b1 - b2 = 0")
```

```
# same hypothesis test using numeric vector of weights
marginaleffects(
    mod,
    newdata = "mean",
    hypothesis = c(1, -1))

# two custom contrasts using a matrix of weights
lc <- matrix(c(
    1, -1,
    2, 3),
    ncol = 2)
colnames(lc) <- c("Contrast A", "Contrast B")
marginaleffects(
    mod,
    newdata = "mean",
    hypothesis = lc)
```

`marginalmeans` *Marginal Means*

Description

Marginal means are adjusted predictions, averaged across a grid of categorical predictors, holding other numeric predictors at their means. To learn more, read the marginal means vignette, visit the package website, or scroll down this page for a full list of vignettes:

- https://vincentarelbundock.github.io/marginaleffects/articles/marginalmeans.html
- https://vincentarelbundock.github.io/marginaleffects/

Usage

```
marginalmeans(
  model,
  variables = NULL,
  variables_grid = NULL,
  vcov = TRUE,
  conf_level = 0.95,
  type = NULL,
  transform_post = NULL,
  cross = FALSE,
  hypothesis = NULL,
  wts = "equal",
  by = NULL,
  ...
)
```

Arguments

model	Model object
variables	character vector Categorical predictors over which to compute marginal means. NULL calculates marginal means for all logical, character, or factor variables in the dataset used to fit model. Set cross=TRUE to compute marginal means at combinations of the predictors specified in the variables argument.
variables_grid	character vector Categorical predictors used to construct the prediction grid over which adjusted predictions are averaged (character vector). NULL creates a grid with all combinations of all categorical predictors. This grid can be very large when there are many variables and many response levels, so it is advisable to select a limited number of variables in the variables and variables_grid arguments.
vcov	Type of uncertainty estimates to report (e.g., for robust standard errors). Acceptable values: • FALSE: Do not compute standard errors. This can speed up computation considerably. • TRUE: Unit-level standard errors using the default vcov(model) variance-covariance matrix. • String which indicates the kind of uncertainty estimates to return. – Heteroskedasticity-consistent: "HC", "HC0", "HC1", "HC2", "HC3", "HC4", "HC4m", "HC5". See ?sandwich::vcovHC – Heteroskedasticity and autocorrelation consistent: "HAC" – Mixed-Models degrees of freedom: "satterthwaite", "kenward-roger" – Other: "NeweyWest", "KernHAC", "OPG". See the sandwich package documentation. • One-sided formula which indicates the name of cluster variables (e.g., ~unit_id). This formula is passed to the cluster argument of the sandwich::vcovCL function. • Square covariance matrix • Function which returns a covariance matrix (e.g., stats::vcov(model))
conf_level	numeric value between 0 and 1. Confidence level to use to build a confidence interval.
type	string indicates the type (scale) of the predictions used to compute marginal effects or contrasts. This can differ based on the model type, but will typically be a string such as: "response", "link", "probs", or "zero". When an unsupported string is entered, the model-specific list of acceptable values is returned in an error message. When type is NULL, the default value is used. This default is the first model-related row in the marginaleffects:::type_dictionary dataframe. If type is NULL and the default value is "response", the function tries to compute marginal means on the link scale before backtransforming them using the inverse link function.
transform_post	(experimental) A function applied to unit-level adjusted predictions and confidence intervals just before the function returns results. For bayesian models, this function is applied to individual draws from the posterior distribution, before computing summaries.

`cross` TRUE or FALSE

- `FALSE` (default): Marginal means are computed for each predictor individually.
- `TRUE`: Marginal means are computed for each combination of predictors specified in the `variables` argument.

`hypothesis` specify a hypothesis test or custom contrast using a vector, matrix, string, or string formula.

- String:
  - "pairwise": pairwise differences between estimates in each row.
  - "reference": differences between the estimates in each row and the estimate in the first row.
  - "sequential": difference between an estimate and the estimate in the next row.
  - "revpairwise", "revreference", "revsequential": inverse of the corresponding hypotheses, as described above.
- String formula to specify linear or non-linear hypothesis tests. If the `term` column uniquely identifies rows, terms can be used in the formula. Otherwise, use `b1`, `b2`, etc. to identify the position of each parameter. Examples:
  - `hp = drat`
  - `hp + drat = 12`
  - `b1 + b2 + b3 = 0`
- Numeric vector: Weights to compute a linear combination of (custom contrast between) estimates. Length equal to the number of rows generated by the same function call, but without the `hypothesis` argument.
- Numeric matrix: Each column is a vector of weights, as describe above, used to compute a distinct linear combination of (contrast between) estimates. The column names of the matrix are used as labels in the output.
- See the Examples section below and the vignette: https://vincentarelbundock.github.io/marginaleffect

`wts` character value. Weights to use in the averaging.

- "equal": each combination of variables in `variables_grid` gets an equal weight.
- "cells": each combination of values for the variables in the `variables_grid` gets a weight proportional to its frequency in the original data.
- "proportional": each combination of values for the variables in the `variables_grid` – except for those in the `variables` argument – gets a weight proportional to its frequency in the original data.

`by` Collapse marginal means into categories. Data frame with a `by` column of group labels, and merging columns shared by `newdata` or the data frame produced by calling the same function without the `by` argument.

`...` Additional arguments are passed to the `predict()` method supplied by the modeling package.These arguments are particularly useful for mixed-effects or bayesian models (see the online vignettes on the `marginaleffects` website). Available arguments can vary from model to model, depending on the range of supported arguments by each modeling package. See the "Model-Specific Arguments" section of the `?marginaleffects` documentation for a non-exhaustive list of available arguments.

Details

This function begins by calling the `predictions` function to obtain a grid of predictors, and adjusted predictions for each cell. The grid includes all combinations of the categorical variables listed in the `variables` and `variables_grid` arguments, or all combinations of the categorical variables used to fit the model if `variables_grid` is `NULL`. In the prediction grid, numeric variables are held at their means.

After constructing the grid and filling the grid with adjusted predictions, `marginalmeans` computes marginal means for the variables listed in the `variables` argument, by average across all categories in the grid.

`marginalmeans` can only compute standard errors for linear models, or for predictions on the link scale, that is, with the `type` argument set to "link".

The `marginaleffects` website compares the output of this function to the popular `emmeans` package, which provides similar but more advanced functionality: https://vincentarelbundock.github.io/marginaleffects/

Value

Data frame of marginal means with one row per variable-value combination.

Vignettes and documentation

Vignettes:

- Adjusted Predictions
- Contrasts
- Marginal Effects
- Marginal Means
- Hypothesis Tests and Custom Contrasts using the Delta Method

Case studies:

- Bayesian Analyses with `brms`
- Causal Inference with the g-Formula
- Elasticity
- Experiments
- Generalized Additive Models
- Mixed effects models
- Multinomial Logit and Discrete Choice Models
- Multiple Imputation
- Plots: interactions, predictions, contrasts, and slopes
- Python NumPyro models in `marginaleffects`
- Unit-level contrasts in logistic regressions

Tips and technical notes:

- 71 Supported Classes of Models

- Index of Functions and Documentation
- Extending `marginaleffects`: add new models or modify existing ones
- Standard Errors and Confidence Intervals
- Tables and Plots
- Performance
- Alternative Software
- Frequently Asked Questions

Model-Specific Arguments

Some model types allow model-specific arguments to modify the nature of marginal effects, predictions, marginal means, and contrasts.

Package	Class	Argument	Documentation
brms	brmsfit	ndraws	brms::posterior_predict
		re_formula	
lme4	merMod	include_random	insight::get_predicted
		re.form	lme4::predict.merMod
		allow.new.levels	lme4::predict.merMod
glmmTMB	glmmTMB	re.form	glmmTMB::predict.glmmTMB
		allow.new.levels	glmmTMB::predict.glmmTMB
		zitype	glmmTMB::predict.glmmTMB
mgcv	bam	exclude	mgcv::predict.bam
robustlmm	rlmerMod	re.form	robustlmm::predict.rlmerMod
		allow.new.levels	robustlmm::predict.rlmerMod

Examples

```
library(marginaleffects)

# simple marginal means for each level of `cyl`
dat <- mtcars
dat$carb <- factor(dat$carb)
dat$cyl <- factor(dat$cyl)
dat$am <- as.logical(dat$am)
mod <- lm(mpg ~ carb + cyl + am, dat)

marginalmeans(
  mod,
  variables = "cyl")

# collapse levels of cyl by averaging
by <- data.frame(
  cyl = c(4, 6, 8),
  by = c("4 & 6", "4 & 6", "8"))
marginalmeans(mod,
  variables = "cyl",
  by = by)
```

```
# pairwise differences between collapsed levels
marginalmeans(mod,
  variables = "cyl",
  by = by,
  hypothesis = "pairwise")

# cross
marginalmeans(mod,
  variables = c("cyl", "carb"),
  cross = TRUE)

# collapsed cross
by <- expand.grid(
  cyl = unique(mtcars$cyl),
  carb = unique(mtcars$carb))
by$by <- ifelse(
  by$cyl == 4,
  paste("Control:", by$carb),
  paste("Treatment:", by$carb))


# Convert numeric variables to categorical before fitting the model
dat <- mtcars
dat$am <- as.logical(dat$am)
dat$carb <- as.factor(dat$carb)
mod <- lm(mpg ~ hp + am + carb, data = dat)

# Compute and summarize marginal means
mm <- marginalmeans(mod)
summary(mm)

# Contrast between marginal means (carb2 - carb1), or "is the 1st marginal means equal to the 2nd?"
# see the vignette on "Hypothesis Tests and Custom Contrasts" on the `marginaleffects` website.
lc <- c(-1, 1, 0, 0, 0, 0)
marginalmeans(mod, variables = "carb", hypothesis = "b2 = b1")

marginalmeans(mod, variables = "carb", hypothesis = lc)

# Multiple custom contrasts
lc <- matrix(c(
    -2, 1, 1, 0, -1, 1,
    -1, 1, 0, 0, 0, 0
    ),
  ncol = 2,
  dimnames = list(NULL, c("A", "B")))
marginalmeans(mod, variables = "carb", hypothesis = lc)
```

---

plot.marginaleffects *Point-range plot of average marginal effects*

Description

Uses the ggplot2 package to draw a point-range plot of the average marginal effects computed by tidy.

Usage

```
## S3 method for class 'marginaleffects'
plot(x, conf_level = 0.95, ...)
```

Arguments

x	An object produced by the marginaleffects function.
conf_level	numeric value between 0 and 1. Confidence level to use to build a confidence interval.
...	Additional arguments are passed to the predict() method supplied by the modeling package.These arguments are particularly useful for mixed-effects or bayesian models (see the online vignettes on the marginaleffects website). Available arguments can vary from model to model, depending on the range of supported arguments by each modeling package. See the "Model-Specific Arguments" section of the ?marginaleffects documentation for a non-exhaustive list of available arguments.

Details

The tidy function calculates average marginal effects by taking the mean of all the unit-level marginal effects computed by the marginaleffects function.

The standard error of the average marginal effects is obtained by taking the mean of each column of the Jacobian. . Then, we use this "Jacobian at the mean" in the Delta method to obtained standard errors.

In Bayesian models (e.g., brms), we compute Average Marginal Effects by applying the mean function twice. First, we apply it to all marginal effects for each posterior draw, thereby estimating one Average (or Median) Marginal Effect per iteration of the MCMC chain. Second, we take the mean and quantile function to the results of Step 1 to obtain the Average (or Median) Marginal Effect and its associated interval.

Value

A ggplot2 object

See Also

Other plot: plot_cap(), plot_cco(), plot_cme()

Examples

```
mod <- glm(am ~ hp + wt, data = mtcars)
mfx <- marginaleffects(mod)
plot(mfx)
```

plot_cap *Plot Conditional Adjusted Predictions*

Description

This function plots adjusted predictions (y-axis) against values of one or more predictors (x-axis and colors).

Usage

```
plot_cap(
  model,
  condition = NULL,
  type = "response",
  vcov = NULL,
  conf_level = 0.95,
  transform_post = NULL,
  draw = TRUE,
  ...
)
```

Arguments

model	Model object
condition	character vector or named list of length smaller than 4. Character vectors must be the names of the predictor variables to display. The names of the list must The first element is displayed on the x-axis. The second element determines the colors. The third element creates facets. Other variables are held at their means or modes. Lists can include these types of values: • Numeric vector • Function which returns a numeric vector or a set of unique categorical values • Shortcut strings for common reference values: "minmax", "quartile", "threenum"
type	string indicates the type (scale) of the predictions used to compute marginal effects or contrasts. This can differ based on the model type, but will typically be a string such as: "response", "link", "probs", or "zero". When an unsupported string is entered, the model-specific list of acceptable values is returned in an error message. When type is NULL, the default value is used. This default is the first model-related row in the marginaleffects:::type_dictionary dataframe.
vcov	Type of uncertainty estimates to report (e.g., for robust standard errors). Acceptable values: • FALSE: Do not compute standard errors. This can speed up computation considerably.

- TRUE: Unit-level standard errors using the default `vcov(model)` variance-covariance matrix.
- String which indicates the kind of uncertainty estimates to return.
  - Heteroskedasticity-consistent: `"HC"`, `"HC0"`, `"HC1"`, `"HC2"`, `"HC3"`, `"HC4"`, `"HC4m"`, `"HC5"`. See `?sandwich::vcovHC`
  - Heteroskedasticity and autocorrelation consistent: `"HAC"`
  - Mixed-Models degrees of freedom: "satterthwaite", "kenward-roger"
  - Other: `"NeweyWest"`, `"KernHAC"`, `"OPG"`. See the `sandwich` package documentation.
- One-sided formula which indicates the name of cluster variables (e.g., `~unit_id`). This formula is passed to the `cluster` argument of the `sandwich::vcovCL` function.
- Square covariance matrix
- Function which returns a covariance matrix (e.g., `stats::vcov(model)`)

`conf_level` numeric value between 0 and 1. Confidence level to use to build a confidence interval.

`transform_post` (experimental) A function applied to unit-level adjusted predictions and confidence intervals just before the function returns results. For bayesian models, this function is applied to individual draws from the posterior distribution, before computing summaries.

`draw` `TRUE` returns a `ggplot2` plot. `FALSE` returns a `data.frame` of the underlying data.

`...` Additional arguments are passed to the `predict()` method supplied by the modeling package.These arguments are particularly useful for mixed-effects or bayesian models (see the online vignettes on the `marginaleffects` website). Available arguments can vary from model to model, depending on the range of supported arguments by each modeling package. See the "Model-Specific Arguments" section of the `?marginaleffects` documentation for a non-exhaustive list of available arguments.

Value

A `ggplot2` object

See Also

Other plot: `plot.marginaleffects()`, `plot_cco()`, `plot_cme()`

Examples

```
mod <- lm(mpg ~ hp + wt, data = mtcars)
plot_cap(mod, condition = "wt")

mod <- lm(mpg ~ hp * wt * am, data = mtcars)
plot_cap(mod, condition = c("hp", "wt"))

plot_cap(mod, condition = list("hp", wt = "threenum"))
```

```
plot_cap(mod, condition = list("hp", wt = range))
```

`plot_cco` *Plot Conditional Contrasts*

Description

This function plots contrasts (y-axis) against values of predictor(s) variable(s) (x-axis and colors). This is especially useful in models with interactions, where the values of contrasts depend on the values of "condition" variables.

Usage

```
plot_cco(
  model,
  effect = NULL,
  condition = NULL,
  type = "response",
  vcov = NULL,
  conf_level = 0.95,
  transform_pre = "difference",
  transform_post = NULL,
  draw = TRUE,
  ...
)
```

Arguments

`model`	Model object
`effect`	Name of the variable whose contrast we want to plot on the y-axis
`condition`	character vector or named list of length smaller than 3. Character vectors must be the names of the predictor variables to display. The names of the list must The first element is displayed on the x-axis. The second element determines the colors. The third element creates facets. Unspecified variables are held at their means or modes. Lists can include these types of values (see Examples section below): • Numeric vector • Function which returns a numeric vector or a set of unique categorical values • Shortcut strings for common reference values: "minmax", "quartile", "threenum"

`type` string indicates the type (scale) of the predictions used to compute marginal effects or contrasts. This can differ based on the model type, but will typically be a string such as: "response", "link", "probs", or "zero". When an unsupported string is entered, the model-specific list of acceptable values is returned in an error message. When `type` is `NULL`, the default value is used. This default is the first model-related row in the `marginaleffects:::type_dictionary` dataframe.

`vcov` Type of uncertainty estimates to report (e.g., for robust standard errors). Acceptable values:

- FALSE: Do not compute standard errors. This can speed up computation considerably.
- TRUE: Unit-level standard errors using the default `vcov(model)` variance-covariance matrix.
- String which indicates the kind of uncertainty estimates to return.
  - Heteroskedasticity-consistent: `"HC"`, `"HC0"`, `"HC1"`, `"HC2"`, `"HC3"`, `"HC4"`, `"HC4m"`, `"HC5"`. See `?sandwich::vcovHC`
  - Heteroskedasticity and autocorrelation consistent: `"HAC"`
  - Mixed-Models degrees of freedom: "satterthwaite", "kenward-roger"
  - Other: `"NeweyWest"`, `"KernHAC"`, `"OPG"`. See the `sandwich` package documentation.
- One-sided formula which indicates the name of cluster variables (e.g., `~unit_id`). This formula is passed to the `cluster` argument of the `sandwich::vcovCL` function.
- Square covariance matrix
- Function which returns a covariance matrix (e.g., `stats::vcov(model)`)

`conf_level` numeric value between 0 and 1. Confidence level to use to build a confidence interval.

`transform_pre` string or function. How should pairs of adjusted predictions be contrasted?

- string: shortcuts to common contrast functions.
  - Supported shortcuts strings: difference, differenceavg, differenceavgwts, dydx, eyex, eydx, dyex, dydxavg, eyexavg, eydxavg, dyexavg, dydxavgwts, eyexavgwts, eydxavgwts, dyexavgwts, ratio, ratioavg, ratioavgwts, lnratio, lnratioavg, lnratioavgwts, lnor, lnoravg, lnoravgwts, expdydx, expdydxavg, expdydxavgwts
  - See the Transformations section below for definitions of each transformation.
- function: accept two equal-length numeric vectors of adjusted predictions (`hi` and `lo`) and returns a vector of contrasts of the same length, or a unique numeric value.
  - See the Transformations section below for examples of valid functions.

`transform_post` string or function. Transformation applied to unit-level estimates and confidence intervals just before the function returns results. Functions must accept a vector and return a vector of the same length. Support string shortcuts: "exp", "ln"

`draw` `TRUE` returns a `ggplot2` plot. `FALSE` returns a `data.frame` of the underlying data.

`...` Additional arguments are passed to the `predict()` method supplied by the modeling package.These arguments are particularly useful for mixed-effects or bayesian models (see the online vignettes on the `marginaleffects` website). Available arguments can vary from model to model, depending on the range of supported arguments by each modeling package. See the "Model-Specific Arguments" section of the `?marginaleffects` documentation for a non-exhaustive list of available arguments.

Value

A `ggplot2` object

See Also

Other plot: `plot.marginaleffects()`, `plot_cap()`, `plot_cme()`

Examples

```
mod <- lm(mpg ~ hp * drat * factor(am), data = mtcars)

plot_cco(mod, effect = "hp", condition = "drat")

plot_cco(mod, effect = "hp", condition = c("drat", "am"))

plot_cco(mod, effect = "hp", condition = list("am", "drat" = 3:5))

plot_cco(mod, effect = "am", condition = list("hp", "drat" = range))
```

---

plot_cme *Plot Conditional Marginal Effects*

Description

This function plots marginal effects (y-axis) against values of predictor(s) variable(s) (x-axis and colors). This is especially useful in models with interactions, where the values of marginal effects depend on the values of "condition" variables.

Usage

```
plot_cme(
  model,
  effect = NULL,
  condition = NULL,
  type = "response",
  vcov = NULL,
  conf_level = 0.95,
  draw = TRUE,
  ...
)
```

Arguments

model	Model object
effect	Name of the variable whose marginal effect we want to plot on the y-axis
condition	character vector or named list of length smaller than 3. Character vectors must be the names of the predictor variables to display. The names of the list must The first element is displayed on the x-axis. The second element determines the colors. The third element creates facets. Unspecified variables are held at their means or modes. Lists can include these types of values (see Examples section below): • Numeric vector • Function which returns a numeric vector or a set of unique categorical values • Shortcut strings for common reference values: "minmax", "quartile", "threenum"
type	string indicates the type (scale) of the predictions used to compute marginal effects or contrasts. This can differ based on the model type, but will typically be a string such as: "response", "link", "probs", or "zero". When an unsupported string is entered, the model-specific list of acceptable values is returned in an error message. When type is NULL, the default value is used. This default is the first model-related row in the marginaleffects:::type_dictionary dataframe.
vcov	Type of uncertainty estimates to report (e.g., for robust standard errors). Acceptable values: • FALSE: Do not compute standard errors. This can speed up computation considerably. • TRUE: Unit-level standard errors using the default vcov(model) variance-covariance matrix. • String which indicates the kind of uncertainty estimates to return. – Heteroskedasticity-consistent: "HC", "HC0", "HC1", "HC2", "HC3", "HC4", "HC4m", "HC5". See ?sandwich::vcovHC – Heteroskedasticity and autocorrelation consistent: "HAC" – Mixed-Models degrees of freedom: "satterthwaite", "kenward-roger" – Other: "NeweyWest", "KernHAC", "OPG". See the sandwich package documentation. • One-sided formula which indicates the name of cluster variables (e.g., ~unit_id). This formula is passed to the cluster argument of the sandwich::vcovCL function. • Square covariance matrix • Function which returns a covariance matrix (e.g., stats::vcov(model))
conf_level	numeric value between 0 and 1. Confidence level to use to build a confidence interval.
draw	TRUE returns a ggplot2 plot. FALSE returns a data.frame of the underlying data.

`...` Additional arguments are passed to the `predict()` method supplied by the modeling package.These arguments are particularly useful for mixed-effects or bayesian models (see the online vignettes on the `marginaleffects` website). Available arguments can vary from model to model, depending on the range of supported arguments by each modeling package. See the "Model-Specific Arguments" section of the `?marginaleffects` documentation for a non-exhaustive list of available arguments.

Value

A `ggplot2` object

See Also

Other plot: `plot.marginaleffects()`, `plot_cap()`, `plot_cco()`

Examples

```
library(marginaleffects)
mod <- lm(mpg ~ hp * drat * factor(am), data = mtcars)

plot_cme(mod, effect = "hp", condition = "drat")

plot_cme(mod, effect = "hp", condition = c("drat", "am"))

plot_cme(mod, effect = "hp", condition = list("am", "drat" = 3:5))

plot_cme(mod, effect = "am", condition = list("hp", "drat" = range))

plot_cme(mod, effect = "am", condition = list("hp", "drat" = "threenum"))
```

---

posteriordraws *Extract posterior draws from a* `predictions`, `comparisons`, *or* `marginaleffects` *object derived from Bayesian models.*

Description

Extract posterior draws from a `predictions`, `comparisons`, or `marginaleffects` object derived from Bayesian models.

Usage

```
posteriordraws(x, shape = "long")
```

Arguments

x	An object produced by the marginaleffects, comparisons, or predictions functions
shape	string indicating the shape of the output format: • "long": long format data frame • "DxP": Matrix with draws as rows and parameters as columns • "PxD": Matrix with draws as rows and parameters as columns

Value

A data.frame with drawid and draw columns.

---

predictions *Adjusted Predictions*

Description

Outcome predicted by a fitted model on a specified scale for a given combination of values of the predictor variables, such as their observed values, their means, or factor levels (a.k.a. "reference grid"). The tidy() and summary() functions can be used to aggregate the output of predictions(). To learn more, read the predictions vignette, visit the package website, or scroll down this page for a full list of vignettes:

- https://vincentarelbundock.github.io/marginaleffects/articles/predictions.html
- https://vincentarelbundock.github.io/marginaleffects/

Usage

```
predictions(
  model,
  newdata = NULL,
  variables = NULL,
  vcov = TRUE,
  conf_level = 0.95,
  type = NULL,
  by = NULL,
  byfun = NULL,
  wts = NULL,
  transform_post = NULL,
  hypothesis = NULL,
  ...
)
```

Arguments

`model`	Model object
`newdata`	`NULL`, data frame, string, or `datagrid()` call. Determines the predictor values for which to compute marginal effects. • `NULL` (default): Unit-level marginal effects for each observed value in the original dataset. • data frame: Unit-level marginal effects for each row of the `newdata` data frame. • string: – "mean": Marginal Effects at the Mean. Marginal effects when each predictor is held at its mean or mode. – "median": Marginal Effects at the Median. Marginal effects when each predictor is held at its median or mode. – "marginalmeans": Marginal Effects at Marginal Means. See Details section below. – "tukey": Marginal Effects at Tukey's 5 numbers. – "grid": Marginal Effects on a grid of representative numbers (Tukey's 5 numbers and unique values of categorical predictors). • `datagrid()` call to specify a custom grid of regressors. For example: – `newdata = datagrid(cyl = c(4, 6))`: `cyl` variable equal to 4 and 6 and other regressors fixed at their means or modes. – See the Examples section and the `datagrid()` documentation.
`variables`	`NULL`, character vector, or named list. The subset of variables to use for creating a counterfactual grid of predictions. The entire dataset replicated for each unique combination of the variables in this list. See the Examples section below. • Warning: This can use a lot of memory if there are many variables and values, and when the dataset is large. • `NULL`: computes one prediction per row of `newdata` • Named list: names identify the subset of variables of interest and their values. For numeric variables, the `variables` argument supports functions and string shortcuts: – A function which returns a numeric value – Numeric vector: Contrast between the 2nd element and the 1st element of the `x` vector. – "iqr": Contrast across the interquartile range of the regressor. – "sd": Contrast across one standard deviation around the regressor mean. – "2sd": Contrast across two standard deviations around the regressor mean. – "minmax": Contrast between the maximum and the minimum values of the regressor. – "threenum": mean and 1 standard deviation on both sides – "fivenum": Tukey's five numbers #' @param newdata `NULL`, data frame, string, or `datagrid()` call. Determines the grid of predictors on which we make predictions.

- NULL (default): Predictions for each observed value in the original dataset.
- data frame: Predictions for each row of the newdata data frame.
- string:
  - "mean": Predictions at the Mean. Predictions when each predictor is held at its mean or mode.
  - "median": Predictions at the Median. Predictions when each predictor is held at its median or mode.
  - "marginalmeans": Predictions at Marginal Means. See Details section below.
  - "tukey": Predictions at Tukey's 5 numbers.
  - "grid": Predictions on a grid of representative numbers (Tukey's 5 numbers and unique values of categorical predictors).
- datagrid() call to specify a custom grid of regressors. For example:
  - newdata = datagrid(cyl = c(4, 6)): cyl variable equal to 4 and 6 and other regressors fixed at their means or modes.
  - See the Examples section and the datagrid() documentation.

vcov

Type of uncertainty estimates to report (e.g., for robust standard errors). Acceptable values:

- FALSE: Do not compute standard errors. This can speed up computation considerably.
- TRUE: Unit-level standard errors using the default vcov(model) variance-covariance matrix.
- String which indicates the kind of uncertainty estimates to return.
  - Heteroskedasticity-consistent: "HC", "HC0", "HC1", "HC2", "HC3", "HC4", "HC4m", "HC5". See ?sandwich::vcovHC
  - Heteroskedasticity and autocorrelation consistent: "HAC"
  - Mixed-Models degrees of freedom: "satterthwaite", "kenward-roger"
  - Other: "NeweyWest", "KernHAC", "OPG". See the sandwich package documentation.
- One-sided formula which indicates the name of cluster variables (e.g., ~unit_id). This formula is passed to the cluster argument of the sandwich::vcovCL function.
- Square covariance matrix
- Function which returns a covariance matrix (e.g., stats::vcov(model))

conf_level

numeric value between 0 and 1. Confidence level to use to build a confidence interval.

type

string indicates the type (scale) of the predictions used to compute marginal effects or contrasts. This can differ based on the model type, but will typically be a string such as: "response", "link", "probs", or "zero". When an unsupported string is entered, the model-specific list of acceptable values is returned in an error message. When type is NULL, the default value is used. This default is the first model-related row in the marginaleffects:::type_dictionary dataframe.

by

Character vector of variable names over which to compute group-wise estimates.

byfun — A function such as `mean()` or `sum()` used to aggregate estimates within the subgroups defined by the by argument. `NULL` uses the `mean()` function. Must accept a numeric vector and return a single numeric value. This is sometimes used to take the sum or mean of predicted probabilities across outcome or predictor levels. See examples section.

wts — string or numeric: weights to use when computing average contrasts or marginaleffects. These weights only affect the averaging in `tidy()` or `summary()`, and not the unit-level estimates themselves.

- string: column name of the weights variable in `newdata`. When supplying a column name to `wts`, it is recommended to supply the original data (including the weights variable) explicitly to `newdata`.
- numeric: vector of length equal to the number of rows in the original data or in `newdata` (if supplied).

transform_post — (experimental) A function applied to unit-level adjusted predictions and confidence intervals just before the function returns results. For bayesian models, this function is applied to individual draws from the posterior distribution, before computing summaries.

hypothesis — specify a hypothesis test or custom contrast using a vector, matrix, string, or string formula.

- String:
  - "pairwise": pairwise differences between estimates in each row.
  - "reference": differences between the estimates in each row and the estimate in the first row.
  - "sequential": difference between an estimate and the estimate in the next row.
  - "revpairwise", "revreference", "revsequential": inverse of the corresponding hypotheses, as described above.
- String formula to specify linear or non-linear hypothesis tests. If the `term` column uniquely identifies rows, terms can be used in the formula. Otherwise, use `b1`, `b2`, etc. to identify the position of each parameter. Examples:
  - `hp = drat`
  - `hp + drat = 12`
  - `b1 + b2 + b3 = 0`
- Numeric vector: Weights to compute a linear combination of (custom contrast between) estimates. Length equal to the number of rows generated by the same function call, but without the `hypothesis` argument.
- Numeric matrix: Each column is a vector of weights, as describe above, used to compute a distinct linear combination of (contrast between) estimates. The column names of the matrix are used as labels in the output.
- See the Examples section below and the vignette: https://vincentarelbundock.github.io/marginaleffect

... — Additional arguments are passed to the `predict()` method supplied by the modeling package.These arguments are particularly useful for mixed-effects or bayesian models (see the online vignettes on the `marginaleffects` website). Available arguments can vary from model to model, depending on the range of

supported arguments by each modeling package. See the "Model-Specific Arguments" section of the ?marginaleffects documentation for a non-exhaustive list of available arguments.

Details

The newdata argument, the tidy() function, and datagrid() function can be used to control the kind of predictions to report:

- Average Predictions
- Predictions at the Mean
- Predictions at User-Specified values (aka Predictions at Representative values).

When possible, predictions() delegates the computation of confidence intervals to the insight::get_predicted() function, which uses back transformation to produce adequate confidence intervals on the scale specified by the type argument. When this is not possible, predictions() uses the Delta Method to compute standard errors around adjusted predictions, and builds symmetric confidence intervals. These naive symmetric intervals may not always be appropriate. For instance, they may stretch beyond the bounds of a binary response variables.

Value

A data.frame with one row per observation and several columns:

- rowid: row number of the newdata data frame
- type: prediction type, as defined by the type argument
- group: (optional) value of the grouped outcome (e.g., categorical outcome models)
- predicted: predicted outcome
- std.error: standard errors computed by the insight::get_predicted function or, if unavailable, via marginaleffects delta method functionality.
- conf.low: lower bound of the confidence interval (or equal-tailed interval for bayesian models)
- conf.high: upper bound of the confidence interval (or equal-tailed interval for bayesian models)

Vignettes and documentation

Vignettes:

- Adjusted Predictions
- Contrasts
- Marginal Effects
- Marginal Means
- Hypothesis Tests and Custom Contrasts using the Delta Method

Case studies:

- Bayesian Analyses with brms

- Causal Inference with the g-Formula
- Elasticity
- Experiments
- Generalized Additive Models
- Mixed effects models
- Multinomial Logit and Discrete Choice Models
- Multiple Imputation
- Plots: interactions, predictions, contrasts, and slopes
- Python NumPyro models in `marginaleffects`
- Unit-level contrasts in logistic regressions

Tips and technical notes:

- 71 Supported Classes of Models
- Index of Functions and Documentation
- Extending `marginaleffects`: add new models or modify existing ones
- Standard Errors and Confidence Intervals
- Tables and Plots
- Performance
- Alternative Software
- Frequently Asked Questions

Model-Specific Arguments

Some model types allow model-specific arguments to modify the nature of marginal effects, predictions, marginal means, and contrasts.

Package	Class	Argument	Documentation
`brms`	`brmsfit`	`ndraws`	brms::posterior_predict
		`re_formula`	
`lme4`	`merMod`	`include_random`	insight::get_predicted
		`re.form`	lme4::predict.merMod
		`allow.new.levels`	lme4::predict.merMod
`glmmTMB`	`glmmTMB`	`re.form`	glmmTMB::predict.glmmTMB
		`allow.new.levels`	glmmTMB::predict.glmmTMB
		`zitype`	glmmTMB::predict.glmmTMB
`mgcv`	`bam`	`exclude`	mgcv::predict.bam
`robustlmm`	`rlmerMod`	`re.form`	robustlmm::predict.rlmerMod
		`allow.new.levels`	robustlmm::predict.rlmerMod

Examples

```
# Adjusted Prediction for every row of the original dataset
mod <- lm(mpg ~ hp + factor(cyl), data = mtcars)
```

```
pred <- predictions(mod)
head(pred)

# Adjusted Predictions at User-Specified Values of the Regressors
predictions(mod, newdata = datagrid(hp = c(100, 120), cyl = 4))

m <- lm(mpg ~ hp + drat + factor(cyl) + factor(am), data = mtcars)
predictions(m, newdata = datagrid(FUN_factor = unique, FUN_numeric = median))

# Average Adjusted Predictions (AAP)
library(dplyr)
mod <- lm(mpg ~ hp * am * vs, mtcars)

pred <- predictions(mod)
summary(pred)

predictions(mod, by = "am")

# Conditional Adjusted Predictions
plot_cap(mod, condition = "hp")

# Counterfactual predictions with the `variables` argument
# the `mtcars` dataset has 32 rows

mod <- lm(mpg ~ hp + am, data = mtcars)
p <- predictions(mod)
head(p)
nrow(p)

# counterfactual predictions obtained by replicating the entire for different
# values of the predictors
p <- predictions(mod, variables = list(hp = c(90, 110)))
nrow(p)


# hypothesis test: is the prediction in the 1st row equal to the prediction in the 2nd row
mod <- lm(mpg ~ wt + drat, data = mtcars)

predictions(
    mod,
    newdata = datagrid(wt = 2:3),
    hypothesis = "b1 = b2")

# same hypothesis test using row indices
predictions(
    mod,
    newdata = datagrid(wt = 2:3),
    hypothesis = "b1 - b2 = 0")

# same hypothesis test using numeric vector of weights
predictions(
    mod,
    newdata = datagrid(wt = 2:3),
```

```
    hypothesis = c(1, -1))

# two custom contrasts using a matrix of weights
lc <- matrix(c(
    1, -1,
    2, 3),
    ncol = 2)
predictions(
    mod,
    newdata = datagrid(wt = 2:3),
    hypothesis = lc)


# `by` argument
mod <- lm(mpg ~ hp * am * vs, data = mtcars)
predictions(mod, by = c("am", "vs"))

library(nnet)
nom <- multinom(factor(gear) ~ mpg + am * vs, data = mtcars, trace = FALSE)

# first 5 raw predictions
predictions(nom, type = "probs") |> head()

# average predictions
predictions(nom, type = "probs", by = "group") |> summary()

by <- data.frame(
    group = c("3", "4", "5"),
    by = c("3,4", "3,4", "5"))

predictions(nom, type = "probs", by = by)

# sum of predicted probabilities for combined response levels
mod <- multinom(factor(cyl) ~ mpg + am, data = mtcars, trace = FALSE)
by <- data.frame(
    by = c("4,6", "4,6", "8"),
    group = as.character(c(4, 6, 8)))
predictions(mod, newdata = "mean", byfun = sum, by = by)
```

`summary.comparisons` *Summarize a* `comparisons` *object*

Description

Summarize a `comparisons` object

Usage

```
## S3 method for class 'comparisons'
summary(object, conf_level = NULL, transform_avg = NULL, ...)
```

Arguments

object	An object produced by the comparisons function
conf_level	numeric value between 0 and 1. Confidence level to use to build a confidence interval.
transform_avg	A function applied to the estimates and confidence intervals *after* the unit-level estimates have been averaged.
...	Additional arguments are passed to the predict() method supplied by the modeling package.These arguments are particularly useful for mixed-effects or bayesian models (see the online vignettes on the marginaleffects website). Available arguments can vary from model to model, depending on the range of supported arguments by each modeling package. See the "Model-Specific Arguments" section of the ?marginaleffects documentation for a non-exhaustive list of available arguments.

Value

Data frame of summary statistics for an object produced by the comparisons function

See Also

Other summary: glance.marginaleffects(), reexports, summary.marginaleffects(), summary.marginalmeans(), summary.predictions(), tidy.comparisons(), tidy.deltamethod(), tidy.marginaleffects(), tidy.marginalmeans(), tidy.predictions()

Examples

```
mod <- lm(mpg ~ hp * wt + factor(gear), data = mtcars)
con <- comparisons(mod)

# average marginal effects
summary(con)
```

---

summary.marginaleffects

Summarize a marginaleffects *object*

Description

Summarize a marginaleffects object

Usage

```
## S3 method for class 'marginaleffects'
summary(object, conf_level = NULL, ...)
```

Arguments

object	An object produced by the marginaleffects function
conf_level	numeric value between 0 and 1. Confidence level to use to build a confidence interval.
...	Additional arguments are passed to the predict() method supplied by the modeling package.These arguments are particularly useful for mixed-effects or bayesian models (see the online vignettes on the marginaleffects website). Available arguments can vary from model to model, depending on the range of supported arguments by each modeling package. See the "Model-Specific Arguments" section of the ?marginaleffects documentation for a non-exhaustive list of available arguments.

Value

Data frame of summary statistics for an object produced by the marginaleffects function

See Also

Other summary: glance.marginaleffects(), reexports, summary.comparisons(), summary.marginalmeans(), summary.predictions(), tidy.comparisons(), tidy.deltamethod(), tidy.marginaleffects(), tidy.marginalmeans(), tidy.predictions()

Examples

```
mod <- lm(mpg ~ hp * wt + factor(gear), data = mtcars)
mfx <- marginaleffects(mod)

# average marginal effects
summary(mfx)
```

---

summary.marginalmeans *Summarize a* marginalmeans *object*

Description

Summarize a marginalmeans object

Usage

```
## S3 method for class 'marginalmeans'
summary(object, transform_avg = NULL, conf_level = 0.95, ...)
```

Arguments

object	An object produced by the marginalmeans function
transform_avg	A function applied to the estimates and confidence intervals *after* the unit-level estimates have been averaged.
conf_level	numeric value between 0 and 1. Confidence level to use to build a confidence interval.
...	Additional arguments are passed to the predict() method supplied by the modeling package.These arguments are particularly useful for mixed-effects or bayesian models (see the online vignettes on the marginaleffects website). Available arguments can vary from model to model, depending on the range of supported arguments by each modeling package. See the "Model-Specific Arguments" section of the ?marginaleffects documentation for a non-exhaustive list of available arguments.

Value

Data frame of summary statistics for an object produced by the marginalmeans function

See Also

Other summary: glance.marginaleffects(), reexports, summary.comparisons(), summary.marginaleffects(), summary.predictions(), tidy.comparisons(), tidy.deltamethod(), tidy.marginaleffects(), tidy.marginalmeans(), tidy.predictions()

---

summary.predictions *Summarize a* predictions *object*

Description

Summarize a predictions object

Usage

```
## S3 method for class 'predictions'
summary(object, conf_level = NULL, transform_avg = NULL, ...)
```

Arguments

object	An object produced by the predictions function
conf_level	numeric value between 0 and 1. Confidence level to use to build a confidence interval.
transform_avg	A function applied to the estimates and confidence intervals *after* the unit-level estimates have been averaged.

...	Additional arguments are passed to the predict() method supplied by the modeling package.These arguments are particularly useful for mixed-effects or bayesian models (see the online vignettes on the marginaleffects website). Available arguments can vary from model to model, depending on the range of supported arguments by each modeling package. See the "Model-Specific Arguments" section of the ?marginaleffects documentation for a non-exhaustive list of available arguments.

Value

Data frame of summary statistics for an object produced by the predictions function

See Also

Other summary: glance.marginaleffects(), reexports, summary.comparisons(), summary.marginaleffects(), summary.marginalmeans(), tidy.comparisons(), tidy.deltamethod(), tidy.marginaleffects(), tidy.marginalmeans(), tidy.predictions()

---

tidy.comparisons *Tidy a* comparisons *object*

Description

Calculate average contrasts by taking the mean of all the unit-level contrasts computed by the predictions function.

Usage

```
## S3 method for class 'comparisons'
tidy(x, conf_level = NULL, transform_avg = NULL, ...)
```

Arguments

x	An object produced by the comparisons function.
conf_level	numeric value between 0 and 1. Confidence level to use to build a confidence interval. The default NULL uses the conf_level value used in the original call to comparisons().
transform_avg	A function applied to the estimates and confidence intervals *after* the unit-level estimates have been averaged.
...	Additional arguments are passed to the predict() method supplied by the modeling package.These arguments are particularly useful for mixed-effects or bayesian models (see the online vignettes on the marginaleffects website). Available arguments can vary from model to model, depending on the range of supported arguments by each modeling package. See the "Model-Specific Arguments" section of the ?marginaleffects documentation for a non-exhaustive list of available arguments.

Details

To compute standard errors around the average marginaleffects, we begin by applying the mean function to each column of the Jacobian. Then, we use this matrix in the Delta method to obtained standard errors.

In Bayesian models (e.g., `brms`), we compute Average Marginal Effects by applying the mean function twice. First, we apply it to all marginal effects for each posterior draw, thereby estimating one Average (or Median) Marginal Effect per iteration of the MCMC chain. Second, we calculate the mean and the `quantile` function to the results of Step 1 to obtain the Average Marginal Effect and its associated interval.

Value

A "tidy" `data.frame` of summary statistics which conforms to the `broom` package specification.

See Also

Other summary: `glance.marginaleffects()`, `reexports`, `summary.comparisons()`, `summary.marginaleffects()`, `summary.marginalmeans()`, `summary.predictions()`, `tidy.deltamethod()`, `tidy.marginaleffects()`, `tidy.marginalmeans()`, `tidy.predictions()`

Examples

```
mod <- lm(mpg ~ factor(gear), data = mtcars)
contr <- comparisons(mod, variables = list(gear = "sequential"))
tidy(contr)
```

`tidy.deltamethod` *Tidy a* `deltamethod` *object*

Description

Tidy a `deltamethod` object

Usage

```
## S3 method for class 'deltamethod'
tidy(x, ...)
```

Arguments

`x`	An object produced by the `marginaleffects` function.
`...`	Additional arguments are passed to the `predict()` method supplied by the modeling package.These arguments are particularly useful for mixed-effects or bayesian models (see the online vignettes on the `marginaleffects` website). Available arguments can vary from model to model, depending on the range of supported arguments by each modeling package. See the "Model-Specific Arguments" section of the `?marginaleffects` documentation for a non-exhaustive list of available arguments.

See Also

Other summary: glance.marginaleffects(), reexports, summary.comparisons(), summary.marginaleffects(), summary.marginalmeans(), summary.predictions(), tidy.comparisons(), tidy.marginaleffects(), tidy.marginalmeans(), tidy.predictions()

`tidy.marginaleffects` *Tidy a* `marginaleffects` *object*

Description

Tidy a `marginaleffects` object

Usage

```
## S3 method for class 'marginaleffects'
tidy(x, conf_level = NULL, ...)
```

Arguments

`x`	An object produced by the `marginaleffects` function.
`conf_level`	numeric value between 0 and 1. Confidence level to use to build a confidence interval. The default `NULL` uses the `conf_level` value used in the original call to `marginaleffects()`.
`...`	Additional arguments are passed to the `predict()` method supplied by the modeling package.These arguments are particularly useful for mixed-effects or bayesian models (see the online vignettes on the `marginaleffects` website). Available arguments can vary from model to model, depending on the range of supported arguments by each modeling package. See the "Model-Specific Arguments" section of the `?marginaleffects` documentation for a non-exhaustive list of available arguments.

Details

The `tidy` function calculates average marginal effects by taking the mean of all the unit-level marginal effects computed by the `marginaleffects` function.

The standard error of the average marginal effects is obtained by taking the mean of each column of the Jacobian. . Then, we use this "Jacobian at the mean" in the Delta method to obtained standard errors.

In Bayesian models (e.g., `brms`), we compute Average Marginal Effects by applying the mean function twice. First, we apply it to all marginal effects for each posterior draw, thereby estimating one Average (or Median) Marginal Effect per iteration of the MCMC chain. Second, we take the `mean` and `quantile` function to the results of Step 1 to obtain the Average (or Median) Marginal Effect and its associated interval.

Value

A "tidy" `data.frame` of summary statistics which conforms to the `broom` package specification.

See Also

Other summary: glance.marginaleffects(), reexports, summary.comparisons(), summary.marginaleffects(), summary.marginalmeans(), summary.predictions(), tidy.comparisons(), tidy.deltamethod(), tidy.marginalmeans(), tidy.predictions()

Examples

```
mod <- lm(mpg ~ hp * wt + factor(gear), data = mtcars)
mfx <- marginaleffects(mod)

# average marginal effects
tidy(mfx)
```

---

tidy.marginalmeans *Tidy a* marginalmeans *object*

Description

Tidy a marginalmeans object

Usage

```
## S3 method for class 'marginalmeans'
tidy(x, conf_level = 0.95, transform_avg = NULL, ...)
```

Arguments

x	An object produced by the marginalmeans function.
conf_level	numeric value between 0 and 1. Confidence level to use to build a confidence interval. The default NULL uses the conf_level value used in the original call to marginaleffects().
transform_avg	A function applied to the estimates and confidence intervals *after* the unit-level estimates have been averaged.
...	Additional arguments are passed to the predict() method supplied by the modeling package.These arguments are particularly useful for mixed-effects or bayesian models (see the online vignettes on the marginaleffects website). Available arguments can vary from model to model, depending on the range of supported arguments by each modeling package. See the "Model-Specific Arguments" section of the ?marginaleffects documentation for a non-exhaustive list of available arguments.

Value

A "tidy" data.frame of summary statistics which conforms to the broom package specification.

See Also

Other summary: glance.marginaleffects(), reexports, summary.comparisons(), summary.marginaleffects(), summary.marginalmeans(), summary.predictions(), tidy.comparisons(), tidy.deltamethod(), tidy.marginaleffects(), tidy.predictions()

---

tidy.predictions *Tidy a* predictions *object*

Description

Calculate average adjusted predictions by taking the mean of all the unit-level adjusted predictions computed by the predictions function.

Usage

```
## S3 method for class 'predictions'
tidy(x, conf_level = NULL, transform_avg = NULL, ...)
```

Arguments

x	An object produced by the predictions function.
conf_level	numeric value between 0 and 1. Confidence level to use to build a confidence interval. The default NULL uses the conf_level value used in the original call to predictions().
transform_avg	A function applied to the estimates and confidence intervals *after* the unit-level estimates have been averaged.
...	Additional arguments are passed to the predict() method supplied by the modeling package.These arguments are particularly useful for mixed-effects or bayesian models (see the online vignettes on the marginaleffects website). Available arguments can vary from model to model, depending on the range of supported arguments by each modeling package. See the "Model-Specific Arguments" section of the ?marginaleffects documentation for a non-exhaustive list of available arguments.

Value

A "tidy" data.frame of summary statistics which conforms to the broom package specification.

See Also

Other summary: glance.marginaleffects(), reexports, summary.comparisons(), summary.marginaleffects(), summary.marginalmeans(), summary.predictions(), tidy.comparisons(), tidy.deltamethod(), tidy.marginaleffects(), tidy.marginalmeans()

Examples

```
mod <- lm(mpg ~ hp * wt + factor(gear), data = mtcars)
mfx <- predictions(mod)
tidy(mfx)
```

Index