Package: marginaleffects (via r-universe)

November 2, 2026

Title Predictions, Comparisons, Slopes, Marginal Means, and Hypothesis Tests

Version 1.0.0.10007

Description Compute and plot predictions, slopes, marginal means, and comparisons (contrasts, risk ratios, odds, etc.) for over 100 classes of statistical and machine learning models in R. Conduct linear and non-linear hypothesis tests, or equivalence tests. Calculate uncertainty estimates using the delta method, bootstrapping, or simulation-based inference. Details can be found in Arel-Bundock, Greifer, and Heiss (2024) <doi:10.18637/jss.v111.i09>.

License GPL (>= 3)

Copyright inst/COPYRIGHTS

LazyData true

Encoding UTF-8

URL https://marginaleffects.com/

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

Roxygen list(markdown = TRUE)

Depends R (>= 3.6.0)

Imports backports, checkmate, data.table (>= 1.16.4), generics, Formula, insight (>= 1.3.0), Matrix, methods, rlang

Suggests AER, Amelia, afex, aod, adrftools, arrow, bayestestR, bench, betareg, BH, bife, biglm, blme, boot, brglm2, brms, brmsmargins, broom, car, carData, causaldata, clarify, cjoint, cobalt, collapse, conflicted, countrycode, covr, crch, DALEXtra, DCchoice, DeclareDesign, dbarts, DirichletReg, distributional, dfidx, dplyr, emmeans, equivalence, estimatr, fixest, flexsurv, fmeffects, fontquiver, future, future.apply, fwb, gam, gamlss, gamlss.dist, geepack, ggdist, ggokabeito, ggplot2, ggrepel, glmmTMB, glmtoolbox, glmx, haven, here, itsadug, ivreg, kableExtra, lme4, lmerTest, logistf, magrittr, margins, MatchIt, MASS, mclogit, MCMCglmm, mhurdle, missRanger,

mgcv, mice, miceadds, mlogit, mlr3verse, mmrm (>= 0.3.14), modelbased, modelsummary, multcomp, mvgam, mvtnorm, nanoparquet, nlme, nestedLogit, nnet, numDeriv, nycflights13, optmatch, ordbetareg, ordinal, parameters, parsnip, partykit, patchwork, pkgdown, phylolm, pbapply, plm, polspline, posterior, probably, pscl, purrr, quantreg, quantregForest, Rchoice, REndo, rcmdcheck, Rdatasets, remotes, rmarkdown, rms, robust, robustbase, robustlmm, rsample, rstanarm, rstantools, rstpm2, rstudioapi, rsvg, sampleSelection, sandwich, scam, spelling, speedglm, splines, survey, svyVGAM, survival, svglite, systemfit, systemfonts, tibble, tictoc, tidymodels, tidyr, tidyverse, tinyplot, tinysnapshot (>= 0.0.7), tinytable (>= 0.6.1), tinytest, titanic, truncreg, tsModel, VGAM, WeightIt, withr, workflows, yaml, xgboost, altdoc, knitr, quarto

Collate 'add_hypothesis.R' 'add_rowid.R' 'attributes.R' 'backtransform.R' 'broom.R' 'by.R' 'class.R' 'comparisons.R' 'comparisons_plan.R' 'comparisons_plan_bayesian.R' 'comparisons_plan_frequentist.R' 'components.R' 'construct_call.R' 'datagrid.R' 'equivalence.R' 'get_ci.R' 'get_coef.R' 'get_comparisons_data.R' 'get_comparisons_data_character.R' 'get_comparisons_data_factor.R' 'get_comparisons_data_logical.R' 'get_comparisons_data_numeric.R' 'get_dataset.R' 'get_degrees_of_freedom.R' 'get_draws.R' 'get_group_names.R' 'get_hypotheses.R' 'get_jacobian.R' 'get_jacobian_analytic.R' 'get_labels.R' 'get_model_matrix.R' 'get_modeldata.R' 'get_predict.R' 'get_se_delta.R' 'get_vcov.R' 'hush.R' 'hypotheses.R' 'hypotheses_joint.R' 'hypothesis.R' 'hypothesis_compile.R' 'hypothesis_formula.R' 'hypothesis_function.R' 'hypothesis_matrix.R' 'hypothesis_string.R' 'imputation.R' 'inferences.R' 'inferences_boot.R' 'inferences_conformal.R' 'inferences_conformal_cv.R' 'inferences_conformal_full.R' 'inferences_conformal_quantile.R' 'inferences_conformal_split.R' 'inferences_fwb.R' 'inferences_rsample.R' 'inferences_simulation.R' 'jacobian_stage.R' 'matrix_apply.R' 'mean_or_mode.R' 'methods.R' 'set_coef.R' 'methods_AER.R' 'sanity_model.R' 'methods_DirichletReg.R' 'methods_MASS.R' 'methods_MCMCglmm.R' 'methods_Rchoice.R' 'methods_WeightIt.R' 'methods_aaa.R' 'methods_afex.R' 'methods_aod.R' 'methods_betareg.R' 'methods_bife.R' 'methods_biglm.R' 'methods_nnet.R' 'methods_brglm2.R' 'methods_brms.R' 'methods_crch.R' 'methods_dataframe.R' 'methods_dbarts.R' 'methods_fixest.R' 'methods_flexsurv.R' 'methods_gamlss.R' 'methods_geepack.R' 'methods_glmmTMB.R' 'methods_glmtoolbox.R' 'methods_glmx.R'

'methods_ivreg.R' 'methods_lme4.R' 'methods_mclogit.R'
'methods_mgcv.R' 'methods_mhurdle.R' 'methods_mice.R'
'methods_mlm.R' 'methods_mlogit.R' 'methods_mlr3.R'
'methods_mmrm.R' 'methods_nestedLogit.R' 'methods_nlme.R'
'methods_ordinal.R' 'methods_plm.R' 'methods_pscl.R'
'methods_quantreg.R' 'methods_rms.R' 'methods_robustlmm.R'
'methods_rstanarm.R' 'methods_rstpm2.R'
'methods_sampleSelection.R' 'methods_scam.R' 'methods_stats.R'
'methods_survey.R' 'methods_survival.R' 'methods_svyVGAM.R'
'methods_systemfit.R' 'methods_tidymodels.R' 'methods_tobit1.R'
'modelarchive.R' 'multcomp.R' 'myTryCatch.R' 'package.R'
'pad.R' 'plan_replay.R' 'plot.R' 'plot_build.R'
'plot_comparisons.R' 'plot_predictions.R' 'plot_slopes.R'
'predictions.R' 'predictions_plan.R'
'predictions_plan_bayesian.R' 'predictions_plan_frequentist.R'
'print.R' 'prune.R' 'refit.R' 'sanitize_by.R'
'sanitize_comparison.R' 'sanitize_condition.R'
'sanitize_conf_level.R' 'sanitize_hypothesis_formula.R'
'sanitize_interaction.R' 'sanitize_newdata.R'
'sanitize_numderiv.R' 'sanitize_reserved.R' 'sanitize_type.R'
'sanitize_variables.R' 'sanitize_vcov.R' 'sanity.R'
'sanity_dots.R' 'sanity_inferences.R' 'sanity_multcomp.R'
'settings.R' 'slopes.R' 'sort.R' 'tinytest.R'
'type_dictionary.R' 'unpack_matrix_cols.R' 'utils.R'
'vcov_unconditional.R' 'vcov_unconditional_sanitization.R'
'zzz.R'

Language en-US

VignetteBuilder knitr, quarto

Config/roxygen2/version 8.0.0

RoxygenNote 7.3.3

Repository https://grantmcdermott.r-universe.dev

Date/Publication 2026-10-23 13:30:07 UTC

RemoteUrl https://github.com/vincentarelbundock/marginaleffects

RemoteRef HEAD

RemoteSha a4ac200424ec397862b434b67c8cd8fff131e972

RemoteSubdir r

Contents

comparisons 4
components.marginaleffects 21
datagrid 22
get_dataset 25
get_draws 26
get_modeldata 27

hypotheses 28
inferences 35
plot_comparisons 39
plot_predictions 43
plot_slopes 49
predictions 53
print.marginaleffects 68
prune.marginaleffects 70
refit 70
slopes 71
vcovUnconditional 85

Index **88**

`comparisons` *Comparisons Between Predictions Made With Different Regressor Values*

Description

Predict the outcome variable at different regressor values (e.g., college graduates vs. others), and compare those predictions by computing a difference, ratio, or some other function. `comparisons()` can return many quantities of interest, such as contrasts, differences, risk ratios, changes in log odds, lift, slopes, elasticities, etc.

- `comparisons()`: unit-level (conditional) estimates.
- `avg_comparisons()`: average (marginal) estimates.

`variables` identifies the focal regressors whose "effect" we are interested in. `comparison` determines how predictions with different regressor values are compared (difference, ratio, odds, etc.). The `newdata` argument and the `datagrid()` function control where statistics are evaluated in the predictor space: "at observed values", "at the mean", "at representative values", etc.

See the comparisons chapter on the package website for worked examples and case studies:

- https://marginaleffects.com/chapters/comparisons.html
- https://marginaleffects.com/

Usage

```
comparisons(
  model,
  newdata = NULL,
  variables = NULL,
  comparison = "difference",
  type = NULL,
  vcov = TRUE,
  by = FALSE,
  conf_level = 0.95,
```

```
  transform = NULL,
  cross = FALSE,
  wts = FALSE,
  hypothesis = NULL,
  equivalence = NULL,
  df = Inf,
  eps = NULL,
  numderiv = "fdforward",
  ...
)

avg_comparisons(
  model,
  newdata = NULL,
  variables = NULL,
  type = NULL,
  vcov = TRUE,
  by = TRUE,
  conf_level = 0.95,
  comparison = "difference",
  transform = NULL,
  cross = FALSE,
  wts = FALSE,
  hypothesis = NULL,
  equivalence = NULL,
  df = Inf,
  eps = NULL,
  numderiv = "fdforward",
  ...
)
```

Arguments

`model`	Model object
`newdata`	Grid of predictor values at which we evaluate the comparisons. • Warning: When `newdata` is `NULL`, the data is retrieved using `get_modeldata()`, which may have to extract it from the modeling environment. This can produce unexpected results if the data has been modified, or when called inside `lapply()`, Shiny apps, or nested functions. To avoid potential issues, consider using `set_modeldata()` to attach the training data to the model object explicitly. • `NULL` (default): Unit-level contrasts for each observed value in the dataset (empirical distribution). The dataset is retrieved using `get_modeldata()`. • 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.
  - "balanced": Contrasts evaluated on a balanced grid with every combination of categories and numeric variables held at their 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.
- `subset()` call with a single argument to select a subset of the dataset used to fit the model, ex: `newdata = subset(treatment == 1)`
- `dplyr::filter()` call with a single argument to select a subset of the dataset used to fit the model, ex: `newdata = filter(treatment == 1)`

`variables` Focal variables

- `NULL`: compute comparisons 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
    - "minmax": The highest and lowest levels of a factor.
    - "revpairwise", "revreference", "revsequential": inverse of the corresponding hypotheses.
    - Vector of length 2 with the two values to compare.
    - Data frame with the same number of rows as `newdata`, with two columns of "lo" and "hi" values to compare.
    - Function that accepts a vector and returns a data frame with two columns of "lo" and "hi" values to compare. See examples below.
  - Logical variables:
    - NULL: contrast between TRUE and FALSE
    - Data frame with the same number of rows as `newdata`, with two columns of "lo" and "hi" values to compare.

    - Function that accepts a vector and returns a data frame with two columns of "lo" and "hi" values to compare. See examples below.
  - Numeric variables:
    - Numeric of length 1: Forward contrast for a gap of `x`, computed between the observed value and the observed value plus `x`. Users can set a global option to get a "center" or "backward" contrast instead: `options(marginaleffects_contrast_direction="center")`
    - Numeric vector of length 2: Contrast between the largest and the smallest elements of the `x` vector.
    - Data frame with the same number of rows as `newdata`, with two columns of "lo" and "hi" values to compare.
    - Function that accepts a vector and returns a data frame with two columns of "lo" and "hi" 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))`
    - `variables = list(hp = function(x) data.frame(low = x - 5, high = x + 10))`
    - See the Examples section below for more.

`comparison` How should pairs of predictions be compared? Difference, ratio, odds ratio, or user-defined functions.

- 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, lift, liftavg, liftavgwts, expdydx, expdydxavg, expdydxavgwts
- 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 "Comparison functions" section below for a list of common transformations and the definitions of available shortcuts.

`type` string indicates the type (scale) of the predictions used to compute contrasts or slopes. 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 first entry in the error message is used by default. See the Type section in the documentation below.

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"`
  - Unconditional: `"unconditional"` accounts for sampling variation in the empirical covariate distribution for averaged or aggregated predictions, comparisons, and slopes. Hypotheses applied directly to unit-level effects are rejected. Use `vcovUnconditional(cluster = ~cluster)` for one-way clustered unconditional inference.
  - Mixed-Models degrees of freedom: "satterthwaite", "kenward-roger"
  - Other: `"NeweyWest"`, `"KernHAC"`, `"OPG"`. See the `sandwich` package documentation.
  - `"rsample"`, `"boot"`, `"fwb"`, or `"simulation"`: forward the result to `inferences()` using that method.
- 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)`)

by

Aggregate unit-level estimates (aka, marginalize, average over). Valid inputs:

- `FALSE`: return the original unit-level estimates.
- `TRUE`: aggregate estimates for each term.
- 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.
- For more complex aggregations, you can use the `FUN` argument of the `hypotheses()` function. See that function's documentation and the Hypothesis Test vignettes on the `marginaleffects` website.

conf_level

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

transform

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

- `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").

wts — logical, string or numeric: weights to use when computing average predictions, contrasts or slopes. These weights only affect the averaging in avg_*() or with the by argument, and not unit-level estimates. See ?weighted.mean

- 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).
- FALSE: Equal weights.
- TRUE: Extract weights from the fitted object with insight::find_weights() and use them when taking weighted averages of estimates. Warning: newdata=datagrid() returns a single average weight, which is equivalent to using wts=FALSE

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

- Number: The null hypothesis used in the computation of Z and p (before applying transform).
- String: Equation to specify linear or non-linear hypothesis tests. Two-tailed tests must include an equal = sign. One-tailed tests must start with < or >. If the terms in coef(object) uniquely identify estimates, they can be used in the formula. Otherwise, use b1, b2, etc. to identify the position of each parameter. The b* wildcard can be used to test hypotheses on all estimates. When the hypothesis string represents a two-sided equation, the estimate column holds the value of the left side minus the right side of the equation. If a named vector is used, the names are used as labels in the output. Examples:
  - hp = drat
  - hp + drat = 12
  - b1 + b2 + b3 = 0
  - b* / b1 = 1
  - <= 0
  - >= -3.5
  - b1 >= 10
- Formula: lhs ~ rhs | group
  - lhs
    - ratio (null = 1)
    - difference (null = 0)
    - Leave empty for default value
  - rhs
    - pairwise and revpairwise: 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.
    * `meandev`: difference between an estimate and the mean of all estimates.
    * `meanotherdev`: difference between an estimate and the mean of all other estimates, excluding the current one.
    * `poly`: polynomial contrasts, as computed by the `stats::contr.poly()` function.
    * `helmert`: Helmert contrasts, as computed by the `stats::contr.helmert()` function. Contrast 2nd level to the first, 3rd to the average of the first two, and so on.
    * `trt_vs_ctrl`: difference between the mean of estimates (except the first) and the first estimate.
    * `I(fun(x))`: custom function to manipulate the vector of estimates `x`. The function `fun()` can return multiple (potentially named) estimates.
  - `group` (optional)
    * Column name of `newdata`. Conduct hypothesis tests withing subsets of the data.
  - Examples:
    * `~ poly`
    * `~ sequential | groupid`
    * `~ reference`
    * `ratio ~ pairwise`
    * `difference ~ pairwise | groupid`
    * `~ I(x - mean(x)) | groupid`
    * `~ I(\(x) c(a = x[1], b = mean(x[2:3]))) | groupid`
- Matrix or Vector: Each column is a vector of weights. The the output is the dot product between these vectors of weights and the vector of estimates. The matrix can have column names to label the estimates.
- Function:
  - Accepts an argument `x`: object produced by a `marginaleffects` function or a data frame with column `rowid` and `estimate`
  - Returns a data frame with columns `term` and `estimate` (mandatory) and `rowid` (optional).
  - The function can also accept optional input arguments: `newdata`, `by`, `draws`.
  - This function approach will not work for Bayesian models or with bootstrapping. In those cases, it is easy to use `get_draws()` to extract and manipulate the draws directly.
- See the Examples section below and the vignette: https://marginaleffects.com/chapters/hypothesis.html
- Warning: When calling `predictions()` with `type="invlink(link)"` (the default in some models), `hypothesis` is tested and p values are computed on the link scale.

`equivalence` Numeric vector of length 2: bounds used for the two-one-sided test (TOST) of equivalence, and for the non-inferiority and non-superiority tests. For bayesian models, this report the proportion of posterior draws in the interval and the ROPE. See Details section below.

`df` Degrees of freedom used to compute p values and confidence intervals.

- A single numeric value between 1 and `Inf`, or a numeric vector with length equal to the number of rows in the output. When `df` is `Inf`, the normal distribution is used. When `df` is finite, the `t` distribution is used.
- "residual": Calls insight::get_df to extract degrees of freedom from the model automatically.
- "satterthwaite" or "kenward-roger": Use the Satterthwaite or Kenward-Roger approximation to compute degrees of freedom in mixed effects models.

`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.

`numderiv` string or list of strings indicating the method to use to for the numeric differentiation used in to compute delta method standard errors.

- "fdforward": finite difference method with forward differences (default)
- "fdcenter": finite difference method with central differences
- "richardson": Richardson extrapolation method
- Extra arguments can be specified by passing a list to the `numDeriv` argument, with the name of the method first and named arguments following, ex: `numderiv=list("fdcenter", eps = 1e-5)`. When an unknown argument is used, `marginaleffects` prints the list of valid arguments for each method.

`...` 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 `?slopes` documentation for a non-exhaustive list of available arguments.

Value

A `data.frame` with one row per estimate. This data frame is pretty-printed by default, but users can interact with it as a regular data frame, with functions like `nrow()`, `head()`, `colnames()`, etc. Values can be extracted using standard `[,]` or `$` operators, and manipulated using external packages like `dplyr` or `data.table`.

Columns may include:

- `rowid`: row number of the `newdata` data frame
- `group`: (optional) value of the grouped outcome (e.g., categorical outcome models)
- `term`: the focal variable.

- `estimate`: an estimate of the prediction, counterfactual comparison, or slope.
- `std.error`: standard errors computed via the delta method.
- `p.value`: p value associated to the `estimate` column. The null is determined by the `hypothesis` argument (0 by default).
- `s.value`: Shannon information transforms of p values. See the S values vignette at https://marginaleffects.com the marginaleffects website.
- `conf.low`: lower bound of the confidence (or credible) interval defined by the `conf_level` argument.
- `conf.high`: upper bound of the confidence (or credible) interval defined by the `conf_level` argument.
- `predicted_lo`: predicted outcome for the "low" value of the focal predictor in a counterfactual comparison.
- `predicted_hi`: predicted outcome for the "high" value of the focal predictor in a counterfactual comparison.
- `p.rope.unconditional`: share of posterior draws in the interval specified by the `equivalence` argument. This is only available for Bayesian models.
- `p.rope.conditional`: share of posterior draws in the interval specified by the `equivalence` argument, among draws in the confidence interval. This is only available for Bayesian models.
- `rope`: share of the posterior draws between `conf.low` and `conf.high` that are covered by the interval specified by the `equivalence` argument.
- `statistic.noninf`: test statistic for non-inferiority test (when `equivalence` argument is used).
- `statistic.nonsup`: test statistic for non-superiority test (when `equivalence` argument is used).
- `p.value.noninf`: p-value for non-inferiority test (when `equivalence` argument is used).
- `p.value.nonsup`: p-value for non-superiority test (when `equivalence` argument is used).
- `p.value.equiv`: p-value for equivalence test using Two One-Sided Tests (TOST) approach (when `equivalence` argument is used).

See ?`print.marginaleffects` for printing options.

The `data.frames` produced by `marginaleffects` stores an attribute that holds many internal objects, such as the original model, data, and much other information that can be used for post-processing. This information can be inspected using the `components()` function.

Warning: The internal attributes retrieved by `components()` are not considered part of the public API of the package. Their names and contents can change without warning or notice. Users should not rely on them.

Warning: In some cases, the internal attributes used by `marginaleffects()` can use up a substantial amount of memory. To clear this data, use the `prune()` function or set `options(marginaleffects_lean=TRUE)`.

Functions

- `avg_comparisons()`: Average comparisons

Standard errors using the delta method

Standard errors for all quantities estimated by `marginaleffects` can be obtained via the delta method. This requires differentiating a function with respect to the coefficients in the model using a finite difference approach. In some models, the delta method standard errors can be sensitive to various aspects of the numeric differentiation strategy, including the step size. By default, the step size is set to `1e-8`, or to `1e-4` times the smallest absolute model coefficient, whichever is largest.

`marginaleffects` can delegate numeric differentiation to the `numDeriv` package, which allows more flexibility. To do this, users can pass arguments to the `numDeriv::jacobian` function through a global option. For example:

- `options(marginaleffects_numDeriv = list(method = "simple", method.args = list(eps = 1e-6)))`
- `options(marginaleffects_numDeriv = list(method = "Richardson", method.args = list(eps = 1e-5)))`
- `options(marginaleffects_numDeriv = NULL)`

See the "Uncertainty" chapter on the `marginaleffects` website for more details on the computation of standard errors, bootstrapping, and more:

https://marginaleffects.com/chapters/uncertainty.html

Model-Specific Arguments

Some model types allow model-specific arguments to modify the nature of marginal effects, predictions, marginal means, and contrasts. Please report other package-specific `predict()` arguments on Github so we can add them to the table below.

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

Package	Class	Argument	Documentation
`brms`	`brmsfit`	`ndraws`	`brms::posterior_predict()`
		`re_formula`	`brms::posterior_predict()`
`lme4`	`merMod`	`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()`
	`gam`	`exclude`	`mgcv::predict.gam()`
`robustlmm`	`rlmerMod`	`re.form`	`robustlmm::predict.rlmerMod()`
		`allow.new.levels`	`robustlmm::predict.rlmerMod()`
`MCMCglmm`	`MCMCglmm`	`ndraws`	
`sampleSelection`	`selection`	`part`	`sampleSelection::predict.selection()`

Comparison functions

Each of the quantities computed by `comparisons()` can be defined as a function of these quantities:

- `hi`: vector of predictions for the "high" side of the contrast.

- `lo`: vector of predictions for the "low" side of the contrast.
- `y`: predictions for the original data.
- `x`: focal predictor in the original data.
- `w`: weights

For example, the "lift" of a binary predictor is a popular quantity of interest, defined as the difference between predictions when the focal predictor $X = 1$, and predictions when the focal predictor is $X = 0$, normalized by the starting point. Or:

$\frac{\hat{Y}_{X=1} - \hat{Y}_{X=0}}{\hat{Y}_{X=0}}$

When, the argument is set to `comparison="lift"`, `marginaleffects` will compute the quantity using this function:

```
function(hi, lo) { (hi - lo) / lo }
```

Users can supply custom functions to the `comparison` argument, or use one of the many shortcuts available for common quantities of interest:

Note on automatic `avg` switching: when the `by` argument is used (including `by = TRUE` or a character vector), `marginaleffects` will automatically switch comparison shortcuts to their `avg` variants unless they already end with `avg`. For example, `comparison = "difference"` becomes `comparison = "differenceavg"` when aggregating over `by`.

Shortcut	Function
difference	function (hi, lo) hi - lo
differenceavg	function (hi, lo) mean(hi - lo)
dydx	function (hi, lo, eps) (hi - lo)/eps
eyex	function (hi, lo, eps, y, x) (hi - lo)/eps * (x/y)
eydx	function (hi, lo, eps, y, x) ((hi - lo)/eps)/y
dyex	function (hi, lo, eps, x) ((hi - lo)/eps) * x
dydxavg	function (hi, lo, eps) mean((hi - lo)/eps)
eyexavg	function (hi, lo, eps, y, x) mean((hi - lo)/eps * (x/y))
eydxavg	function (hi, lo, eps, y, x) mean(((hi - lo)/eps)/y)
dyexavg	function (hi, lo, eps, x) mean(((hi - lo)/eps) * x)
ratio	function (hi, lo) hi/lo
ratioavg	function (hi, lo) mean(hi)/mean(lo)
lnratio	function (hi, lo) log(hi/lo)
lnratioavg	function (hi, lo) log(mean(hi)/mean(lo))
lnor	function (hi, lo) log((hi/(1 - hi))/(lo/(1 - lo)))
lnoravg	function (hi, lo) log((mean(hi)/(1 - mean(hi)))/(mean(lo)/(1 - mean(lo))))
lift	function (hi, lo) (hi - lo)/lo
liftavg	function (hi, lo) (mean(hi - lo))/mean(lo)
expdydx	function (hi, lo, eps) (exp(hi) - exp(lo))/eps
expdydxavg	function (hi, lo, eps) mean((exp(hi) - exp(lo))/eps)

Bayesian posterior summaries

By default, credible intervals in bayesian models are built as equal-tailed intervals. This can be changed to a highest density interval by setting a global option:

```
options("marginaleffects_posterior_interval" = "eti")
options("marginaleffects_posterior_interval" = "hdi")
```

By default, the center of the posterior distribution in bayesian models is identified by the median. Users can use a different summary function by setting a global option:

```
options("marginaleffects_posterior_center" = "mean")
options("marginaleffects_posterior_center" = "median")
```

When estimates are averaged using the by argument, the `tidy()` function, or the `summary()` function, the posterior distribution is marginalized twice over. First, we take the average *across* units but *within* each iteration of the MCMC chain, according to what the user requested in by argument or `tidy()/summary()` functions. Then, we identify the center of the resulting posterior using the function supplied to the `"marginaleffects_posterior_center"` option (the median by default).

Equivalence, Inferiority, Superiority

θ is an estimate, σ_θ its estimated standard error, and $[a, b]$ are the bounds of the interval supplied to the `equivalence` argument.

Non-inferiority:

- H_0: $\theta \leq a$
- H_1: $\theta > a$
- $t = (\theta - a)/\sigma_\theta$
- p: Upper-tail probability

Non-superiority:

- H_0: $\theta \geq b$
- H_1: $\theta < b$
- $t = (\theta - b)/\sigma_\theta$
- p: Lower-tail probability

Equivalence: Two One-Sided Tests (TOST)

- p: Maximum of the non-inferiority and non-superiority p values.

Thanks to Russell V. Lenth for the excellent `emmeans` package and documentation which inspired this feature.

Order of operations

Behind the scenes, the arguments of `marginaleffects` functions are evaluated in this order:

1. `newdata`
2. `variables`
3. `comparison` and `slope`
4. `by`
5. `vcov`
6. `hypothesis`
7. `transform`

Parallel computation

The `slopes()` and `comparisons()` functions can use parallelism to speed up computation. Operations are parallelized for the computation of standard errors, at the model coefficient level. There is always considerable overhead when using parallel computation, mainly involved in passing the whole dataset to the different processes. Thus, parallel computation is most likely to be useful when the model includes many parameters and the dataset is relatively small.

Warning: In many cases, parallel processing will not be useful at all.

To activate parallel computation, users must load the `future.apply` package, call `plan()` function, and set a global option.

`options(marginaleffects_parallel = TRUE)`: parallelize delta method computation of standard errors. `options(marginaleffects_parallel_inferences = TRUE)`: parallelize `"rsample"` or `"fwb"` bootstrap computation in `inferences()`. `options(marginaleffects_parallel_packages = TRUE)`: vector of strings with the names of modeling packages used to fit the model, ex: c("survival", "splines")

For example:

```
library(future.apply)
plan("multisession", workers = 4)
options(marginaleffects_parallel = FALSE)
options(marginaleffects_parallel_inferences = TRUE)
options(marginaleffects_parallel_packages = c("survival", "splines"))

slopes(model)
```

To disable parallelism in `marginaleffects` altogether, you can set a global option:

```
options(marginaleffects_parallel = FALSE)
```

Global options

The behavior of `marginaleffects` functions can be modified by setting global options.

Disable some safety checks and warnings:

- `options(marginaleffects_startup_message = FALSE)`
  - Disable the startup message printed on `library(marginaleffects)`.
- `options(marginaleffects_safe = FALSE)`
  - Disable safety checks and warnings.
- `options(marginaleffects_print_omit = c("p.value", "s.value"))`
  - Omit some columns from the printed output.

Enforce lean return objects, sans information about the original model and data, and other ancillary attributes. Note that this will disable some advanced post-processing features and functions like hypotheses.

```
options(marginaleffects_lean = TRUE)
```

Other options:

- `marginaleffects_plot_gray`: logical. If `TRUE`, the default color of the plot is gray. Default is `FALSE`.

Types

The `type` argument determines the scale of the predictions used to compute quantities of interest with functions from the `marginaleffects` package. Admissible values for `type` depend on the model object. When users specify an incorrect value for `type`, `marginaleffects` will raise an informative error with a list of valid `type` values for the specific model object. The first entry in the list in that error message is the default type.

The `invlink(link)` is a special type defined by `marginaleffects`. It is available for some (but not all) models, and only for the `predictions()` function. With this link type, we first compute predictions on the link scale, then we use the inverse link function to backtransform the predictions to the response scale. This is useful for models with non-linear link functions as it can ensure that confidence intervals stay within desirable bounds, ex: 0 to 1 for a logit model. Note that an average of estimates with `type="invlink(link)"` will not always be equivalent to the average of estimates with `type="response"`. This type is default when calling `predictions()`. It is available—but not default—when calling `avg_predictions()` or `predictions()` with the `by` argument.

Some of the most common `type` values are:

class	type
DirichletRegModel	response
Gam	invlink(link), response, link
Gls	lp
MCMCglmm	response
aft	surv, haz, cumhaz, density, odds, link
bam	response, link
bart	ev, ppd
betareg	response, link, precision, quantile, variance
bife	response, link
bracl	probs
brglmFit	response, link
brmsfit	response, link, prediction, average
brmultinom	probs, class
clm	prob, cum.prob, linear.predictor
clmm2	prob
clogit	expected, lp, risk, survival
coxph	survival, expected, lp, risk
coxph_weightit	survival, expected, lp, risk
crch	response, location, scale, density
fixest	invlink(link), response, link
flexsurvreg	survival, response, mean, link, lp, linear, rmst, hazard, cumhaz
gam	response, link
geeglm	response, link
glimML	response, link
glm	invlink(link), response, link

glm_weightit	invlink(link), probs, response, lp, link
glmerMod	response, link
glmgee	response, link
glmmPQL	response, link
glmmTMB	response, link, conditional, zprob, zlink, disp
glmrob	response, link
glmx	response
gnm	response, link
hetprob	pr, xb
hurdle	response, prob, count, zero
hxlr	location, cumprob, scale, density
iv_robust	response
ivpml	pr, xb
ivreg	response
lda	class, posterior
lm	response
lm_robust	response
lmerMod	response
lmerModLmerTest	response
lmrob	response
lrm	fitted, lp, mean
mblogit	response, latent, link
mclogit	response, latent, link
mhurdle	E, Ep, p
model_fit	numeric, prob, class
multinom	probs, latent
multinom_weightit	probs, response, mean
mvgam	response, link, expected, detection, latent_N
negbin	invlink(link), response, link
nestedLogit	response, link
ols	lp
oohbchoice	probability, utility
ordinal_weightit	probs, response, link, lp, mean
orm	fitted, mean, lp
polr	probs
rendo.base	response, link
rlm	response
selection	response, link, unconditional, conditional
speedglm	response, link
speedlm	response
stanreg	response, link
survreg	response, link, quantile
svy_vglm	response, link
svyglm	response, link
svyolr	probs
tobit	response, link
tobit1	expvalue, linpred, prob
workflow	numeric, prob, class

zeroinfl	response, prob, count, zero

References

- Arel-Bundock V, Greifer N, Heiss A (2024). "How to Interpret Statistical Models Using marginaleffects for R and Python." *Journal of Statistical Software*, *111*(9), 1-32. doi:10.18637/jss.v111.i09 doi:10.18637/jss.v111.i09
- Arel-Bundock (2026). "Model to Meaning: How to interpret statistical models in R and Python." CRC Press. https://routledge.com/9781032908724
- Greenland S. 2019. "Valid P-Values Behave Exactly as They Should: Some Misleading Criticisms of P-Values and Their Resolution With S-Values." The American Statistician. 73(S1): 106–114.
- Cole, Stephen R, Jessie K Edwards, and Sander Greenland. 2020. "Surprise!" American Journal of Epidemiology 190 (2): 191–93. doi:10.1093/aje/kwaa136
- Hansen SN, Overgaard M (2025). "Variance Estimation for Average Treatment Effects Estimated by G-Computation." *Metrika*, *88*(4), 419–443. doi:10.1007/s00184024009624

Examples

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

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

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

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

# 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)
avg_comparisons(mod, variables = list(hp = 1))
avg_comparisons(mod, variables = list(hp = 5))
```

```
avg_comparisons(mod, variables = list(hp = c(90, 100)))
avg_comparisons(mod, variables = list(hp = "iqr"))
avg_comparisons(mod, variables = list(hp = "sd"))
avg_comparisons(mod, variables = list(hp = "minmax"))

# 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 <- function(x) data.frame(x, x + 10)
avg_comparisons(modlog, variables = list(new_hp = fdiff))

# Adjusted Risk Ratio
mod <- glm(vs ~ mpg, data = mtcars, family = binomial)
avg_comparisons(mod, comparison = "lnratioavg", transform = exp)

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

# variable-specific contrasts
avg_comparisons(mod, variables = list(gear = "sequential", hp = 10))

# 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)

# Effect of a 1 group-wise standard deviation change
# First we calculate the SD in each group of `cyl`
# Second, we use that SD as the treatment size in the `variables` argument
library(dplyr)
mod <- lm(mpg ~ hp + factor(cyl), mtcars)
tmp <- mtcars %>%
  group_by(cyl) %>%
  mutate(hp_sd = sd(hp))
avg_comparisons(mod,
  variables = list(hp = function(x) data.frame(x, x + tmp$hp_sd)),
  by = "cyl")

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

mod <- lm(mpg ~ hp * am * vs, data = mtcars)
avg_comparisons(mod, variables = "hp", by = c("vs", "am"))

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)
```

`components.marginaleffects` *Extract components from marginaleffects objects*

Description

Extract components from marginaleffects objects

Usage

```
## S3 method for class 'marginaleffects'
components(object, component = NULL, ...)
```

Arguments

`object`	A marginaleffects object (predictions, comparisons, slopes, or hypotheses)

component	Character string specifying which component to extract. Must be a valid slot name from the internal S4 object. If NULL (the default), components() prints a message with all available component names. Common components include: "model", "newdata", "modeldata", "call", "jacobian", "vcov_model", "type", "by", "comparison", "variables", etc.
...	Ignored.

Details

This function provides access to the internal components stored in the mfx attribute of marginaleffects objects. The mfx attribute contains an S4 object of class "marginaleffects_internal" with various slots containing model information, data, and computational details used by the marginaleffects functions.

Warning: the internal slot names are not considered part of the public API and may change without warning in future versions of the marginaleffects package.

Value

The requested component from the mfx object

---

datagrid *Data grids*

Description

Generate a data grid of user-specified values for use in the newdata argument of the predictions(), comparisons(), and slopes() functions. This is useful to define where in the predictor space we want to evaluate the quantities of interest. Ex: the predicted outcome or slope for a 37 year old college graduate.

Usage

```
datagrid(
  ...,
  model = NULL,
  newdata = NULL,
  by = NULL,
  grid_type = "mean_or_mode",
  response = FALSE,
  FUN = NULL,
  FUN_character = NULL,
  FUN_factor = NULL,
  FUN_logical = NULL,
  FUN_numeric = NULL,
  FUN_integer = NULL,
  FUN_binary = NULL,
  FUN_other = 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 can be used.)
`by`	character vector with grouping variables within which `FUN_*` functions are applied to create "sub-grids" with unspecified variables.
`grid_type`	character. Determines the functions to apply to each variable. The defaults can be overridden by defining individual variables explicitly in `...`, or by supplying a function to one of the `FUN_*` arguments. • "mean_or_mode": Character, factor, logical, and binary variables are set to their modes. Numeric, integer, and other variables are set to their means. • "balanced": Each unique level of character, factor, logical, and binary variables are preserved. Numeric, integer, and other variables are set to their means. Warning: When there are many variables and many levels per variable, a balanced grid can be very large. In those cases, it is better to use `grid_type="mean_or_mode"` and to specify the unique levels of a subset of named variables explicitly. • "dataframe": Similar to "mean_or_mode" but creates a data frame by binding columns element-wise rather than taking the cross-product. All explicitly specified vectors must have the same length (or length 1), and the result has as many rows as the longest vector. This differs from other grid types which use `expand.grid()` or `data.table::CJ()` to create all combinations. • "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.
`response`	Logical should the response variable be included in the grid, even if it is not specified explicitly.
`FUN`	a function to be applied to all variables in the grid. This is useful when you want to apply the same function to all variables, such as `mean` or `median`. If you specify `FUN`, it will override the `grid_type` defaults, but not other `FUN_*` arguments below.
`FUN_character`	the function to be applied to character variables.
`FUN_factor`	the function to be applied to factor variables, including variables converted to factor in the model-fitting formula (e.g., `factor(cyl)`).
`FUN_logical`	the function to be applied to logical variables.
`FUN_numeric`	the function to be applied to numeric variables.

FUN_integer	the function to be applied to integer-ish variables (including columns without decimal places).
FUN_binary	the function to be applied to binary variables.
FUN_other	the function to be applied to other variable types.

Details

If datagrid is used in a predictions(), comparisons(), or slopes() call as the newdata argument, the model is automatically inserted in the model argument of datagrid() call, and users do not need to specify either the model or newdata arguments. The same behavior will occur when the value supplied to newdata= is a function call which starts with "datagrid". This is intended to allow users to create convenience shortcuts like:

Warning about hierarchical grouping variables: When using the default grid_type = "mean_or_mode" with hierarchical models (such as mixed models with nested grouping factors), datagrid() may create invalid combinations of grouping variables. For example, if you have students nested within schools, or countries nested within regions, the modal values of each grouping variable may not correspond to valid nested relationships in the data. This can cause prediction errors. To avoid this issue, explicitly specify valid combinations of hierarchical grouping variables in the datagrid() call, or use grid_type = "counterfactual" to preserve the original data structure.

```
mod <- lm(mpg ~ am + vs + factor(cyl) + hp, mtcars)
datagrid_bal <- function(...) datagrid(..., grid_type = "balanced")
predictions(model, newdata = datagrid_bal(cyl = 4))
```

If users supply a model, the data used to fit that model is retrieved using get_modeldata(). Consider using set_modeldata() to attach the training data explicitly when working in non-standard environments (e.g., lapply(), Shiny, nested functions).

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.

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 `slopes()` or `predictions()` we do not need to specify the
# `model` or `newdata` arguments.
slopes(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)

# Use `by` to hold variables at group-specific values
mod2 <- lm(mpg ~ hp + cyl, mtcars)
datagrid(model = mod2, hp = mean, by = "cyl")

# Use `FUN` to apply function to all variables
datagrid(model = mod2, FUN = median)

# Use `grid_type="dataframe"` for column-wise binding instead of cross-product
datagrid(model = mod2, hp = c(100, 200), cyl = c(4, 6), grid_type = "dataframe")
```

`get_dataset` *Download and Read Datasets from* `marginaleffects` *or Rdatasets*

Description

Downloads a dataset from the `marginaleffects` or the Rdatasets archives, and return it as a data frame. Opens the documentation as an HTML page. Search available datasets.

https://vincentarelbundock.github.io/Rdatasets/

Usage

```
get_dataset(dataset = "thornton", package = NULL, docs = FALSE, search = NULL)
```

Arguments

`dataset`	String. Name of the dataset to download. • `marginaleffects` archive: `affairs`, `airbnb`, `ces_demographics`, `ces_survey`, `immigration`, `lottery`, `military`, `thornton`, `factorial_01`, `interaction_01`, `interaction_02`, `interaction_03`, `interaction_04`, `polynomial_01`, `polynomial_02` • Rdatasets archive: The name of a dataset listed on the Rdatasets index. See the website or the `search` argument.
`package`	String. Package name that originally published the data.

docs	Logical. If TRUE open the documentation using `getOption("viewer")` or the Rstudio viewer.
search	Regular expression. Download the dataset index from Rdatasets; search the "Package", "Item", and "Title" columns; and return the matching rows.

Value

A data frame containing the dataset.

Examples

```
## Not run:
dat <- get_dataset("Titanic", "Stat2Data")
head(dat)

get_dataset(search = "(?i)titanic")

# View documentation in the browser
get_dataset("Titanic", "Stat2Data", docs = TRUE)

## End(Not run)
```

---

get_draws *Extract Posterior Draws or Bootstrap Resamples from* `marginaleffects` *Objects*

Description

Extract Posterior Draws or Bootstrap Resamples from `marginaleffects` Objects

Usage

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

Arguments

x	An object produced by a `marginaleffects` package function, such as `predictions()`, `avg_slopes()`, `hypotheses()`, etc.
shape	string indicating the shape of the output format: • "long" (default): long format data frame • "PxD" (fastest): Matrix with parameters as rows and draws as columns • "DxP": Matrix with draws as rows and parameters as columns • "rvar": Random variable datatype (see `posterior` package documentation).

Details

If DxP and PxD and the names returned by `coef(x)` are unique, `marginaleffects` sets parameter names to those names. Otherwise, it sets them to `b1`, `b2`, etc.

Value

A data.frame with `drawid` and `draw` columns.

---

get_modeldata *Get or set the dataset used to fit a model*

Description

`get_modeldata()` retrieves the data used to fit a model. If the data was previously attached via `set_modeldata()`, it is returned directly. Otherwise, the data is retrieved from the model object or the environment using `insight::get_data()`.

Usage

```
get_modeldata(model, ...)

set_modeldata(model, newdata)
```

Arguments

`model`	Model object
`...`	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 `?slopes` documentation for a non-exhaustive list of available arguments.
`newdata`	A data frame to attach to the model object.

Details

`set_modeldata()` attaches a dataset to a model object as an attribute. This is useful when the training data may not be available in the environment where `marginaleffects` functions are called, such as inside `lapply()`, in Shiny apps, or in nested function calls. By attaching the data explicitly, you ensure that `marginaleffects` always uses the correct dataset.

Value

`get_modeldata()` returns a data frame of the original data used to fit the model, or `NULL` if not available. `set_modeldata()` returns the model with the data attached as an attribute.

hypotheses *(Non-)Linear Tests for Null Hypotheses, Joint Hypotheses, Equivalence, Non Superiority, and Non Inferiority*

Description

Uncertainty estimates are calculated as first-order approximate standard errors for linear or non-linear functions of a vector of random variables with known or estimated covariance matrix. In that sense, hypotheses emulates the behavior of the excellent and well-established `car::deltaMethod()` and `car::linearHypothesis()` functions, but it supports more models; requires fewer dependencies; expands the range of tests to equivalence and superiority/inferiority; and offers convenience features like robust standard errors.

To learn more, read the hypothesis tests vignette, visit the package website:

- https://marginaleffects.com/chapters/hypothesis.html
- https://marginaleffects.com/

Warning #1: Tests are conducted directly on the scale defined by the `type` argument. For some models, it can make sense to conduct hypothesis or equivalence tests on the `"link"` scale instead of the `"response"` scale which is often the default.

Warning #2: For hypothesis tests on objects produced by the `marginaleffects` package, it is safer to use the `hypothesis` argument of the original function. Using `hypotheses()` may not work in certain environments, in lists, or when working programmatically with *apply style functions.

Warning #3: The tests assume that the `hypothesis` expression is (approximately) normally distributed, which for non-linear functions of the parameters may not be realistic. More reliable confidence intervals can be obtained using the `inferences()` function with `method = "boot"`.

Usage

```
hypotheses(
  model = NULL,
  hypothesis = NULL,
  vcov = TRUE,
  conf_level = NULL,
  df = NULL,
  equivalence = NULL,
  joint = FALSE,
  joint_test = "f",
  multcomp = FALSE,
  numderiv = "fdforward",
  ...
)
```

Arguments

`model`	Model object or object generated by the `comparisons()`, `slopes()`, or `predictions()` functions.

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

- Number: The null hypothesis used in the computation of Z and p (before applying `transform`).
- String: Equation to specify linear or non-linear hypothesis tests. Two-tailed tests must include an equal = sign. One-tailed tests must start with < or >. If the terms in `coef(object)` uniquely identify estimates, they can be used in the formula. Otherwise, use `b1`, `b2`, etc. to identify the position of each parameter. The `b*` wildcard can be used to test hypotheses on all estimates. When the hypothesis string represents a two-sided equation, the `estimate` column holds the value of the left side minus the right side of the equation. If a named vector is used, the names are used as labels in the output. Examples:
  - `hp = drat`
  - `hp + drat = 12`
  - `b1 + b2 + b3 = 0`
  - `b* / b1 = 1`
  - `<= 0`
  - `>= -3.5`
  - `b1 >= 10`
- Formula: `lhs ~ rhs | group`
  - `lhs`
    - `ratio` (null = 1)
    - `difference` (null = 0)
    - Leave empty for default value
  - `rhs`
    - `pairwise` and `revpairwise`: 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.
    - `meandev`: difference between an estimate and the mean of all estimates.
    - `meanotherdev`: difference between an estimate and the mean of all other estimates, excluding the current one.
    - `poly`: polynomial contrasts, as computed by the `stats::contr.poly()` function.
    - `helmert`: Helmert contrasts, as computed by the `stats::contr.helmert()` function. Contrast 2nd level to the first, 3rd to the average of the first two, and so on.
    - `trt_vs_ctrl`: difference between the mean of estimates (except the first) and the first estimate.
    - `I(fun(x))`: custom function to manipulate the vector of estimates `x`. The function `fun()` can return multiple (potentially named) estimates.

  - group (optional)
    * Column name of newdata. Conduct hypothesis tests withing subsets of the data.
  - Examples:
    * ~ poly
    * ~ sequential | groupid
    * ~ reference
    * ratio ~ pairwise
    * difference ~ pairwise | groupid
    * ~ I(x - mean(x)) | groupid
    * ~ I(\(x) c(a = x[1], b = mean(x[2:3]))) | groupid
- Matrix or Vector: Each column is a vector of weights. The the output is the dot product between these vectors of weights and the vector of estimates. The matrix can have column names to label the estimates.
- Function:
  - Accepts an argument x: object produced by a marginaleffects function or a data frame with column rowid and estimate
  - Returns a data frame with columns term and estimate (mandatory) and rowid (optional).
  - The function can also accept optional input arguments: newdata, by, draws.
  - This function approach will not work for Bayesian models or with bootstrapping. In those cases, it is easy to use get_draws() to extract and manipulate the draws directly.
- See the Examples section below and the vignette: https://marginaleffects.com/chapters/hypothesis.html
- Warning: When calling predictions() with type="invlink(link)" (the default in some models), hypothesis is tested and p values are computed on the link scale.

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"
  - Unconditional: "unconditional" accounts for sampling variation in the empirical covariate distribution for averaged or aggregated predictions, comparisons, and slopes. Hypotheses applied directly to unit-level effects are rejected. Use vcovUnconditional(cluster = ~cluster) for one-way clustered unconditional inference.

  - Mixed-Models degrees of freedom: "satterthwaite", "kenward-roger"
  - Other: `"NeweyWest"`, `"KernHAC"`, `"OPG"`. See the `sandwich` package documentation.
  - `"rsample"`, `"boot"`, `"fwb"`, or `"simulation"`: forward the result to `inferences()` using that method.
- 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` NULL or numeric value between 0 and 1. Confidence level to use to build a confidence interval. When `NULL` and `model` was generated by `marginaleffects`, the confidence level is taken from the `conf_level` attribute of the model. Otherwise, the default value is 0.95.

`df` Degrees of freedom used to compute p values and confidence intervals.

- A single numeric value between 1 and `Inf`, or a numeric vector with length equal to the number of rows in the output. When `df` is `Inf`, the normal distribution is used. When `df` is finite, the `t` distribution is used.
- "residual": Calls insight::get_df to extract degrees of freedom from the model automatically.
- When using `joint_test="f"`, the `df` argument should be a numeric vector of length 2.

`equivalence` Numeric vector of length 2: bounds used for the two-one-sided test (TOST) of equivalence, and for the non-inferiority and non-superiority tests. For bayesian models, this report the proportion of posterior draws in the interval and the ROPE. See Details section below.

`joint` Joint test of statistical significance. The null hypothesis value can be set using the `hypothesis` argument.

- FALSE: Hypotheses are not tested jointly.
- TRUE: All parameters are tested jointly.
- String: A regular expression to match parameters to be tested jointly. `grep(joint, perl = TRUE)`
- Character vector of parameter names to be tested. Characters refer to the names of the vector returned by `marginaleffects::get_coef(object)`.
- Integer vector of indices. Which parameters positions to test jointly.
- Note: When using the `joint` argument, the `hypothesis` argument is limited to `NULL` or numeric values. Users can chain multiple `hypotheses()` for complex joint hypothesis tests.

`joint_test` A character string specifying the type of test, either "f" or "chisq". The null hypothesis is set by the `hypothesis` argument, with default null equal to 0 for all parameters.

`multcomp` Logical or string. If `TRUE` or string, apply multiple comparison adjustment to the p values and report family-wise confidence intervals. Valid strings: "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr", "single-step", "Shaffer", "Westfall", "free". When `multcomp` is `TRUE`, the "holm" method is used.

`numderiv` string or list of strings indicating the method to use to for the numeric differentiation used in to compute delta method standard errors.

- "fdforward": finite difference method with forward differences (default)
- "fdcenter": finite difference method with central differences
- "richardson": Richardson extrapolation method
- Extra arguments can be specified by passing a list to the `numDeriv` argument, with the name of the method first and named arguments following, ex: `numderiv=list("fdcenter", eps = 1e-5)`. When an unknown argument is used, `marginaleffects` prints the list of valid arguments for each method.

`...` 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 `?slopes` documentation for a non-exhaustive list of available arguments.

Joint hypothesis tests

The test statistic for the joint Wald test is calculated as (R * theta_hat - r)' * inv(R * V_hat * R') * (R * theta_hat - r) / Q, where theta_hat is the vector of estimated parameters, V_hat is the estimated covariance matrix, R is a Q x P matrix for testing Q hypotheses on P parameters, r is a Q x 1 vector for the null hypothesis, and Q is the number of rows in R. If the test is a Chi-squared test, the test statistic is not normalized.

The p-value is then calculated based on either the F-distribution (for F-test) or the Chi-squared distribution (for Chi-squared test). For the F-test, the degrees of freedom are Q and (n - P), where n is the sample size and P is the number of parameters. For the Chi-squared test, the degrees of freedom are Q.

Equivalence, Inferiority, Superiority

θ is an estimate, σ_θ its estimated standard error, and $[a, b]$ are the bounds of the interval supplied to the `equivalence` argument.

Non-inferiority:

- H_0: $\theta \leq a$
- H_1: $\theta > a$
- $t = (\theta - a)/\sigma_\theta$
- p: Upper-tail probability

Non-superiority:

- H_0: $\theta \geq b$
- H_1: $\theta < b$
- $t = (\theta - b)/\sigma_\theta$
- p: Lower-tail probability

Equivalence: Two One-Sided Tests (TOST)

- p: Maximum of the non-inferiority and non-superiority p values.

Thanks to Russell V. Lenth for the excellent emmeans package and documentation which inspired this feature.

Examples

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

hypotheses(mod)

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

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

# Robust standard errors
hypotheses(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
hypotheses(mod, hypothesis = "b2 = b3")

# wildcard
hypotheses(mod, hypothesis = "b* / b2 = 1")

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

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

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

mfx <- slopes(mod, newdata = "mean")
hypotheses(mfx, hypothesis = "b2 = 0.2")

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

# The `hypothesis` 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 <- hypotheses(mod, hypothesis = f)
head(p)
```

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

# Complex aggregation
# Step 1: Collapse predicted probabilities by outcome level, for each individual
# Step 2: Take the mean of the collapsed probabilities by group and `cyl`
library(dplyr)
library(MASS)
library(dplyr)
library(magrittr)

dat <- transform(mtcars, gear = factor(gear))
mod <- polr(gear ~ factor(cyl) + hp, dat)

aggregation_fun <- function(x) {
    predictions(x, vcov = FALSE) %>%
        mutate(group = ifelse(group %in% c("3", "4"), "3 & 4", "5")) %>%
        summarize(estimate = sum(estimate), .by = c("rowid", "cyl", "group")) %>%
        summarize(estimate = mean(estimate), .by = c("cyl", "group")) %>%
        rename(term = cyl)
}

hypotheses(mod, hypothesis = aggregation_fun)

# Equivalence, non-inferiority, and non-superiority tests
mod <- lm(mpg ~ hp + factor(gear), data = mtcars)
p <- predictions(mod, newdata = "median")
hypotheses(p, equivalence = c(17, 18))

mfx <- avg_slopes(mod, variables = "hp")
hypotheses(mfx, equivalence = c(-.1, .1))

cmp <- avg_comparisons(mod, variables = "gear", hypothesis = ~pairwise)
hypotheses(cmp, equivalence = c(0, 10))

# joint hypotheses: character vector
model <- lm(mpg ~ as.factor(cyl) * hp, data = mtcars)
hypotheses(model, joint = c("as.factor(cyl)6:hp", "as.factor(cyl)8:hp"))

# joint hypotheses: regular expression
hypotheses(model, joint = "cyl")

# joint hypotheses: integer indices
hypotheses(model, joint = 2:3)

# joint hypotheses: different null hypotheses
hypotheses(model, joint = 2:3, hypothesis = 1)
hypotheses(model, joint = 2:3, hypothesis = 1:2)

# joint hypotheses: marginaleffects object
cmp <- avg_comparisons(model)
hypotheses(cmp, joint = "cyl")
```

```
# Multiple comparison adjustment
# p values and family-wise confidence intervals
cmp <- avg_comparisons(model)
hypotheses(cmp, multcomp = "hochberg")
```

`inferences` *(EXPERIMENTAL) Bootstrap, Conformal, and Simulation-Based Inference*

Description

Warning: This function is experimental. It may be renamed, the user interface may change, or the functionality may migrate to arguments in other `marginaleffects` functions.

Apply this function to a `marginaleffects` object to change the inferential method used to compute uncertainty estimates.

Usage

```
inferences(
  x,
  method,
  R = 1000,
  conf_type = "perc",
  data_train = NULL,
  data_test = NULL,
  data_calib = NULL,
  conformal_score = "residual_abs",
  estimator = NULL,
  ...
)
```

Arguments

`x`	Object produced by one of the core `marginaleffects` functions, or a data frame suitable for the function supplied to the `estimator` argument.
`method`	String • "delta": delta method standard errors • "boot" package • "fwb": fractional weighted bootstrap • "rsample" package • "simulation" from a multivariate normal distribution (Krinsky & Robb, 1986) • "conformal_split": prediction intervals using split conformal prediction (see Angelopoulos & Bates, 2022)

- "conformal_cv+": prediction intervals using cross-validation+ conformal prediction (see Barber et al., 2020)
- "conformal_full": prediction intervals using full conformal prediction (see Lei et al., 2018). **Warning**: This method is computationally expensive and typically much slower than split or CV+ methods.
- "conformal_quantile": prediction intervals using full conformal prediction (see Romano et al., 2020).

`R` Number of resamples, simulations, or cross-validation folds.

`conf_type` String: type of bootstrap interval to construct.

- `boot`: "perc", "norm", "basic", or "bca"
- `fwb`: "perc", "norm", "wald", "basic", "bc", or "bca"
- `rsample`: "perc" or "bca"
- `simulation`: "perc" or "wald"

`data_train` Data frame used to train/fit the model. If `NULL`, `marginaleffects` tries to extract the data from the original model object. Test data are inferred directly from the `newdata` supplied to the originating `marginaleffects` call (e.g., `predictions()`).

`data_test` Data frame make out of sample prediction. Only used for conformal inference. If `NULL`, the data frame supplied to `newdata` in the original `marginaleffects` call is used.

`data_calib` Data frame used for calibration in split conformal prediction.

`conformal_score`

String. Warning: The `type` argument in `predictions()` must generate predictions which are on the same scale as the outcome variable. Typically, this means that `type` must be "response" or "probs".

- "residual_abs" or "residual_sq" for regression tasks (numeric outcome)
- "softmax" for classification tasks (when `predictions()` returns a `group` columns, such as multinomial or ordinal logit models.

`estimator` Function that accepts a data frame, fits a model, applies a `marginaleffects` function, and returns the object. Only supported with `method = "rsample"` or `method = "boot"`. When `method = "rsample"`, the output must include a "term" column. This is not always the case for `predictions()`, in which case users may have to create the column manually.

`...`

- If `method = "boot"`, additional arguments are passed to `boot::boot()`.
- If `method = "fwb"`, additional arguments are passed to `fwb::fwb()`.
- If `method = "rsample"`, additional arguments are passed to `rsample::bootstraps()`, unless the user supplies a `group` argument, in which case all arguments are passed to `rsample::group_bootstraps()`.
- If `method = "conformal_full"`, additional arguments control the optimization process:
  - `var_multiplier`: multiplier for initial search bounds (default: 10)
  - `max_iter`: maximum iterations for root finding (default: 100)
  - `tolerance`: tolerance for root finding convergence (default: `.Machine$double.eps^0.25`)
- If `method = "conformal_quantile"`, additional arguments are passed to `quantregForest::quantregForest()` for fitting the quantile regression forest (e.g., `ntree`, `mtry`, `nodesize`, `nthreads`).

- Additional arguments are ignored for other conformal methods (`conformal_split`, `conformal_cv+`).

Details

When `method = "simulation"`, we conduct simulation-based inference following the method discussed in Krinsky & Robb (1986):

1. Draw `R` sets of simulated coefficients from a multivariate normal distribution with mean equal to the original model's estimated coefficients and variance equal to the model's variance-covariance matrix (classical, "HC3", or other).
2. Use the `R` sets of coefficients to compute `R` sets of estimands: predictions, comparisons, slopes, or hypotheses.
3. Take quantiles of the resulting distribution of estimands to obtain a confidence interval (when `conf_type = "perc"`) and the standard deviation of simulated estimates to estimate the standard error (which is used for a Z-test and Wald confidence intervals when `conf_type = "wald"`).

When `method = "fwb"`, drawn weights are supplied to the model fitting function's `weights` argument; if the model doesn't accept non-integer weights, this method should not be used. If weights were included in the original model fit, they are extracted by `weights()` and multiplied by the drawn weights. These weights are supplied to the `wts` argument of the estimation function (e.g., `comparisons()`).

Warning: custom model classes are not supported by `inferences()` because they are not guaranteed to come with an appropriate `update()` method.

Value

A `marginaleffects` object with simulation or bootstrap resamples and objects attached.

References

Krinsky, I., and A. L. Robb. 1986. "On Approximating the Statistical Properties of Elasticities." Review of Economics and Statistics 68 (4): 715–9.

King, Gary, Michael Tomz, and Jason Wittenberg. "Making the most of statistical analyses: Improving interpretation and presentation." American journal of political science (2000): 347-361

Dowd, Bryan E., William H. Greene, and Edward C. Norton. "Computation of standard errors." Health services research 49.2 (2014): 731-750.

Angelopoulos, Anastasios N., and Stephen Bates. 2022. "A Gentle Introduction to Conformal Prediction and Distribution-Free Uncertainty Quantification." arXiv. https://doi.org/10.48550/arXiv.2107.07511.

Barber, Rina Foygel, Emmanuel J. Candes, Aaditya Ramdas, and Ryan J. Tibshirani. 2020. "Predictive Inference with the Jackknife+." arXiv. http://arxiv.org/abs/1905.02928.

Lei, Jing, Max G'Sell, Alessandro Rinaldo, Ryan J. Tibshirani, and Larry Wasserman. 2018. "Distribution-Free Predictive Inference for Regression." Journal of the American Statistical Association 113 (523): 1094–1111.

Romano, Yaniv, Evan Patterson, and Emmanuel Candes. 2020. "Conformalized quantile regression." Advances in neural information processing systems 32.

Parallel computation

The `slopes()` and `comparisons()` functions can use parallelism to speed up computation. Operations are parallelized for the computation of standard errors, at the model coefficient level. There is always considerable overhead when using parallel computation, mainly involved in passing the whole dataset to the different processes. Thus, parallel computation is most likely to be useful when the model includes many parameters and the dataset is relatively small.

Warning: In many cases, parallel processing will not be useful at all.

To activate parallel computation, users must load the `future.apply` package, call `plan()` function, and set a global option.

`options(marginaleffects_parallel = TRUE)`: parallelize delta method computation of standard errors. `options(marginaleffects_parallel_inferences = TRUE)`: parallelize `"rsample"` or `"fwb"` bootstrap computation in `inferences()`. `options(marginaleffects_parallel_packages = TRUE)`: vector of strings with the names of modeling packages used to fit the model, ex: c("survival", "splines")

For example:

```
library(future.apply)
plan("multisession", workers = 4)
options(marginaleffects_parallel = FALSE)
options(marginaleffects_parallel_inferences = TRUE)
options(marginaleffects_parallel_packages = c("survival", "splines"))

slopes(model)
```

To disable parallelism in `marginaleffects` altogether, you can set a global option:

```
options(marginaleffects_parallel = FALSE)
```

Examples

```
## Not run:
library(magrittr)
set.seed(1024)
mod <- lm(Sepal.Length ~ Sepal.Width * Species, data = iris)

# bootstrap
avg_predictions(mod, by = "Species") %>%
    inferences(method = "boot")

avg_predictions(mod, by = "Species") %>%
    inferences(method = "rsample")

# Fractional (bayesian) bootstrap
avg_slopes(mod, by = "Species") %>%
    inferences(method = "fwb") %>%
    get_draws("rvar") %>%
    data.frame()
```

```
# Simulation-based inference
slopes(mod) %>%
    inferences(method = "simulation") %>%
    head()

# Two-step estimation procedure: Propensity score + G-Computation
lalonde <- get_dataset("lalonde")
estimator <- function(data) {
    # Step 1: Estimate propensity scores
    fit1 <- glm(treat ~ age + educ + race, family = binomial, data = data)
    ps <- predict(fit1, type = "response")
    # Step 2: Fit weighted outcome model
    m <- lm(re78 ~ treat * (re75 + age + educ + race),
        data = data, weight = ps
    )
    # Step 3: Compute average treatment effect by G-computation
    avg_comparisons(m, variables = "treat", wts = ps, vcov = FALSE)
}
inferences(lalonde, method = "rsample", estimator = estimator)

## End(Not run)
```

`plot_comparisons` *Plot Conditional or Marginal Comparisons*

Description

Plot comparisons on the y-axis against values of one or more predictors (x-axis, colors/shapes, and facets).

The `by` argument is used to plot marginal comparisons, that is, comparisons made on the original data, but averaged by subgroups. This is analogous to using the `by` argument in the `comparisons()` function.

The `condition` argument is used to plot conditional comparisons, that is, comparisons made on a user-specified grid. This is analogous to using the `newdata` argument and `datagrid()` function in a `comparisons()` call. All variables whose values are not specified explicitly are treated as usual by `datagrid()`, that is, they are held at their mean or mode (or rounded mean for integers). This includes grouping variables in mixed-effects models, so analysts who fit such models may want to specify the groups of interest using the `condition` argument, or supply model-specific arguments to compute population-level estimates. See details below.

See the "Plots" vignette and website for tutorials and information on how to customize plots:

- https://marginaleffects.com/bonus/plot.html
- https://marginaleffects.com

Usage

```
plot_comparisons(
  model,
  variables = NULL,
  condition = NULL,
  by = NULL,
  newdata = NULL,
  type = NULL,
  vcov = NULL,
  conf_level = 0.95,
  wts = FALSE,
  comparison = "difference",
  transform = NULL,
  rug = FALSE,
  gray = getOption("marginaleffects_plot_gray", default = FALSE),
  draw = TRUE,
  ...
)
```

Arguments

`model`	Model object
`variables`	Name of the variable whose contrast we want to plot on the y-axis.
`condition`	Conditional slopes • Character vector (max length 4): Names of the predictors to display. • Named list (max length 4): List names correspond to predictors. List elements can be: – Numeric vector – Function which returns a numeric vector or a set of unique categorical values – Shortcut strings for common reference values: "minmax", "quartile", "threenum" • 1: x-axis. 2: color/shape. 3: facet (wrap if no fourth variable, otherwise cols of grid). 4: facet (rows of grid). • Numeric variables in positions 2 and 3 are summarized by Tukey's five numbers `?stats::fivenum`.
`by`	Aggregate unit-level estimates (aka, marginalize, average over). Valid inputs: • `FALSE`: return the original unit-level estimates. • `TRUE`: aggregate estimates for each term. • 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.

- For more complex aggregations, you can use the FUN argument of the hypotheses() function. See that function's documentation and the Hypothesis Test vignettes on the marginaleffects website.

newdata When newdata is NULL, the grid is determined by the condition argument. When newdata is not NULL, the argument behaves in the same way as in the predictions() function. Note that the condition argument builds its own grid, so the newdata argument is ignored if the condition argument is supplied.

type string indicates the type (scale) of the predictions used to compute contrasts or slopes. 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 first entry in the error message is used by default. See the Type section in the documentation below.

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"
  - Unconditional: "unconditional" accounts for sampling variation in the empirical covariate distribution for averaged or aggregated predictions, comparisons, and slopes. Hypotheses applied directly to unit-level effects are rejected. Use vcovUnconditional(cluster = ~cluster) for one-way clustered unconditional inference.
  - Mixed-Models degrees of freedom: "satterthwaite", "kenward-roger"
  - Other: "NeweyWest", "KernHAC", "OPG". See the sandwich package documentation.
  - "rsample", "boot", "fwb", or "simulation": forward the result to inferences() using that method.
- 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.

wts logical, string or numeric: weights to use when computing average predictions, contrasts or slopes. These weights only affect the averaging in avg_*() or with the by argument, and not unit-level estimates. See ?weighted.mean

- 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).
- FALSE: Equal weights.
- TRUE: Extract weights from the fitted object with `insight::find_weights()` and use them when taking weighted averages of estimates. Warning: `newdata=datagrid()` returns a single average weight, which is equivalent to using `wts=FALSE`

`comparison` How should pairs of predictions be compared? Difference, ratio, odds ratio, or user-defined functions.

- 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, lift, liftavg, liftavgwts, expdydx, expdydxavg, expdydxavgwts
- 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 "Comparison functions" section below for a list of common transformations and the definitions of available shortcuts.

`transform` 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"

`rug` TRUE displays tick marks on the axes to mark the distribution of raw data.

`gray` FALSE grayscale or color plot

`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 `?slopes` documentation for a non-exhaustive list of available arguments.

Value

A `ggplot2` object

Model-Specific Arguments

Some model types allow model-specific arguments to modify the nature of marginal effects, predictions, marginal means, and contrasts. Please report other package-specific `predict()` arguments on Github so we can add them to the table below.

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

Package	Class	Argument	Documentation
brms	brmsfit	ndraws	brms::posterior_predict()
		re_formula	brms::posterior_predict()
lme4	merMod	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()
	gam	exclude	mgcv::predict.gam()
robustlmm	rlmerMod	re.form	robustlmm::predict.rlmerMod()
		allow.new.levels	robustlmm::predict.rlmerMod()
MCMCglmm	MCMCglmm	ndraws	
sampleSelection	selection	part	sampleSelection::predict.selection()

Examples

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

plot_comparisons(mod, variables = "hp", condition = "drat")

plot_comparisons(mod, variables = "hp", condition = c("drat", "am"))

plot_comparisons(mod, variables = "hp", condition = list("am", "drat" = 3:5))

plot_comparisons(mod, variables = "am", condition = list("hp", "drat" = range))

plot_comparisons(mod, variables = "am", condition = list("hp", "drat" = "threenum"))

# marginal comparisons
plot_comparisons(mod, variables = "hp", by = "am")

# marginal comparisons on a counterfactual grid
plot_comparisons(mod,
    variables = "hp",
    by = "am",
    newdata = datagrid(am = 0:1, grid_type = "counterfactual")
)
```

---

plot_predictions *Plot Conditional or Marginal Predictions*

Description

Plot predictions on the y-axis against values of one or more predictors (x-axis, colors/shapes, and facets).

The by argument is used to plot marginal predictions, that is, predictions made on the original data, but averaged by subgroups. This is analogous to using the by argument in the `predictions()` function.

The `condition` argument is used to plot conditional predictions, that is, predictions made on a user-specified grid. This is analogous to using the `newdata` argument and `datagrid()` function in a `predictions()` call. All variables whose values are not specified explicitly are treated as usual by `datagrid()`, that is, they are held at their mean or mode (or rounded mean for integers). This includes grouping variables in mixed-effects models, so analysts who fit such models may want to specify the groups of interest using the `condition` argument, or supply model-specific arguments to compute population-level estimates. See details below.

See the "Plots" vignette and website for tutorials and information on how to customize plots:

- https://marginaleffects.com/bonus/plot.html
- https://marginaleffects.com

Usage

```
plot_predictions(
  model,
  condition = NULL,
  by = NULL,
  newdata = NULL,
  type = NULL,
  vcov = NULL,
  conf_level = 0.95,
  wts = FALSE,
  transform = NULL,
  points = 0,
  rug = FALSE,
  gray = getOption("marginaleffects_plot_gray", default = FALSE),
  draw = TRUE,
  ...
)
```

Arguments

`model` Model object

`condition` Conditional predictions.

- Character vector (max length 4): Names of the predictors to display.
- Named list (max length 4): List names correspond to predictors. List elements can be:
  - Numeric vector
  - Function which returns a numeric vector or a set of unique categorical values
  - Shortcut strings for common reference values: "minmax", "quartile", "threenum"

- 1: x-axis. 2: color/shape. 3: facet (wrap if no fourth variable, otherwise cols of grid). 4: facet (rows of grid).
- Numeric variables in positions 2 and 3 are summarized by Tukey's five numbers `?stats::fivenum`

`by` Marginal predictions

- Character vector (max length 3): Names of the categorical predictors to marginalize across.
- 1: x-axis. 2: color. 3: facets.

`newdata` When `newdata` is `NULL`, the grid is determined by the `condition` argument. When `newdata` is not `NULL`, the argument behaves in the same way as in the `predictions()` function. Note that the `condition` argument builds its own grid, so the `newdata` argument is ignored if the `condition` argument is supplied.

`type` string indicates the type (scale) of the predictions used to compute contrasts or slopes. 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 first entry in the error message is used by default. See the Type section in the documentation below.

`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"`
  - Unconditional: `"unconditional"` accounts for sampling variation in the empirical covariate distribution for averaged or aggregated predictions, comparisons, and slopes. Hypotheses applied directly to unit-level effects are rejected. Use `vcovUnconditional(cluster = ~cluster)` for one-way clustered unconditional inference.
  - Mixed-Models degrees of freedom: "satterthwaite", "kenward-roger"
  - Other: `"NeweyWest"`, `"KernHAC"`, `"OPG"`. See the `sandwich` package documentation.
  - `"rsample"`, `"boot"`, `"fwb"`, or `"simulation"`: forward the result to `inferences()` using that method.
- 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.

`wts` logical, string or numeric: weights to use when computing average predictions, contrasts or slopes. These weights only affect the averaging in `avg_*()` or with the `by` argument, and not unit-level estimates. See `?weighted.mean`

- 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).
- FALSE: Equal weights.
- TRUE: Extract weights from the fitted object with `insight::find_weights()` and use them when taking weighted averages of estimates. Warning: `newdata=datagrid()` returns a single average weight, which is equivalent to using `wts=FALSE`

`transform` 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.

`points` Number between 0 and 1 which controls the transparency of raw data points. 0 (default) does not display any points. Warning: The points displayed are raw data, so the resulting plot is not a "partial residual plot."

`rug` TRUE displays tick marks on the axes to mark the distribution of raw data.

`gray` FALSE grayscale or color plot

`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 `?slopes` documentation for a non-exhaustive list of available arguments.

Value

A `ggplot2` object or data frame (if `draw=FALSE`)

Model-Specific Arguments

Some model types allow model-specific arguments to modify the nature of marginal effects, predictions, marginal means, and contrasts. Please report other package-specific `predict()` arguments on Github so we can add them to the table below.

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

Package	Class	Argument	Documentation
`brms`	`brmsfit`	`ndraws`	`brms::posterior_predict()`
		`re_formula`	`brms::posterior_predict()`
`lme4`	`merMod`	`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()
	gam	exclude	mgcv::predict.gam()
robustlmm	rlmerMod	re.form	robustlmm::predict.rlmerMod()
		allow.new.levels	robustlmm::predict.rlmerMod()
MCMCglmm	MCMCglmm	ndraws	
sampleSelection	selection	part	sampleSelection::predict.selection()

Types

The type argument determines the scale of the predictions used to compute quantities of interest with functions from the marginaleffects package. Admissible values for type depend on the model object. When users specify an incorrect value for type, marginaleffects will raise an informative error with a list of valid type values for the specific model object. The first entry in the list in that error message is the default type.

The invlink(link) is a special type defined by marginaleffects. It is available for some (but not all) models, and only for the predictions() function. With this link type, we first compute predictions on the link scale, then we use the inverse link function to backtransform the predictions to the response scale. This is useful for models with non-linear link functions as it can ensure that confidence intervals stay within desirable bounds, ex: 0 to 1 for a logit model. Note that an average of estimates with type="invlink(link)" will not always be equivalent to the average of estimates with type="response". This type is default when calling predictions(). It is available—but not default—when calling avg_predictions() or predictions() with the by argument.

Some of the most common type values are:

class	type
DirichletRegModel	response
Gam	invlink(link), response, link
Gls	lp
MCMCglmm	response
aft	surv, haz, cumhaz, density, odds, link
bam	response, link
bart	ev, ppd
betareg	response, link, precision, quantile, variance
bife	response, link
bracl	probs
brglmFit	response, link
brmsfit	response, link, prediction, average
brmultinom	probs, class
clm	prob, cum.prob, linear.predictor
clmm2	prob
clogit	expected, lp, risk, survival
coxph	survival, expected, lp, risk
coxph_weightit	survival, expected, lp, risk

crch	response, location, scale, density
fixest	invlink(link), response, link
flexsurvreg	survival, response, mean, link, lp, linear, rmst, hazard, cumhaz
gam	response, link
geeglm	response, link
glimML	response, link
glm	invlink(link), response, link
glm_weightit	invlink(link), probs, response, lp, link
glmerMod	response, link
glmgee	response, link
glmmPQL	response, link
glmmTMB	response, link, conditional, zprob, zlink, disp
glmrob	response, link
glmx	response
gnm	response, link
hetprob	pr, xb
hurdle	response, prob, count, zero
hxlr	location, cumprob, scale, density
iv_robust	response
ivpml	pr, xb
ivreg	response
lda	class, posterior
lm	response
lm_robust	response
lmerMod	response
lmerModLmerTest	response
lmrob	response
lrm	fitted, lp, mean
mblogit	response, latent, link
mclogit	response, latent, link
mhurdle	E, Ep, p
model_fit	numeric, prob, class
multinom	probs, latent
multinom_weightit	probs, response, mean
mvgam	response, link, expected, detection, latent_N
negbin	invlink(link), response, link
nestedLogit	response, link
ols	lp
oohbchoice	probability, utility
ordinal_weightit	probs, response, link, lp, mean
orm	fitted, mean, lp
polr	probs
rendo.base	response, link
rlm	response
selection	response, link, unconditional, conditional
speedglm	response, link
speedlm	response
stanreg	response, link

survreg	response, link, quantile
svy_vglm	response, link
svyglm	response, link
svyolr	probs
tobit	response, link
tobit1	expvalue, linpred, prob
workflow	numeric, prob, class
zeroinfl	response, prob, count, zero

Examples

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

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

plot_predictions(mod, condition = list("hp", wt = "threenum"))

plot_predictions(mod, condition = list("hp", wt = range))

# marginal predictions
mod <- lm(mpg ~ hp * am, data = mtcars)
plot_predictions(mod, by = "am")

# marginal predictions on a counterfactual grid
plot_predictions(mod,
    by = "am",
    newdata = datagrid(am = 0:1, grid_type = "counterfactual")
)
```

---

plot_slopes *Plot Conditional or Marginal Slopes*

Description

Plot slopes on the y-axis against values of one or more predictors (x-axis, colors/shapes, and facets).

The `by` argument is used to plot marginal slopes, that is, slopes made on the original data, but averaged by subgroups. This is analogous to using the `by` argument in the `slopes()` function.

The `condition` argument is used to plot conditional slopes, that is, slopes computed on a user-specified grid. This is analogous to using the `newdata` argument and `datagrid()` function in a `slopes()` call. All variables whose values are not specified explicitly are treated as usual by `datagrid()`, that is, they are held at their mean or mode (or rounded mean for integers). This includes grouping variables in mixed-effects models, so analysts who fit such models may want to specify the groups of interest using the `condition` argument, or supply model-specific arguments to compute population-level estimates. See details below. See the "Plots" vignette and website for tutorials and information on how to customize plots:

- https://marginaleffects.com/bonus/plot.html
- https://marginaleffects.com

Usage

```
plot_slopes(
  model,
  variables = NULL,
  condition = NULL,
  by = NULL,
  newdata = NULL,
  type = NULL,
  vcov = NULL,
  conf_level = 0.95,
  wts = FALSE,
  slope = "dydx",
  rug = FALSE,
  gray = getOption("marginaleffects_plot_gray", default = FALSE),
  draw = TRUE,
  ...
)
```

Arguments

`model`	Model object
`variables`	Name of the variable whose marginal effect (slope) we want to plot on the y-axis.
`condition`	Conditional slopes • Character vector (max length 4): Names of the predictors to display. • Named list (max length 4): List names correspond to predictors. List elements can be: – Numeric vector – Function which returns a numeric vector or a set of unique categorical values – Shortcut strings for common reference values: "minmax", "quartile", "threenum" • 1: x-axis. 2: color/shape. 3: facet (wrap if no fourth variable, otherwise cols of grid). 4: facet (rows of grid). • Numeric variables in positions 2 and 3 are summarized by Tukey's five numbers `?stats::fivenum`.
`by`	Aggregate unit-level estimates (aka, marginalize, average over). Valid inputs: • `FALSE`: return the original unit-level estimates. • `TRUE`: aggregate estimates for each term. • 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.
- For more complex aggregations, you can use the FUN argument of the hypotheses() function. See that function's documentation and the Hypothesis Test vignettes on the marginaleffects website.

newdata When newdata is NULL, the grid is determined by the condition argument. When newdata is not NULL, the argument behaves in the same way as in the predictions() function. Note that the condition argument builds its own grid, so the newdata argument is ignored if the condition argument is supplied.

type string indicates the type (scale) of the predictions used to compute contrasts or slopes. 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 first entry in the error message is used by default. See the Type section in the documentation below.

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"
  - Unconditional: "unconditional" accounts for sampling variation in the empirical covariate distribution for averaged or aggregated predictions, comparisons, and slopes. Hypotheses applied directly to unit-level effects are rejected. Use vcovUnconditional(cluster = ~cluster) for one-way clustered unconditional inference.
  - Mixed-Models degrees of freedom: "satterthwaite", "kenward-roger"
  - Other: "NeweyWest", "KernHAC", "OPG". See the sandwich package documentation.
  - "rsample", "boot", "fwb", or "simulation": forward the result to inferences() using that method.
- 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.

`wts` logical, string or numeric: weights to use when computing average predictions, contrasts or slopes. These weights only affect the averaging in `avg_*()` or with the by argument, and not unit-level estimates. See `?weighted.mean`

- 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).
- FALSE: Equal weights.
- TRUE: Extract weights from the fitted object with `insight::find_weights()` and use them when taking weighted averages of estimates. Warning: `newdata=datagrid()` returns a single average weight, which is equivalent to using `wts=FALSE`

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

- "dydx": dY/dX
- "eyex": dY/dX * X / Y
- "eydx": dY/dX / Y
- "dyex": dY/dX * X
- Y is the predicted value of the outcome; X is the observed value of the predictor.

`rug` TRUE displays tick marks on the axes to mark the distribution of raw data.

`gray` FALSE grayscale or color plot

`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 `?slopes` documentation for a non-exhaustive list of available arguments.

Value

A `ggplot2` object

Model-Specific Arguments

Some model types allow model-specific arguments to modify the nature of marginal effects, predictions, marginal means, and contrasts. Please report other package-specific `predict()` arguments on Github so we can add them to the table below.

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

Package	Class	Argument	Documentation
`brms`	`brmsfit`	`ndraws`	`brms::posterior_predict()`
		`re_formula`	`brms::posterior_predict()`

lme4	merMod	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()
	gam	exclude	mgcv::predict.gam()
robustlmm	rlmerMod	re.form	robustlmm::predict.rlmerMod()
		allow.new.levels	robustlmm::predict.rlmerMod()
MCMCglmm	MCMCglmm	ndraws	
sampleSelection	selection	part	sampleSelection::predict.selection()

Examples

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

plot_slopes(mod, variables = "hp", condition = "drat")

plot_slopes(mod, variables = "hp", condition = c("drat", "am"))

plot_slopes(mod, variables = "hp", condition = list("am", "drat" = 3:5))

plot_slopes(mod, variables = "am", condition = list("hp", "drat" = range))

plot_slopes(mod, variables = "am", condition = list("hp", "drat" = "threenum"))

# marginal slopes
plot_slopes(mod, variables = "hp", by = "am")

# marginal slopes on a counterfactual grid
plot_slopes(mod,
    variables = "hp",
    by = "am",
    newdata = datagrid(am = 0:1, grid_type = "counterfactual")
)
```

---

predictions *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").

- `predictions()`: unit-level (conditional) estimates.
- `avg_predictions()`: average (marginal) estimates.

The newdata argument and the datagrid() function can be used to control where statistics are evaluated in the predictor space: "at observed values", "at the mean", "at representative values", etc.

See the predictions vignette and package website for worked examples and case studies:

- https://marginaleffects.com/chapters/predictions.html
- https://marginaleffects.com/

Usage

```
predictions(
  model,
  newdata = NULL,
  variables = NULL,
  vcov = TRUE,
  conf_level = 0.95,
  type = NULL,
  by = FALSE,
  wts = FALSE,
  transform = NULL,
  hypothesis = NULL,
  equivalence = NULL,
  df = Inf,
  numderiv = "fdforward",
  ...
)

avg_predictions(
  model,
  newdata = NULL,
  variables = NULL,
  vcov = TRUE,
  conf_level = 0.95,
  type = NULL,
  by = TRUE,
  wts = FALSE,
  transform = NULL,
  hypothesis = NULL,
  equivalence = NULL,
  df = Inf,
  numderiv = "fdforward",
  ...
)
```

Arguments

model	Model object
newdata	Grid of predictor values at which we evaluate predictions.

- Warning: When newdata is NULL, the data is retrieved using get_modeldata(), which may have to extract it from the modeling environment. This can produce unexpected results if the data has been modified, or when called inside lapply(), Shiny apps, or nested functions. To avoid potential issues, consider using set_modeldata() to attach the training data to the model object explicitly.
- NULL (default): Unit-level predictions for each observed value in the dataset (empirical distribution). The dataset is retrieved using get_modeldata().
- string:
  - "mean": Predictions evaluated when each predictor is held at its mean or mode.
  - "median": Predictions evaluated when each predictor is held at its median or mode.
  - "balanced": Predictions evaluated on a balanced grid with every combination of categories and numeric variables held at their means.
  - "tukey": Predictions evaluated at Tukey's 5 numbers.
  - "grid": Predictions evaluated 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.
- subset() call with a single argument to select a subset of the dataset used to fit the model, ex: newdata = subset(treatment == 1)
- dplyr::filter() call with a single argument to select a subset of the dataset used to fit the model, ex: newdata = filter(treatment == 1)

variables Counterfactual variables.

- Output:
  - predictions(): The entire dataset is replicated once for each unique combination of variables, and predictions are made.
  - avg_predictions(): The entire dataset is replicated, predictions are made, and they are marginalized by variables categories.
  - Warning: This can be expensive in large datasets.
  - Warning: Users who need "conditional" predictions should use the newdata argument instead of variables.
- Input:
  - NULL: computes one prediction per row of newdata
  - Character vector: the dataset is replicated once of every combination of unique values of the variables identified in variables.
  - 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

`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"`
  - Unconditional: `"unconditional"` accounts for sampling variation in the empirical covariate distribution for averaged or aggregated predictions, comparisons, and slopes. Hypotheses applied directly to unit-level effects are rejected. Use `vcovUnconditional(cluster = ~cluster)` for one-way clustered unconditional inference.
  - Mixed-Models degrees of freedom: "satterthwaite", "kenward-roger"
  - Other: `"NeweyWest"`, `"KernHAC"`, `"OPG"`. See the `sandwich` package documentation.
  - `"rsample"`, `"boot"`, `"fwb"`, or `"simulation"`: forward the result to `inferences()` using that method.
- 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 contrasts or slopes. 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 first entry in the error message is used by default. See the Type section in the documentation below.

`by` Aggregate unit-level estimates (aka, marginalize, average over). Valid inputs:

- `FALSE`: return the original unit-level estimates.

- TRUE: aggregate estimates for each term.
- 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.
- For more complex aggregations, you can use the FUN argument of the hypotheses() function. See that function's documentation and the Hypothesis Test vignettes on the marginaleffects website.

wts — logical, string or numeric: weights to use when computing average predictions, contrasts or slopes. These weights only affect the averaging in avg_*() or with the by argument, and not unit-level estimates. See ?weighted.mean

- 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).
- FALSE: Equal weights.
- TRUE: Extract weights from the fitted object with insight::find_weights() and use them when taking weighted averages of estimates. Warning: newdata=datagrid() returns a single average weight, which is equivalent to using wts=FALSE

transform — 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 number , formula, string equation, vector, matrix, or function.

- Number: The null hypothesis used in the computation of Z and p (before applying transform).
- String: Equation to specify linear or non-linear hypothesis tests. Two-tailed tests must include an equal = sign. One-tailed tests must start with < or >. If the terms in coef(object) uniquely identify estimates, they can be used in the formula. Otherwise, use b1, b2, etc. to identify the position of each parameter. The b* wildcard can be used to test hypotheses on all estimates. When the hypothesis string represents a two-sided equation, the estimate column holds the value of the left side minus the right side of the equation. If a named vector is used, the names are used as labels in the output. Examples:
  - hp = drat
  - hp + drat = 12
  - b1 + b2 + b3 = 0
  - b* / b1 = 1
  - <= 0
  - >= -3.5

  - `b1 >= 10`
- Formula: `lhs ~ rhs | group`
  - `lhs`
    * `ratio` (null = 1)
    * `difference` (null = 0)
    * Leave empty for default value
  - `rhs`
    * `pairwise` and `revpairwise`: 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.
    * `meandev`: difference between an estimate and the mean of all estimates.
    * `meanotherdev`: difference between an estimate and the mean of all other estimates, excluding the current one.
    * `poly`: polynomial contrasts, as computed by the `stats::contr.poly()` function.
    * `helmert`: Helmert contrasts, as computed by the `stats::contr.helmert()` function. Contrast 2nd level to the first, 3rd to the average of the first two, and so on.
    * `trt_vs_ctrl`: difference between the mean of estimates (except the first) and the first estimate.
    * `I(fun(x))`: custom function to manipulate the vector of estimates `x`. The function `fun()` can return multiple (potentially named) estimates.
  - `group` (optional)
    * Column name of `newdata`. Conduct hypothesis tests withing subsets of the data.
  - Examples:
    * `~ poly`
    * `~ sequential | groupid`
    * `~ reference`
    * `ratio ~ pairwise`
    * `difference ~ pairwise | groupid`
    * `~ I(x - mean(x)) | groupid`
    * `~ I(\(x) c(a = x[1], b = mean(x[2:3]))) | groupid`
- Matrix or Vector: Each column is a vector of weights. The the output is the dot product between these vectors of weights and the vector of estimates. The matrix can have column names to label the estimates.
- Function:
  - Accepts an argument `x`: object produced by a `marginaleffects` function or a data frame with column `rowid` and `estimate`

  - Returns a data frame with columns `term` and `estimate` (mandatory) and `rowid` (optional).
  - The function can also accept optional input arguments: `newdata`, `by`, `draws`.
  - This function approach will not work for Bayesian models or with bootstrapping. In those cases, it is easy to use `get_draws()` to extract and manipulate the draws directly.
- See the Examples section below and the vignette: https://marginaleffects.com/chapters/hypothesis.html
- Warning: When calling `predictions()` with `type="invlink(link)"` (the default in some models), `hypothesis` is tested and p values are computed on the link scale.

`equivalence` Numeric vector of length 2: bounds used for the two-one-sided test (TOST) of equivalence, and for the non-inferiority and non-superiority tests. For bayesian models, this report the proportion of posterior draws in the interval and the ROPE. See Details section below.

`df` Degrees of freedom used to compute p values and confidence intervals.

- A single numeric value between 1 and `Inf`, or a numeric vector with length equal to the number of rows in the output. When `df` is `Inf`, the normal distribution is used. When `df` is finite, the `t` distribution is used.
- "residual": Calls insight::get_df to extract degrees of freedom from the model automatically.
- "satterthwaite" or "kenward-roger": Use the Satterthwaite or Kenward-Roger approximation to compute degrees of freedom in mixed effects models.

`numderiv` string or list of strings indicating the method to use to for the numeric differentiation used in to compute delta method standard errors.

- "fdforward": finite difference method with forward differences (default)
- "fdcenter": finite difference method with central differences
- "richardson": Richardson extrapolation method
- Extra arguments can be specified by passing a list to the `numDeriv` argument, with the name of the method first and named arguments following, ex: `numderiv=list("fdcenter", eps = 1e-5)`. When an unknown argument is used, `marginaleffects` prints the list of valid arguments for each method.

`...` 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 `?slopes` documentation for a non-exhaustive list of available arguments.

Value

A `data.frame` with one row per estimate. This data frame is pretty-printed by default, but users can interact with it as a regular data frame, with functions like `nrow()`, `head()`, `colnames()`, etc.

Values can be extracted using standard `[,]` or `$` operators, and manipulated using external packages like `dplyr` or `data.table`.

Columns may include:

- `rowid`: row number of the `newdata` data frame
- `group`: (optional) value of the grouped outcome (e.g., categorical outcome models)
- `term`: the focal variable.
- `estimate`: an estimate of the prediction, counterfactual comparison, or slope.
- `std.error`: standard errors computed via the delta method.
- `p.value`: p value associated to the `estimate` column. The null is determined by the `hypothesis` argument (0 by default).
- `s.value`: Shannon information transforms of p values. See the S values vignette at https://marginaleffects.com the marginaleffects website.
- `conf.low`: lower bound of the confidence (or credible) interval defined by the `conf_level` argument.
- `conf.high`: upper bound of the confidence (or credible) interval defined by the `conf_level` argument.
- `predicted_lo`: predicted outcome for the "low" value of the focal predictor in a counterfactual comparison.
- `predicted_hi`: predicted outcome for the "high" value of the focal predictor in a counterfactual comparison.
- `p.rope.unconditional`: share of posterior draws in the interval specified by the `equivalence` argument. This is only available for Bayesian models.
- `p.rope.conditional`: share of posterior draws in the interval specified by the `equivalence` argument, among draws in the confidence interval. This is only available for Bayesian models.
- `rope`: share of the posterior draws between `conf.low` and `conf.high` that are covered by the interval specified by the `equivalence` argument.
- `statistic.noninf`: test statistic for non-inferiority test (when `equivalence` argument is used).
- `statistic.nonsup`: test statistic for non-superiority test (when `equivalence` argument is used).
- `p.value.noninf`: p-value for non-inferiority test (when `equivalence` argument is used).
- `p.value.nonsup`: p-value for non-superiority test (when `equivalence` argument is used).
- `p.value.equiv`: p-value for equivalence test using Two One-Sided Tests (TOST) approach (when `equivalence` argument is used).

See `?print.marginaleffects` for printing options.

The `data.frames` produced by `marginaleffects` stores an attribute that holds many internal objects, such as the original model, data, and much other information that can be used for post-processing. This information can be inspected using the `components()` function.

Warning: The internal attributes retrieved by `components()` are not considered part of the public API of the package. Their names and contents can change without warning or notice. Users should not rely on them.

Warning: In some cases, the internal attributes used by `marginaleffects()` can use up a substantial amount of memory. To clear this data, use the `prune()` function or set `options(marginaleffects_lean=TRUE)`.

Functions

- `avg_predictions()`: Average predictions

Standard errors using the delta method

Standard errors for all quantities estimated by `marginaleffects` can be obtained via the delta method. This requires differentiating a function with respect to the coefficients in the model using a finite difference approach. In some models, the delta method standard errors can be sensitive to various aspects of the numeric differentiation strategy, including the step size. By default, the step size is set to `1e-8`, or to `1e-4` times the smallest absolute model coefficient, whichever is largest.

`marginaleffects` can delegate numeric differentiation to the `numDeriv` package, which allows more flexibility. To do this, users can pass arguments to the `numDeriv::jacobian` function through a global option. For example:

- `options(marginaleffects_numDeriv = list(method = "simple", method.args = list(eps = 1e-6)))`
- `options(marginaleffects_numDeriv = list(method = "Richardson", method.args = list(eps = 1e-5)))`
- `options(marginaleffects_numDeriv = NULL)`

See the "Uncertainty" chapter on the `marginaleffects` website for more details on the computation of standard errors, bootstrapping, and more:

https://marginaleffects.com/chapters/uncertainty.html

Model-Specific Arguments

Some model types allow model-specific arguments to modify the nature of marginal effects, predictions, marginal means, and contrasts. Please report other package-specific `predict()` arguments on Github so we can add them to the table below.

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

Package	Class	Argument	Documentation
brms	brmsfit	ndraws	brms::posterior_predict()
		re_formula	brms::posterior_predict()
lme4	merMod	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()
	gam	exclude	mgcv::predict.gam()
robustlmm	rlmerMod	re.form	robustlmm::predict.rlmerMod()
		allow.new.levels	robustlmm::predict.rlmerMod()
MCMCglmm	MCMCglmm	ndraws	
sampleSelection	selection	part	sampleSelection::predict.selection()

Bayesian posterior summaries

By default, credible intervals in bayesian models are built as equal-tailed intervals. This can be changed to a highest density interval by setting a global option:

```
options("marginaleffects_posterior_interval" = "eti")
```

```
options("marginaleffects_posterior_interval" = "hdi")
```

By default, the center of the posterior distribution in bayesian models is identified by the median. Users can use a different summary function by setting a global option:

```
options("marginaleffects_posterior_center" = "mean")
```

```
options("marginaleffects_posterior_center" = "median")
```

When estimates are averaged using the by argument, the `tidy()` function, or the `summary()` function, the posterior distribution is marginalized twice over. First, we take the average *across* units but *within* each iteration of the MCMC chain, according to what the user requested in by argument or `tidy()/summary()` functions. Then, we identify the center of the resulting posterior using the function supplied to the `"marginaleffects_posterior_center"` option (the median by default).

Equivalence, Inferiority, Superiority

θ is an estimate, σ_θ its estimated standard error, and $[a, b]$ are the bounds of the interval supplied to the `equivalence` argument.

Non-inferiority:

- H_0: $\theta \leq a$
- H_1: $\theta > a$
- $t = (\theta - a)/\sigma_\theta$
- p: Upper-tail probability

Non-superiority:

- H_0: $\theta \geq b$
- H_1: $\theta < b$
- $t = (\theta - b)/\sigma_\theta$
- p: Lower-tail probability

Equivalence: Two One-Sided Tests (TOST)

- p: Maximum of the non-inferiority and non-superiority p values.

Thanks to Russell V. Lenth for the excellent `emmeans` package and documentation which inspired this feature.

Order of operations

Behind the scenes, the arguments of `marginaleffects` functions are evaluated in this order:

1. `newdata`
2. `variables`

3. comparison and slope
4. by
5. vcov
6. hypothesis
7. transform

Parallel computation

The slopes() and comparisons() functions can use parallelism to speed up computation. Operations are parallelized for the computation of standard errors, at the model coefficient level. There is always considerable overhead when using parallel computation, mainly involved in passing the whole dataset to the different processes. Thus, parallel computation is most likely to be useful when the model includes many parameters and the dataset is relatively small.

Warning: In many cases, parallel processing will not be useful at all.

To activate parallel computation, users must load the future.apply package, call plan() function, and set a global option.

options(marginaleffects_parallel = TRUE): parallelize delta method computation of standard errors. options(marginaleffects_parallel_inferences = TRUE): parallelize "rsample" or "fwb" bootstrap computation in inferences(). options(marginaleffects_parallel_packages = TRUE): vector of strings with the names of modeling packages used to fit the model, ex: c("survival", "splines")

For example:

```
library(future.apply)
plan("multisession", workers = 4)
options(marginaleffects_parallel = FALSE)
options(marginaleffects_parallel_inferences = TRUE)
options(marginaleffects_parallel_packages = c("survival", "splines"))

slopes(model)
```

To disable parallelism in marginaleffects altogether, you can set a global option:

```
options(marginaleffects_parallel = FALSE)
```

Global options

The behavior of marginaleffects functions can be modified by setting global options.

Disable some safety checks and warnings:

- options(marginaleffects_startup_message = FALSE)
  - Disable the startup message printed on library(marginaleffects).
- options(marginaleffects_safe = FALSE)
  - Disable safety checks and warnings.
- options(marginaleffects_print_omit = c("p.value", "s.value"))

- Omit some columns from the printed output.

Enforce lean return objects, sans information about the original model and data, and other ancillary attributes. Note that this will disable some advanced post-processing features and functions like hypotheses.

```
options(marginaleffects_lean = TRUE)
```

Other options:

- `marginaleffects_plot_gray`: logical. If `TRUE`, the default color of the plot is gray. Default is `FALSE`.

Types

The `type` argument determines the scale of the predictions used to compute quantities of interest with functions from the `marginaleffects` package. Admissible values for `type` depend on the model object. When users specify an incorrect value for `type`, `marginaleffects` will raise an informative error with a list of valid `type` values for the specific model object. The first entry in the list in that error message is the default type.

The `invlink(link)` is a special type defined by `marginaleffects`. It is available for some (but not all) models, and only for the `predictions()` function. With this link type, we first compute predictions on the link scale, then we use the inverse link function to backtransform the predictions to the response scale. This is useful for models with non-linear link functions as it can ensure that confidence intervals stay within desirable bounds, ex: 0 to 1 for a logit model. Note that an average of estimates with `type="invlink(link)"` will not always be equivalent to the average of estimates with `type="response"`. This type is default when calling `predictions()`. It is available—but not default—when calling `avg_predictions()` or `predictions()` with the `by` argument.

Some of the most common `type` values are:

class	type
DirichletRegModel	response
Gam	invlink(link), response, link
Gls	lp
MCMCglmm	response
aft	surv, haz, cumhaz, density, odds, link
bam	response, link
bart	ev, ppd
betareg	response, link, precision, quantile, variance
bife	response, link
bracl	probs
brglmFit	response, link
brmsfit	response, link, prediction, average
brmultinom	probs, class
clm	prob, cum.prob, linear.predictor
clmm2	prob
clogit	expected, lp, risk, survival
coxph	survival, expected, lp, risk

coxph_weightit	survival, expected, lp, risk
crch	response, location, scale, density
fixest	invlink(link), response, link
flexsurvreg	survival, response, mean, link, lp, linear, rmst, hazard, cumhaz
gam	response, link
geeglm	response, link
glimML	response, link
glm	invlink(link), response, link
glm_weightit	invlink(link), probs, response, lp, link
glmerMod	response, link
glmgee	response, link
glmmPQL	response, link
glmmTMB	response, link, conditional, zprob, zlink, disp
glmrob	response, link
glmx	response
gnm	response, link
hetprob	pr, xb
hurdle	response, prob, count, zero
hxlr	location, cumprob, scale, density
iv_robust	response
ivpml	pr, xb
ivreg	response
lda	class, posterior
lm	response
lm_robust	response
lmerMod	response
lmerModLmerTest	response
lmrob	response
lrm	fitted, lp, mean
mblogit	response, latent, link
mclogit	response, latent, link
mhurdle	E, Ep, p
model_fit	numeric, prob, class
multinom	probs, latent
multinom_weightit	probs, response, mean
mvgam	response, link, expected, detection, latent_N
negbin	invlink(link), response, link
nestedLogit	response, link
ols	lp
oohbchoice	probability, utility
ordinal_weightit	probs, response, link, lp, mean
orm	fitted, mean, lp
polr	probs
rendo.base	response, link
rlm	response
selection	response, link, unconditional, conditional
speedglm	response, link
speedlm	response

stanreg	response, link
survreg	response, link, quantile
svy_vglm	response, link
svyglm	response, link
svyolr	probs
tobit	response, link
tobit1	expvalue, linpred, prob
workflow	numeric, prob, class
zeroinfl	response, prob, count, zero

References

- Arel-Bundock V, Greifer N, Heiss A (2024). "How to Interpret Statistical Models Using marginaleffects for R and Python." *Journal of Statistical Software*, *111*(9), 1-32. doi:10.18637/jss.v111.i09 doi:10.18637/jss.v111.i09
- Arel-Bundock (2026). "Model to Meaning: How to interpret statistical models in R and Python." CRC Press. https://routledge.com/9781032908724
- Greenland S. 2019. "Valid P-Values Behave Exactly as They Should: Some Misleading Criticisms of P-Values and Their Resolution With S-Values." The American Statistician. 73(S1): 106–114.
- Cole, Stephen R, Jessie K Edwards, and Sander Greenland. 2020. "Surprise!" American Journal of Epidemiology 190 (2): 191–93. doi:10.1093/aje/kwaa136
- Hansen SN, Overgaard M (2025). "Variance Estimation for Average Treatment Effects Estimated by G-Computation." *Metrika*, *88*(4), 419–443. doi:10.1007/s00184024009624

Examples

```
library("marginaleffects")
# 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)

avg_predictions(mod)

predictions(mod, by = "am")

# Conditional Adjusted Predictions
```

```
plot_predictions(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)

# average counterfactual predictions
avg_predictions(mod, variables = "am")

# 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
p <- predictions(nom, type = "probs")
head(p)

# average predictions
avg_predictions(nom, type = "probs", by = "group")

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)))
hyp <- function(x) {
  out <- aggregate(estimate ~ by, merge(x, by, by = "group"), sum)
  names(out)[1] <- "hypothesis"
  out
}
predictions(mod, newdata = "mean", hypothesis = hyp)
```

`print.marginaleffects` *Print* `marginaleffects` *objects*

Description

This function controls the text which is printed to the console when one of the core `marginalefffects` functions is called and the object is returned: `predictions()`, `comparisons()`, `slopes()`, `hypotheses()`, `avg_predictions()`, `avg_comparisons()`, `avg_slopes()`.

All of those functions return standard data frames. Columns can be extracted by name, `predictions(model)$estimate`, and all the usual data manipulation functions work out-of-the-box: `colnames()`, `head()`, `subset()`, `dplyr::filter()`, `dplyr::arrange()`, etc.

Some of the data columns are not printed by default. You can disable pretty printing and print the full results as a standard data frame using the `style` argument or by applying `as.data.frame()` on the object. See examples below.

Usage

```
## S3 method for class 'marginaleffects'
print(
  x,
  style = getOption("marginaleffects_print_style", default = "summary"),
  digits = getOption("marginaleffects_print_digits", default = 3),
  p_eps = getOption("marginaleffects_print_p_eps", default = 0.001),
  topn = getOption("marginaleffects_print_topn", default = 5),
  nrows = getOption("marginaleffects_print_nrows", default = 30),
  ncols = getOption("marginaleffects_print_ncols", default = 30),
  type = getOption("marginaleffects_print_type", default = TRUE),
 column_names = getOption("marginaleffects_print_column_names", default = FALSE),
  ...
)
```

Arguments

x	An object produced by one of the marginaleffects package functions.
style	"summary", "data.frame", or "tinytable"
digits	The number of digits to display.
p_eps	p values smaller than this number are printed in "<0.001" style.
topn	The number of rows to be printed from the beginning and end of tables with more than nrows rows.
nrows	The number of rows which will be printed before truncation.
ncols	The maximum number of column names to display at the bottom of the printed output.
type	boolean: should the type be printed?
column_names	boolean: should the column names be printed?
...	Other arguments are currently ignored.

Examples

```
library(marginaleffects)
mod <- lm(mpg ~ hp + am + factor(gear), data = mtcars)
p <- predictions(mod, by = c("am", "gear"))
p

subset(p, am == 1)

print(p, style = "data.frame")

data.frame(p)
```

prune.marginaleffects *Prune marginaleffects objects to reduce memory usage*

Description

Remove large attributes from marginaleffects objects to reduce memory usage. Warning: This will disable many useful post-processing features of `marginaleffects`

Usage

```
## S3 method for class 'marginaleffects'
prune(tree, component = "all", ...)
```

Arguments

`tree`	A marginaleffects object (predictions, comparisons, slopes, or hypotheses)
`component`	A character string indicating which component to prune: "all" or "modeldata".
`...`	Unused

Details

...

Value

A pruned marginaleffects object

refit *Refit a marginaleffects object with new data*

Description

Refit a marginaleffects object with new data

Usage

```
refit(object, ...)

## S3 method for class 'marginaleffects'
refit(object, data = NULL, newdata = NULL, vcov = NULL, ...)

## S3 method for class 'predictions'
refit(object, data = NULL, newdata = NULL, vcov = NULL, ...)

## S3 method for class 'comparisons'
```

```
refit(object, data = NULL, newdata = NULL, vcov = NULL, ...)

## S3 method for class 'slopes'
refit(object, data = NULL, newdata = NULL, vcov = NULL, ...)

## S3 method for class 'hypotheses'
refit(object, data = NULL, newdata = NULL, vcov = NULL, ...)
```

Arguments

object	A marginaleffects object (predictions, comparisons, or slopes)
...	Additional arguments passed to methods
data	Optional data frame to refit the underlying model
newdata	Optional data frame to re-evaluate the marginaleffects call
vcov	Optional logical or variance-covariance matrix specification to pass to the marginaleffects call

Details

If `data` is supplied, the underlying model is refitted using that data. If `newdata` is supplied, the marginaleffects call is re-evaluated with the new data. Both can be supplied together to refit the model and make predictions on new data.

Value

A marginaleffects object

`slopes` *Slopes (aka Partial derivatives, Marginal Effects, or Trends)*

Description

Partial derivative of the regression equation with respect to a regressor of interest.

- `slopes()`: unit-level (conditional) estimates.
- `avg_slopes()`: average (marginal) estimates.

The `newdata` argument and the `datagrid()` function can be used to control where statistics are evaluated in the predictor space: "at observed values", "at the mean", "at representative values", etc.

See the slopes vignette and package website for worked examples and case studies:

- https://marginaleffects.com/chapters/slopes.html
- https://marginaleffects.com/

Warning: Slopes and elasticities can only be calculated for continuous numeric variables. The `slopes()` functions will automatically revert to `comparisons()` for binary or categorical variables.

Usage

```
slopes(
  model,
  newdata = NULL,
  variables = NULL,
  type = NULL,
  by = FALSE,
  vcov = TRUE,
  conf_level = 0.95,
  slope = "dydx",
  wts = FALSE,
  hypothesis = NULL,
  equivalence = NULL,
  df = Inf,
  eps = NULL,
  numderiv = "fdforward",
  ...
)

avg_slopes(
  model,
  newdata = NULL,
  variables = NULL,
  type = NULL,
  by = TRUE,
  vcov = TRUE,
  conf_level = 0.95,
  slope = "dydx",
  wts = FALSE,
  hypothesis = NULL,
  equivalence = NULL,
  df = Inf,
  eps = NULL,
  numderiv = "fdforward",
  ...
)
```

Arguments

`model`	Model object
`newdata`	Grid of predictor values at which we evaluate the slopes. • Warning: When `newdata` is `NULL`, the data is retrieved using `get_modeldata()`, which may have to extract it from the modeling environment. This can produce unexpected results if the data has been modified, or when called inside `lapply()`, Shiny apps, or nested functions. To avoid potential issues, consider using `set_modeldata()` to attach the training data to the model object explicitly.

- NULL (default): Unit-level slopes for each observed value in the dataset (empirical distribution). The dataset is retrieved using get_modeldata().
- 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.
- subset() call with a single argument to select a subset of the dataset used to fit the model, ex: newdata = subset(treatment == 1)
- dplyr::filter() call with a single argument to select a subset of the dataset used to fit the model, ex: newdata = filter(treatment == 1)
- string:
  - "mean": Slopes evaluated when each predictor is held at its mean or mode.
  - "median": Slopes evaluated when each predictor is held at its median or mode.
  - "balanced": Slopes evaluated on a balanced grid with every combination of categories and numeric variables held at their means.
  - "tukey": Slopes evaluated at Tukey's 5 numbers.
  - "grid": Slopes evaluated on a grid of representative numbers (Tukey's 5 numbers and unique values of categorical predictors).

variables — Focal variables

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

type — string indicates the type (scale) of the predictions used to compute contrasts or slopes. 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 first entry in the error message is used by default. See the Type section in the documentation below.

by — Aggregate unit-level estimates (aka, marginalize, average over). Valid inputs:

- FALSE: return the original unit-level estimates.
- TRUE: aggregate estimates for each term.
- 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.
- For more complex aggregations, you can use the FUN argument of the hypotheses() function. See that function's documentation and the Hypothesis Test vignettes on the marginaleffects website.

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"`
  - Unconditional: `"unconditional"` accounts for sampling variation in the empirical covariate distribution for averaged or aggregated predictions, comparisons, and slopes. Hypotheses applied directly to unit-level effects are rejected. Use `vcovUnconditional(cluster = ~cluster)` for one-way clustered unconditional inference.
  - Mixed-Models degrees of freedom: "satterthwaite", "kenward-roger"
  - Other: `"NeweyWest"`, `"KernHAC"`, `"OPG"`. See the `sandwich` package documentation.
  - `"rsample"`, `"boot"`, `"fwb"`, or `"simulation"`: forward the result to `inferences()` using that method.
- 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.

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

- "dydx": dY/dX
- "eyex": dY/dX * X / Y
- "eydx": dY/dX / Y
- "dyex": dY/dX * X
- Y is the predicted value of the outcome; X is the observed value of the predictor.

`wts` logical, string or numeric: weights to use when computing average predictions, contrasts or slopes. These weights only affect the averaging in `avg_*()` or with the `by` argument, and not unit-level estimates. See `?weighted.mean`

- 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).
- FALSE: Equal weights.
- TRUE: Extract weights from the fitted object with `insight::find_weights()` and use them when taking weighted averages of estimates. Warning: `newdata=datagrid()` returns a single average weight, which is equivalent to using `wts=FALSE`

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

- Number: The null hypothesis used in the computation of Z and p (before applying `transform`).
- String: Equation to specify linear or non-linear hypothesis tests. Two-tailed tests must include an equal `=` sign. One-tailed tests must start with `<` or `>`. If the terms in `coef(object)` uniquely identify estimates, they can be used in the formula. Otherwise, use `b1`, `b2`, etc. to identify the position of each parameter. The `b*` wildcard can be used to test hypotheses on all estimates. When the hypothesis string represents a two-sided equation, the `estimate` column holds the value of the left side minus the right side of the equation. If a named vector is used, the names are used as labels in the output. Examples:
  - `hp = drat`
  - `hp + drat = 12`
  - `b1 + b2 + b3 = 0`
  - `b* / b1 = 1`
  - `<= 0`
  - `>= -3.5`
  - `b1 >= 10`
- Formula: `lhs ~ rhs | group`
  - `lhs`
    - `ratio` (null = 1)
    - `difference` (null = 0)
    - Leave empty for default value
  - `rhs`
    - `pairwise` and `revpairwise`: 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.
    - `meandev`: difference between an estimate and the mean of all estimates.
    - `meanotherdev`: difference between an estimate and the mean of all other estimates, excluding the current one.
    - `poly`: polynomial contrasts, as computed by the `stats::contr.poly()` function.
    - `helmert`: Helmert contrasts, as computed by the `stats::contr.helmert()` function. Contrast 2nd level to the first, 3rd to the average of the first two, and so on.
    - `trt_vs_ctrl`: difference between the mean of estimates (except the first) and the first estimate.
    - `I(fun(x))`: custom function to manipulate the vector of estimates `x`. The function `fun()` can return multiple (potentially named) estimates.

  - group (optional)
    * Column name of newdata. Conduct hypothesis tests withing subsets of the data.
  - Examples:
    * ~ poly
    * ~ sequential | groupid
    * ~ reference
    * ratio ~ pairwise
    * difference ~ pairwise | groupid
    * ~ I(x - mean(x)) | groupid
    * ~ I(\(x) c(a = x[1], b = mean(x[2:3]))) | groupid
- Matrix or Vector: Each column is a vector of weights. The the output is the dot product between these vectors of weights and the vector of estimates. The matrix can have column names to label the estimates.
- Function:
  - Accepts an argument x: object produced by a marginaleffects function or a data frame with column rowid and estimate
  - Returns a data frame with columns term and estimate (mandatory) and rowid (optional).
  - The function can also accept optional input arguments: newdata, by, draws.
  - This function approach will not work for Bayesian models or with bootstrapping. In those cases, it is easy to use get_draws() to extract and manipulate the draws directly.
- See the Examples section below and the vignette: https://marginaleffects.com/chapters/hypothesis.html
- Warning: When calling predictions() with type="invlink(link)" (the default in some models), hypothesis is tested and p values are computed on the link scale.

equivalence
: Numeric vector of length 2: bounds used for the two-one-sided test (TOST) of equivalence, and for the non-inferiority and non-superiority tests. For bayesian models, this report the proportion of posterior draws in the interval and the ROPE. See Details section below.

df
: Degrees of freedom used to compute p values and confidence intervals.

- A single numeric value between 1 and Inf, or a numeric vector with length equal to the number of rows in the output. When df is Inf, the normal distribution is used. When df is finite, the t distribution is used.
- "residual": Calls insight::get_df to extract degrees of freedom from the model automatically.
- "satterthwaite" or "kenward-roger": Use the Satterthwaite or Kenward-Roger approximation to compute degrees of freedom in mixed effects models.

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.

`numderiv` string or list of strings indicating the method to use to for the numeric differentiation used in to compute delta method standard errors.

- "fdforward": finite difference method with forward differences (default)
- "fdcenter": finite difference method with central differences
- "richardson": Richardson extrapolation method
- Extra arguments can be specified by passing a list to the `numDeriv` argument, with the name of the method first and named arguments following, ex: `numderiv=list("fdcenter", eps = 1e-5)`. When an unknown argument is used, `marginaleffects` prints the list of valid arguments for each method.

`...` 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 `?slopes` documentation for a non-exhaustive list of available arguments.

Details

A "slope" or "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 slopes for a vast array of models, including non-linear models with transformations (e.g., polynomials). Uncertainty estimates are computed using the delta method.

Numerical derivatives for the `slopes` function are calculated using a simple epsilon difference approach: $\partial Y/\partial X = (f(X + \varepsilon/2) - f(X - \varepsilon/2))/\varepsilon$, where f is the `predict()` method associated with the model class, and ε is determined by the `eps` argument.

Value

A `data.frame` with one row per estimate. This data frame is pretty-printed by default, but users can interact with it as a regular data frame, with functions like `nrow()`, `head()`, `colnames()`, etc. Values can be extracted using standard `[,]` or `$` operators, and manipulated using external packages like `dplyr` or `data.table`.

Columns may include:

- `rowid`: row number of the `newdata` data frame
- `group`: (optional) value of the grouped outcome (e.g., categorical outcome models)
- `term`: the focal variable.
- `estimate`: an estimate of the prediction, counterfactual comparison, or slope.
- `std.error`: standard errors computed via the delta method.
- `p.value`: p value associated to the `estimate` column. The null is determined by the `hypothesis` argument (0 by default).
- `s.value`: Shannon information transforms of p values. See the S values vignette at https://marginaleffects.com the marginaleffects website.

- conf.low: lower bound of the confidence (or credible) interval defined by the conf_level argument.
- conf.high: upper bound of the confidence (or credible) interval defined by the conf_level argument.
- predicted_lo: predicted outcome for the "low" value of the focal predictor in a counterfactual comparison.
- predicted_hi: predicted outcome for the "high" value of the focal predictor in a counterfactual comparison.
- p.rope.unconditional: share of posterior draws in the interval specified by the equivalence argument. This is only available for Bayesian models.
- p.rope.conditional: share of posterior draws in the interval specified by the equivalence argument, among draws in the confidence interval. This is only available for Bayesian models.
- rope: share of the posterior draws between conf.low and conf.high that are covered by the interval specified by the equivalence argument.
- statistic.noninf: test statistic for non-inferiority test (when equivalence argument is used).
- statistic.nonsup: test statistic for non-superiority test (when equivalence argument is used).
- p.value.noninf: p-value for non-inferiority test (when equivalence argument is used).
- p.value.nonsup: p-value for non-superiority test (when equivalence argument is used).
- p.value.equiv: p-value for equivalence test using Two One-Sided Tests (TOST) approach (when equivalence argument is used).

See ?print.marginaleffects for printing options.

The data.frames produced by marginaleffects stores an attribute that holds many internal objects, such as the original model, data, and much other information that can be used for post-processing. This information can be inspected using the components() function.

Warning: The internal attributes retrieved by components() are not considered part of the public API of the package. Their names and contents can change without warning or notice. Users should not rely on them.

Warning: In some cases, the internal attributes used by marginaleffects() can use up a substantial amount of memory. To clear this data, use the prune() function or set options(marginaleffects_lean=TRUE).

Functions

- avg_slopes(): Average slopes

Standard errors using the delta method

Standard errors for all quantities estimated by marginaleffects can be obtained via the delta method. This requires differentiating a function with respect to the coefficients in the model using a finite difference approach. In some models, the delta method standard errors can be sensitive to various aspects of the numeric differentiation strategy, including the step size. By default, the step size is set to 1e-8, or to 1e-4 times the smallest absolute model coefficient, whichever is largest.

marginaleffects can delegate numeric differentiation to the numDeriv package, which allows more flexibility. To do this, users can pass arguments to the numDeriv::jacobian function through a global option. For example:

- options(marginaleffects_numDeriv = list(method = "simple", method.args = list(eps = 1e-6)))
- options(marginaleffects_numDeriv = list(method = "Richardson", method.args = list(eps = 1e-5)))
- options(marginaleffects_numDeriv = NULL)

See the "Uncertainty" chapter on the marginaleffects website for more details on the computation of standard errors, bootstrapping, and more:

https://marginaleffects.com/chapters/uncertainty.html

Model-Specific Arguments

Some model types allow model-specific arguments to modify the nature of marginal effects, predictions, marginal means, and contrasts. Please report other package-specific predict() arguments on Github so we can add them to the table below.

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

Package	Class	Argument	Documentation
brms	brmsfit	ndraws	brms::posterior_predict()
		re_formula	brms::posterior_predict()
lme4	merMod	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()
	gam	exclude	mgcv::predict.gam()
robustlmm	rlmerMod	re.form	robustlmm::predict.rlmerMod()
		allow.new.levels	robustlmm::predict.rlmerMod()
MCMCglmm	MCMCglmm	ndraws	
sampleSelection	selection	part	sampleSelection::predict.selection()

Bayesian posterior summaries

By default, credible intervals in bayesian models are built as equal-tailed intervals. This can be changed to a highest density interval by setting a global option:

options("marginaleffects_posterior_interval" = "eti")

options("marginaleffects_posterior_interval" = "hdi")

By default, the center of the posterior distribution in bayesian models is identified by the median. Users can use a different summary function by setting a global option:

options("marginaleffects_posterior_center" = "mean")

options("marginaleffects_posterior_center" = "median")

When estimates are averaged using the by argument, the `tidy()` function, or the `summary()` function, the posterior distribution is marginalized twice over. First, we take the average *across* units but *within* each iteration of the MCMC chain, according to what the user requested in by argument or `tidy()/summary()` functions. Then, we identify the center of the resulting posterior using the function supplied to the `"marginaleffects_posterior_center"` option (the median by default).

Equivalence, Inferiority, Superiority

θ is an estimate, σ_θ its estimated standard error, and $[a, b]$ are the bounds of the interval supplied to the `equivalence` argument.

Non-inferiority:

- H_0: $\theta \leq a$
- H_1: $\theta > a$
- $t = (\theta - a)/\sigma_\theta$
- p: Upper-tail probability

Non-superiority:

- H_0: $\theta \geq b$
- H_1: $\theta < b$
- $t = (\theta - b)/\sigma_\theta$
- p: Lower-tail probability

Equivalence: Two One-Sided Tests (TOST)

- p: Maximum of the non-inferiority and non-superiority p values.

Thanks to Russell V. Lenth for the excellent `emmeans` package and documentation which inspired this feature.

Parallel computation

The `slopes()` and `comparisons()` functions can use parallelism to speed up computation. Operations are parallelized for the computation of standard errors, at the model coefficient level. There is always considerable overhead when using parallel computation, mainly involved in passing the whole dataset to the different processes. Thus, parallel computation is most likely to be useful when the model includes many parameters and the dataset is relatively small.

Warning: In many cases, parallel processing will not be useful at all.

To activate parallel computation, users must load the `future.apply` package, call `plan()` function, and set a global option.

`options(marginaleffects_parallel = TRUE)`: parallelize delta method computation of standard errors. `options(marginaleffects_parallel_inferences = TRUE)`: parallelize `"rsample"` or `"fwb"` bootstrap computation in `inferences()`. `options(marginaleffects_parallel_packages = TRUE)`: vector of strings with the names of modeling packages used to fit the model, ex: c("survival", "splines")

For example:

```
library(future.apply)
plan("multisession", workers = 4)
options(marginaleffects_parallel = FALSE)
options(marginaleffects_parallel_inferences = TRUE)
options(marginaleffects_parallel_packages = c("survival", "splines"))

slopes(model)
```

To disable parallelism in `marginaleffects` altogether, you can set a global option:

```
options(marginaleffects_parallel = FALSE)
```

Order of operations

Behind the scenes, the arguments of `marginaleffects` functions are evaluated in this order:

1. `newdata`
2. `variables`
3. `comparison` and `slope`
4. `by`
5. `vcov`
6. `hypothesis`
7. `transform`

Global options

The behavior of `marginaleffects` functions can be modified by setting global options.

Disable some safety checks and warnings:

- `options(marginaleffects_startup_message = FALSE)`
  - Disable the startup message printed on `library(marginaleffects)`.
- `options(marginaleffects_safe = FALSE)`
  - Disable safety checks and warnings.
- `options(marginaleffects_print_omit = c("p.value", "s.value"))`
  - Omit some columns from the printed output.

Enforce lean return objects, sans information about the original model and data, and other ancillary attributes. Note that this will disable some advanced post-processing features and functions like hypotheses.

```
options(marginaleffects_lean = TRUE)
```

Other options:

- `marginaleffects_plot_gray`: logical. If `TRUE`, the default color of the plot is gray. Default is `FALSE`.

Types

The `type` argument determines the scale of the predictions used to compute quantities of interest with functions from the `marginaleffects` package. Admissible values for `type` depend on the model object. When users specify an incorrect value for `type`, `marginaleffects` will raise an informative error with a list of valid `type` values for the specific model object. The first entry in the list in that error message is the default type.

The `invlink(link)` is a special type defined by `marginaleffects`. It is available for some (but not all) models, and only for the `predictions()` function. With this link type, we first compute predictions on the link scale, then we use the inverse link function to backtransform the predictions to the response scale. This is useful for models with non-linear link functions as it can ensure that confidence intervals stay within desirable bounds, ex: 0 to 1 for a logit model. Note that an average of estimates with `type="invlink(link)"` will not always be equivalent to the average of estimates with `type="response"`. This type is default when calling `predictions()`. It is available—but not default—when calling `avg_predictions()` or `predictions()` with the `by` argument.

Some of the most common `type` values are:

class	type
DirichletRegModel	response
Gam	invlink(link), response, link
Gls	lp
MCMCglmm	response
aft	surv, haz, cumhaz, density, odds, link
bam	response, link
bart	ev, ppd
betareg	response, link, precision, quantile, variance
bife	response, link
bracl	probs
brglmFit	response, link
brmsfit	response, link, prediction, average
brmultinom	probs, class
clm	prob, cum.prob, linear.predictor
clmm2	prob
clogit	expected, lp, risk, survival
coxph	survival, expected, lp, risk
coxph_weightit	survival, expected, lp, risk
crch	response, location, scale, density
fixest	invlink(link), response, link
flexsurvreg	survival, response, mean, link, lp, linear, rmst, hazard, cumhaz
gam	response, link
geeglm	response, link
glimML	response, link
glm	invlink(link), response, link
glm_weightit	invlink(link), probs, response, lp, link
glmerMod	response, link
glmgee	response, link
glmmPQL	response, link
glmmTMB	response, link, conditional, zprob, zlink, disp

glmrob	response, link
glmx	response
gnm	response, link
hetprob	pr, xb
hurdle	response, prob, count, zero
hxlr	location, cumprob, scale, density
iv_robust	response
ivpml	pr, xb
ivreg	response
lda	class, posterior
lm	response
lm_robust	response
lmerMod	response
lmerModLmerTest	response
lmrob	response
lrm	fitted, lp, mean
mblogit	response, latent, link
mclogit	response, latent, link
mhurdle	E, Ep, p
model_fit	numeric, prob, class
multinom	probs, latent
multinom_weightit	probs, response, mean
mvgam	response, link, expected, detection, latent_N
negbin	invlink(link), response, link
nestedLogit	response, link
ols	lp
oohbchoice	probability, utility
ordinal_weightit	probs, response, link, lp, mean
orm	fitted, mean, lp
polr	probs
rendo.base	response, link
rlm	response
selection	response, link, unconditional, conditional
speedglm	response, link
speedlm	response
stanreg	response, link
survreg	response, link, quantile
svy_vglm	response, link
svyglm	response, link
svyolr	probs
tobit	response, link
tobit1	expvalue, linpred, prob
workflow	numeric, prob, class
zeroinfl	response, prob, count, zero

References

- Arel-Bundock V, Greifer N, Heiss A (2024). "How to Interpret Statistical Models Using marginaleffects for R and Python." *Journal of Statistical Software*, *111*(9), 1-32. doi:10.18637/jss.v111.i09 doi:10.18637/jss.v111.i09
- Arel-Bundock (2026). "Model to Meaning: How to interpret statistical models in R and Python." CRC Press. https://routledge.com/9781032908724
- Greenland S. 2019. "Valid P-Values Behave Exactly as They Should: Some Misleading Criticisms of P-Values and Their Resolution With S-Values." The American Statistician. 73(S1): 106–114.
- Cole, Stephen R, Jessie K Edwards, and Sander Greenland. 2020. "Surprise!" American Journal of Epidemiology 190 (2): 191–93. doi:10.1093/aje/kwaa136
- Hansen SN, Overgaard M (2025). "Variance Estimation for Average Treatment Effects Estimated by G-Computation." *Metrika*, *88*(4), 419–443. doi:10.1007/s00184024009624

Examples

```
library("marginaleffects")
# Unit-level (conditional) Marginal Effects
mod <- glm(am ~ hp * wt, data = mtcars, family = binomial)
mfx <- slopes(mod)
head(mfx)

# Average Marginal Effect (AME)
avg_slopes(mod, by = TRUE)


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

# Marginal Effect at User-Specified Values
# Variables not explicitly included in `datagrid()` are held at their means
slopes(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)
avg_slopes(mod2, variables = "hp", by = "cyl")

# 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 <- slopes(mod,
    newdata = datagrid(
        hp = c(100, 110),
        grid_type = "counterfactual"))
head(mfx)
```

```
# Heteroskedasticity robust standard errors
mfx <- slopes(mod, vcov = sandwich::vcovHC(mod))
head(mfx)

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

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

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

# same hypothesis test using numeric vector of weights
slopes(
    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")
slopes(
    mod,
    newdata = "mean",
    hypothesis = lc)
```

`vcovUnconditional` *EXPERIMENTAL: Request unconditional variance in marginaleffects calls*

Description

EXPERIMENTAL: Request unconditional variance in marginaleffects calls

Usage

```
vcovUnconditional(type = "HC0", cluster = NULL)
```

Arguments

type	Character string specifying the finite-sample adjustment. The available types are "HC0" and "HC1". "HC0" uses the raw plug-in covariance, while "HC1" multiplies the complete unconditional covariance by a conventional model degrees-of-freedom factor analogous to the one used by the sandwich package.
cluster	An optional one-sided formula such as ~id identifying the variable used for one-way clustered inference. The right-hand side must be a bare variable name; transformations and multiple variables are not supported. As in the default behavior of sandwich::vcovCL(), clustered estimates include the cluster-count adjustment G / (G - 1).

Details

The bare function form vcov = vcovUnconditional is equivalent to vcov = vcovUnconditional().

Without clustering, HC0 applies no finite-sample multiplier and HC1 applies n / (n - k). With clustering, HC0 applies G / (G - 1) and HC1 applies G / (G - 1) * (n - 1) / (n - k), where k is the dimension of the model's estimating-function system and G is the number of clusters. These multipliers match the corresponding sandwich adjustments. They are applied after the coefficient-estimation and empirical-distribution influence components have been combined. HC1 therefore scales the variance of both components and their cross-covariance; it does not correct only the first-stage model-score contribution. HC0 is the plug-in estimator derived by Hansen and Overgaard. Applying the HC1 multiplier to the complete unconditional influence function is a documented convention, not a target-level correction derived in that paper, and it is not generally an unbiased finite-sample correction. Because its factor depends on k, HC1 can differ across model specifications even when they produce the same averaged estimand. The df argument of the calling marginaleffects function controls the reference distribution for inference separately and does not determine these covariance multipliers.

Unconditional variance is available for effects evaluated over original model-data rows, valid subsets of those rows, or counterfactual grids that preserve a valid rowid/rowidcf mapping to original model-data rows. The effect must be averaged or aggregated with avg_*() or by, or it must be a scalar comparison. Hypotheses applied directly to unit-level effects are rejected because the empirical-distribution influence function is not identifiable from an arbitrary post-hoc hypothesis function. Synthetic grids such as newdata = "mean" are rejected because the current implementation cannot generally infer how those grid values vary with the empirical covariate distribution. Models must provide compatible score and bread matrices through sandwich::estfun()/sandwich::bread() or model-specific equivalents. Supported model classes are validated through an allow-list. Survey-weighted linear and generalized linear models fitted by survey::svyglm() are supported using their observation-level coefficient influence functions. Propensity-score-weighted models fitted by WeightIt::glm_weightit() or WeightIt::lm_weightit() are supported because their estfun() and bread() methods already account for estimation of the weights; the multi-equation multinom_weightit, ordinal_weightit, and coxph_weightit classes are not supported. For multiple-imputation objects, unconditional variance is estimated in each completed dataset and the results are pooled using Rubin's rules. Prediction methods that return posterior draws, censored and survival models such as tobit, survreg, and coxph, average predictions from fixest models with fixed effects, and nonlinear fixest models with fixed effects are rejected explicitly. For feols models with fixed effects, unconditional inference is available for additive differences and dydx/dyex slopes when the counterfactual data leave every fixed-effect and varying-slope variable unchanged. The combined

influence function retains the covariance between coefficient estimation and the empirical covariate distribution, which is one ingredient of robustness to conditional-mean misspecification. That robustness also requires the model's score and bread methods to represent the derivative of its full estimating equations; this is not guaranteed for every supported model and link under misspecification. HC2 through HC5 are not available because their regression-leverage adjustments are not defined for this combined influence function.

Value

An object which can be supplied to the vcov argument of `avg_predictions()`, `avg_comparisons()`, or `avg_slopes()`.

Index

avg_comparisons *(*comparisons*)*, 4
avg_comparisons(), *87*
avg_predictions *(*predictions*)*, 53
avg_predictions(), *87*
avg_slopes *(*slopes*)*, 71
avg_slopes(), *87*

comparisons, 4
components.marginaleffects, 21

datagrid, *6*, 22
datagrid(), *6*, *55*, *73*
dplyr::filter(), *6*, *55*, *73*

get_dataset, 25
get_draws, 26
get_modeldata, 27
get_modeldata(), *5*, *24*, *55*, *72, 73*

hypotheses, *16*, 28, *28*, *64*, *81*

inferences, 35
insight::get_df, *11*, *31*, *59*, *76*

plot_comparisons, 39
plot_predictions, 43
plot_slopes, 49
predictions, 53
print.marginaleffects, 68
prune.marginaleffects, 70

refit, 70

sandwich::vcovCL(), *86*
set_modeldata *(*get_modeldata*)*, 27
set_modeldata(), *5*, *24*, *55*, *72*
slopes, 71
subset(), *6*, *55*, *73*
survey::svyglm(), *86*

vcovUnconditional, 85

weights(), *37*