

# Package: clarify (via r-universe)

June 8, 2026

**Type** Package

**Title** Simulation-Based Inference for Regression Models

**Version** 0.2.3

**Description** Performs simulation-based inference as an alternative to the delta method for obtaining valid confidence intervals and p-values for regression post-estimation quantities, such as average marginal effects and predictions at representative values. This framework for simulation-based inference is especially useful when the resulting quantity is not normally distributed and the delta method approximation fails. The methodology is described in Greifer, et al. (2025) [doi:10.32614/RJ-2024-015](https://doi.org/10.32614/RJ-2024-015). 'clarify' is meant to replace some of the functionality of the archived package 'Zelig'; see the vignette ``Translating Zelig to clarify'' for replicating this functionality.

**License** GPL (>= 3)

**Encoding** UTF-8

**Depends** R (>= 4.1.0)

**Imports** arg (>= 0.1.1), ggplot2 (>= 3.5.0), pbapply (>= 1.7-4), cli (>= 3.6.5), rlang (>= 1.1.6), insight (>= 1.4.2), marginaleffects (>= 0.30.0)

**Suggests** testthat (>= 3.0.0), withr, MatchIt (>= 4.0.0), parallel, knitr, rmarkdown, Amelia, MASS, betareg, survey, estimatr, fixest, logistf, geepack, rms, robustbase, robust, AER, ivreg, mgcv, sandwich, WeightIt (>= 1.3.2)

**Config/testthat/edition** 3

**URL** <https://github.com/iqss/clarify>, <https://iqss.github.io/clarify/>

**BugReports** <https://github.com/iqss/clarify/issues>

**VignetteBuilder** knitr

**Config/roxygen2/version** 8.0.0

**NeedsCompilation** no

**Author** Noah Greifer [aut, cre] (ORCID: <https://orcid.org/0000-0003-3067-7154>), Steven Worthington [aut] (ORCID: <https://orcid.org/0000-0001-9550-5797>), Stefano Iacus [aut] (ORCID: <https://orcid.org/0000-0002-4884-0047>), Gary King [aut] (ORCID: <https://orcid.org/0000-0002-5327-7631>)

**Maintainer** Noah Greifer <ngreifer@iq.harvard.edu>

**Repository** <https://cran.r-universe.dev>

**Date/Publication** 2026-06-08 20:50:02 UTC

**RemoteUrl** <https://github.com/cran/clarify>

**RemoteRef** HEAD

**RemoteSha** 7352bc7993345f87bb9e1a0270a97cccfec5ff1b

## Contents

misim . . . . .	2
plot.clarify_adrf . . . . .	4
plot.clarify_est . . . . .	5
plot.clarify_setx . . . . .	9
sim . . . . .	10
sim_adrf . . . . .	12
sim_ame . . . . .	16
sim_apply . . . . .	20
sim_setx . . . . .	23
transform.clarify_est . . . . .	26
<b>Index</b>	<b>29</b>

---

misim

*Simulate model coefficients after multiple imputation*

---

## Description

`misim()` simulates model parameters from multivariate normal or t distributions after multiple imputation that are then used by `sim_apply()` to calculate quantities of interest.

## Usage

```
misim(fitlist, n = 1000, vcov = NULL, coefs = NULL, dist = NULL)
```

## Arguments

<code>fitlist</code>	a list of model fits, one for each imputed dataset, or a <code>&lt;mira&gt;</code> object (the output of a call to <code>with()</code> applied to a <code>&lt;mids&gt;</code> object from <i>mice</i> ).
<code>n</code>	the number of simulations to run for each imputed dataset; default is 1000. More is always better but resulting calculations will take longer.
<code>vcov</code>	a square covariance matrix of the coefficient covariance estimates, a function to use to extract it from <code>fit</code> , or a list thereof with an element for each imputed dataset. By default, uses <code>stats::vcov()</code> or <code>insight::get_varcov()</code> if that doesn't work.
<code>coefs</code>	a vector of coefficient estimates, a function to use to extract it from <code>fit</code> , or a list thereof with an element for each imputed dataset. By default, uses <code>stats::coef()</code> or <code>insight::get_parameters()</code> if that doesn't work.
<code>dist</code>	a character vector containing the name of the multivariate distribution(s) to use to draw simulated coefficients. Should be one of "normal" (multivariate normal distribution) or "t_{#}" (multivariate t distribution), where {#} corresponds to the desired degrees of freedom (e.g., "t_100"). If NULL, the right distributions to use will be determined based on heuristics; see <code>sim()</code> for details.

## Details

`misim()` essentially combines multiple `sim()` calls applied to a list of model fits, each fit in an imputed dataset, into a single combined pool of simulated coefficients. When simulation-based inference is to be used with multiply imputed data, many imputations are required; see Zhou and Reiter (2010).

## Value

A `<clarify_misim>` object, which inherits from `<clarify_sim>` and has the following components:

<code>sim.coefs</code>	a matrix containing the simulated coefficients with a column for each coefficient and a row for each simulation for each imputation
<code>coefs</code>	a matrix containing the original coefficients extracted from <code>fitlist</code> or supplied to <code>coefs</code> , with a row per imputation.
<code>fit</code>	the list of model fits supplied to <code>fitlist</code>
<code>imp</code>	a identifier of which imputed dataset each set of simulated coefficients corresponds to.

The "dist" attribute contains "normal" if the coefficients were sampled from a multivariate normal distribution and "t({df})" if sampled from a multivariate t distribution. The "clarify\_hash" attribute contains a unique hash generated by `rlang::hash()`.

## References

Zhou, X., & Reiter, J. P. (2010). A Note on Bayesian Inference After Multiple Imputation. *The American Statistician*, 64(2), 159–163. doi:10.1198/tast.2010.09109

**See Also**

- `sim()` for simulating model coefficients for a single dataset
- `sim_apply()` for applying a function to each set of simulated coefficients
- `sim_ame()` for computing average marginal effects in each simulation draw
- `sim_setx()` for computing marginal predictions and first differences at typical values in each simulation draw

**Examples**

```
data("africa", package = "Amelia")

# Multiple imputation using Amelia
a.out <- Amelia::amelia(x = africa, m = 10,
                       cs = "country",
                       ts = "year", logs = "gdp_pc",
                       p2s = 0)

fits <- with(a.out, lm(gdp_pc ~ infl * trade))

# Simulate coefficients
s <- misim(fits)
s
```

---

`plot.clarify_adrf`      *Plot marginal predictions from sim\_adrf()*

---

**Description**

`plot.clarify_adrf()` plots the output of `sim_adrf()`. For the average dose-response function (ADRF, requested with `contrast = "adrf"` in `sim_adrf()`), this is a plot of the average marginal mean of the outcome against the requested values of the focal predictor; for the average marginal effects function (AMEF, requested with `contrast = "amef"` in `sim_adrf()`), this is a plot of the instantaneous average marginal effect of the focal predictor on the outcome against the requested values of the focal predictor.

**Usage**

```
## S3 method for class 'clarify_adrf'
plot(
  x,
  ci = TRUE,
  level = 0.95,
  method = "quantile",
  baseline = NULL,
  color = "black",
  simultaneous = FALSE,
  ...
)
```

**Arguments**

x	a <clarify_adrf> object resulting from a call to <a href="#">sim_adrf()</a> .
ci	logical; whether to display confidence bands for the estimates. Default is TRUE.
level	the confidence level desired. Default is .95 for 95% confidence intervals.
method	the method used to compute confidence bands. Can be "wald" to use a Normal approximation or "quantile" to use the simulated sampling distribution (default). See <a href="#">summary.clarify_est()</a> for details. Abbreviations allowed.
baseline	logical; whether to include a horizontal line at $y = 0$ on the plot. Default is FALSE for the ADRF (since 0 might not be in the range of the outcome) and TRUE for the AMEF.
color	the color of the line and confidence band in the plot.
simultaneous	logical; whether confidence bands should be simultaneous or not (i.e., for nominal coverage of the whole effect curve); default is FALSE, but TRUE is recommended. See Details at <a href="#">summary.clarify_est()</a> for details.
...	for <a href="#">plot()</a> , further arguments passed to <a href="#">ggplot2::geom_density()</a> .

**Details**

These plots are produced using [ggplot2::geom\\_line\(\)](#) and [ggplot2::geom\\_ribbon\(\)](#). The confidence bands should be interpreted pointwise (i.e., they do not account for simultaneous inference) unless `simultaneous = TRUE`.

**Value**

A <ggplot> object.

**See Also**

[summary.clarify\\_est\(\)](#) for computing p-values and confidence intervals for the estimated quantities.

**Examples**

```
## See help("sim_adrf") for examples
```

---

plot.clarify\_est      *Plotting and inference for clarify\_est objects*

---

**Description**

[summary\(\)](#) tabulates the estimates and confidence intervals and (optionally) p-values from a <clarify\_est> object. [confint\(\)](#) computes confidence intervals. [plot\(\)](#) plots the "posterior" distribution of estimates.

**Usage**

```
## S3 method for class 'clarify_est'
plot(
  x,
  parm,
  ci = TRUE,
  level = 0.95,
  method = "quantile",
  reference = FALSE,
  ncol = 3L,
  simultaneous = FALSE,
  ...
)

## S3 method for class 'clarify_est'
summary(
  object,
  parm,
  level = 0.95,
  method = "quantile",
  null = NA,
  simultaneous = FALSE,
  ...
)

## S3 method for class 'clarify_est'
confint(
  object,
  parm,
  level = 0.95,
  method = "quantile",
  simultaneous = FALSE,
  ...
)
```

**Arguments**

parm	a vector of the names or indices of the estimates to plot. If unspecified, all estimates will be displayed.
ci	logical; whether to display confidence interval limits for the estimates. Default is TRUE.
level	the confidence level desired. Default is .95 for 95% confidence intervals.
method	the method used to compute p-values and confidence intervals. Can be "wald" to use a Normal approximation or "quantile" to use the simulated sampling distribution (default). See Details. Abbreviations allowed.
reference	logical; whether to overlay a normal density reference distribution over the plots. Default is FALSE.

<code>ncol</code>	the number of columns used when wrapping multiple plots; default is 3.
<code>simultaneous</code>	logical; whether confidence intervals and p-values should be simultaneous or not (i.e., adjusted for multiple comparisons); default is FALSE. See Details.
<code>...</code>	for <code>plot()</code> , further arguments passed to <code>ggplot2::geom_density()</code> .
<code>object, x</code>	a <code>&lt;clarify_est&gt;</code> object; the output of a call to <code>sim_apply()</code> or its wrappers.
<code>null</code>	the values of the parameters under the null hypothesis for the p-value calculations. Should have length equal to the number of quantities estimated, or one, in which case it will be recycled, or it can be a named vector with just the names of quantities for which null values are to be set. Set values to NA to omit p-values for those quantities. When all values are NA, the default, no p-values are produced.

## Details

`summary()` uses the estimates computed from the original model as its estimates and uses the simulated parameters for inference only, in line with the recommendations of Rainey (2023).

When `method = "wald"`, the standard deviation of the simulation estimates is used as the standard error, which is used in the z-statistics and the confidence intervals. The p-values and confidence intervals are valid only when the sampling distribution of the resulting statistic is normal (which can be assessed using `plot()`). When `method = "quantile"`, the confidence interval is calculated using the quantiles of the simulation estimates corresponding to `level`, and the p-value is calculated as twice the proportion of simulation estimates less than or greater than `null`, whichever is smaller; this is equivalent to inverting the confidence interval but is only truly valid when the true sampling distribution is only a location shift from the sampling distribution under the null hypothesis and should therefore be interpreted with caution. Using `method = "quantile"` (the default) is recommended because the confidence intervals will be valid even if the sampling distribution is not Normally distributed. The precision of the p-values and confidence intervals depends on the number of simulations requested (the value of `n` supplied to `sim()`).

When `simultaneous = TRUE`, confidence intervals and p-values are adjusted to account for multiple comparisons using the "sup-t" confidence region and its inversion. The sup-t confidence region is the smallest rectangular region that ensures simultaneous coverage of all parameters at the desired confidence level. It is found by adjusting the nominal confidence level until a new level is found that, when simultaneously applied to all estimates, yields intervals that contain all estimates with rates equal to the original level. Unlike some other adjustments for multiple comparisons (e.g., Holm, Bonferroni, Benjamini-Hochberg), this method takes into account the joint distribution of the estimates, often yielding narrower regions that are less conservative (but still valid). P-values are found by finding the level of the narrowest band that guarantees simultaneous coverage while containing the null value of the given parameter. When `method = "quantile"`, the Bayesian algorithm described by Montiel Olea and Plagborg-Møller (2019) is used with the simulated estimates; when `method = "wald"`, a new simulation is performed treating the estimates as coming from a multivariate normal distribution.

The plots are produced using `ggplot2::geom_density()` and can be customized with `ggplot2` functions. When `reference = TRUE`, a reference Normal distribution is produced using the empirical mean and standard deviation of the simulated values. A blue reference line is plotted at the median of the simulated values. For Wald-based inference to be valid, the reference distribution should overlap with the empirical distribution, in which case the quantile-based and Wald-based

intervals should be similar. For quantile-based inference to be valid, the median of the estimates should overlap with the estimated value; this is a necessary but not sufficient condition, though.

### Value

For `summary()`, a `<summary.clarify_est>` object, which is a matrix containing the coefficient estimates, standard errors, test statistics, p-values, and confidence intervals. Not all columns will be present depending on the arguments supplied to `summary()`.

For `confint()`, a matrix containing the confidence intervals for the requested quantities.

For `plot()`, a `<ggplot>` object.

### References

Montiel Olea, J. L., & Plagborg-Møller, M. (2019). Simultaneous confidence bands: Theory, implementation, and an application to SVARs. *Journal of Applied Econometrics*, 34(1), 1–17. doi:10.1002/jae.2656

Rainey, C. (2023). A careful consideration of CLARIFY: Simulation-induced bias in point estimates of quantities of interest. *Political Science Research and Methods*, 1–10. doi:10.1017/psrm.2023.8

### See Also

- `sim_apply()` for applying a function to each set of simulated coefficients

### Examples

```
data("lalonde", package = "MatchIt")
fit <- glm(I(re78 > 0) ~ treat + age + race + nodegree + re74,
          data = lalonde)

s <- sim(fit, n = 100)

# Compute average marginal means for `treat`
est <- sim_ame(s, var = "treat")
coef(est)

# Compute average marginal effects on risk difference
# (RD) and risk ratio (RR) scale
est <- transform(est,
                 RD = `E[Y(1)]` - `E[Y(0)]`,
                 RR = `E[Y(1)]` / `E[Y(0)]`)

# Compute confidence intervals and p-values,
# using given null values for computing p-values
summary(est, null = c(`RD` = 0, `RR` = 1))

# Same tests using normal approximation and alternate
# syntax for `null`
summary(est, null = c(NA, NA, 0, 1),
        normal = TRUE)

# Plot the RD and RR with a reference distribution
```

```
plot(est, parm = c("RD", "RR"), reference = TRUE,
      ci = FALSE)

# Plot the RD and RR with quantile confidence bounds
plot(est, parm = c("RD", "RR"), ci = TRUE)
```

---

plot.clarify\_setx      *Plot marginal predictions from sim\_setx()*

---

## Description

plot.clarify\_sext() plots the output of [sim\\_setx\(\)](#), providing graphics similar to those of [plot.clarify\\_est\(\)](#) but with features specifically for plot marginal predictions. For continuous predictors, this is a plot of the marginal predictions and their confidence bands across levels of the predictor. Otherwise, this is a plot of simulated sampling distribution of the marginal predictions.

## Usage

```
## S3 method for class 'clarify_setx'
plot(
  x,
  var = NULL,
  ci = TRUE,
  level = 0.95,
  method = "quantile",
  reference = FALSE,
  simultaneous = FALSE,
  ...
)
```

## Arguments

x	a <clarify_est> object resulting from a call to <a href="#">sim_setx()</a> .
var	the name of the focal varying predictor, i.e., the variable to be on the x-axis of the plot. All other variables with varying set values will be used to color the resulting plot. See Details. Ignored if no predictors vary or if only one predictor varies in the reference grid or if x1 was specified in <a href="#">sim_setx()</a> . If not set, will use the predictor with the greatest number of unique values specified in the reference grid.
ci	logical; whether to display confidence intervals or bands for the estimates. Default is TRUE.
level	the confidence level desired. Default is .95 for 95% confidence intervals.
method	the method used to compute confidence intervals or bands. Can be "wald" to use a Normal approximation or "quantile" to use the simulated sampling distribution (default). See <a href="#">summary.clarify_est()</a> for details. Abbreviations allowed.

reference	logical; whether to overlay a normal density reference distribution over the plots. Default is FALSE. Ignored when variables other than the focal varying predictor vary.
simultaneous	logical; whether confidence bands should be simultaneous or not (i.e., for nominal coverage of the whole effect curve); default is FALSE, but TRUE is recommended. See Details at <a href="#">summary.clarify_est()</a> for details.
...	for <code>plot()</code> , further arguments passed to <code>ggplot2::geom_density()</code> .

### Details

`plot()` creates one of two kinds of plots depending on how the reference grid was specified in the call to `sim_setx()` and what `var` is set to. When the focal varying predictor (i.e., the one set in `var`) is numeric and takes on three or more unique values in the reference grid, the produced plot is a line graph displaying the value of the marginal prediction (denoted as  $E[Y|X]$ ) across values of the focal varying predictor, with confidence bands displayed when `ci = TRUE`. If other predictors also vary, lines for different values will be displayed in different colors. These plots are produced using `ggplot2::geom_line()` and `ggplot2::geom_ribbon()`.

When the focal varying predictor is a factor or character or only takes on two or fewer values in the reference grid, the produced plot is a density plot of the simulated predictions, similar to the plot resulting from `plot.clarify_est()`. When other variables vary, densities for different values will be displayed in different colors. These plots are produced using `ggplot2::geom_density()`.

Marginal predictions are identified by the corresponding levels of the predictors that vary. The user should keep track of whether the non-varying predictors are set at specified or automatically set "typical" levels.

### Value

A `<ggplot>` object.

### See Also

[summary.clarify\\_est\(\)](#) for computing p-values and confidence intervals for the estimated quantities.

### Examples

```
## See help("sim_setx") for examples
```

---

sim

*Simulate model parameters*

---

### Description

`sim()` simulates model parameters from a multivariate normal or t distribution that are then used by `sim_apply()` to calculate quantities of interest.

**Usage**

```
sim(fit, n = 1000L, vcov = NULL, coefs = NULL, dist = NULL)
```

**Arguments**

<code>fit</code>	a model fit, such as the output of a call to <code>lm()</code> or <code>glm()</code> . Can be left unspecified if <code>coefs</code> and <code>vcov</code> are not functions.
<code>n</code>	the number of simulations to run; default is 1000. More is always better but resulting calculations will take longer.
<code>vcov</code>	either a square covariance matrix of the coefficient covariance estimates or a function to use to extract it from <code>fit</code> . By default, uses <code>stats::vcov()</code> or <code>insight::get_varcov()</code> if that doesn't work.
<code>coefs</code>	either a vector of coefficient estimates or a function to use to extract it from <code>fit</code> . By default, uses <code>stats::coef()</code> or <code>insight::get_parameters()</code> if that doesn't work.
<code>dist</code>	a string containing the name of the multivariate distribution to use to draw simulated coefficients. Should be one of "normal" (multivariate normal distribution) or "t({#})" (multivariate t distribution), where {#} corresponds to the desired degrees of freedom (e.g., "t(100)"). If NULL, the right distribution to use will be determined based on heuristics; see Details.

**Details**

When `dist` is NULL, `sim()` samples from a multivariate normal or t distribution depending on the degrees of freedom extracted from `insight::get_df(., type = "wald")`. If Inf, a normal distribution will be used; otherwise, a t-distribution with the returned degrees of freedom will be used. Models not supported by `insight` will use a normal distribution.

**Value**

A `<clarify_sim>` object, which has the following components:

<code>sim.coefs</code>	a matrix containing the simulated coefficients with a column for each coefficient and a row for each simulation
<code>coefs</code>	the original coefficients extracted from <code>fit</code> or supplied to <code>coefs</code> .
<code>vcov</code>	the covariance matrix of the coefficients extracted from <code>fit</code> or supplied to <code>vcov</code>
<code>fit</code>	the original model fit supplied to <code>fit</code>

The "dist" attribute contains "normal" if the coefficients were sampled from a multivariate normal distribution and "t(df)" if sampled from a multivariate t distribution. The "clarify\_hash" attribute contains a unique hash generated by `rlang::hash()`.

**See Also**

- `misim()` for simulating model coefficients after multiple imputation
- `sim_apply()` for applying a function to each set of simulated coefficients
- `sim_ame()` for computing average marginal effects in each simulation draw

- `sim_setx()` for computing marginal predictions and first differences at typical values in each simulation draw
- `sim_adrf()` for computing average dose-response functions in each simulation draw

## Examples

```
data("lalonde", package = "MatchIt")
fit <- lm(re78 ~ treat * (age + race + nodegree + re74),
         data = lalonde)

# Simulate coefficients
s <- sim(fit)
s

## Could also use a robust covariance matrix, e.g.,
s <- sim(fit, vcov = "HC3")

# Simulated coefficients assuming a normal distribution
# for coefficients; default for `lm` objects is a t-
# distribution
s <- sim(fit, dist = "normal")
s
```

---

sim\_adrf

*Compute an average dose-response function*

---

## Description

`sim_adrf()` is a wrapper for `sim_apply()` that computes average dose-response functions (ADRFs) and average marginal effect functions (AMEFs). An ADRF describes the relationship between values a focal variable can take and the expected value of the outcome were all units to be given each value of the variable. An AMEF describes the relationship between values a focal variable can take and the derivative of ADRF at each value.

## Usage

```
sim_adrf(
  sim,
  var,
  subset = NULL,
  by = NULL,
  contrast = "adrf",
  at = NULL,
  n = 21,
  outcome = NULL,
  type = NULL,
  eps = 1e-05,
```

```

    verbose = interactive(),
    cl = NULL,
    ...
)

## S3 method for class 'clarify_adrf'
print(x, digits = 4L, max.ests = 6L, ...)

```

## Arguments

sim	a <clarify_sim> object; the output of a call to <a href="#">sim()</a> or <a href="#">misim()</a> .
var	the name of a variable for which the ADRF or AMEF is to be computed. This variable must be present in the model supplied to <a href="#">sim()</a> and must be a numeric variable taking on more than two unique values.
subset	optional; a vector used to subset the data used to compute the ADRF or AMEF. This will be evaluated within the original dataset used to fit the model using <a href="#">subset()</a> , so nonstandard evaluation is allowed.
by	a one-sided formula or character vector containing the names of variables for which to stratify the estimates. Each quantity will be computed within each level of the complete cross of the variables specified in by.
contrast	a string naming the type of quantity to be produced: "adrf" for the ADRF (the default) or "amef" for the AMEF.
at	the levels of the variable named in var at which to evaluate the ADRF or AMEF. Should be a vector of numeric values corresponding to possible levels of var. If NULL, will be set to a range from slightly below the lowest observed value of var to slightly above the largest value.
n	when at = NULL, the number of points to evaluate the ADRF or AMEF. Default is 21. Ignored when at is not NULL.
outcome	a string containing the name of the outcome or outcome level for multivariate (multiple outcomes) or multi-category outcomes. Ignored for univariate (single outcome) and binary outcomes.
type	a string containing the type of predicted values (e.g., the link or the response). Passed to <a href="#">marginaleffects::get_predict()</a> and eventually to <a href="#">predict()</a> in most cases. The default and allowable option depend on the type of model supplied, but almost always corresponds to the response scale (e.g., predicted probabilities for binomial models).
eps	when contrast = "amef", the value by which to shift the value of var to approximate the derivative. See Details.
verbose	logical; whether to display a text progress bar indicating progress and estimated time remaining for the procedure. Default is TRUE for interactive sessions and FALSE otherwise.
cl	a cluster object created by <a href="#">parallel::makeCluster()</a> , or an integer to indicate the number of child-processes (integer values are ignored on Windows) for parallel evaluations. See <a href="#">pbapply::pblapply()</a> for details. If NULL, no parallelization will take place.

...	for <code>sim_adrf()</code> , additional arguments passed to <code>marginalEffects::get_predict()</code> (and eventually to <code>predict()</code> ) to compute predictions. For <code>print()</code> , ignored.
x	a <code>&lt;clarify_adrf&gt;</code> object.
digits	the minimum number of significant digits to be used; passed to <code>print.data.frame()</code> .
max.est	the maximum number of estimates to display.

## Details

The ADRF is composed of average marginal means across levels of the focal predictor. For each level of the focal predictor, predicted values of the outcome are computed after setting the value of the predictor to that level, and those values of the outcome are averaged across all units in the sample to arrive at an average marginal mean. Thus, the ADRF represent the relationship between the "dose" (i.e., the level of the focal predictor) and the average "response" (i.e., the outcome variable). It is the continuous analog to the average marginal effect computed for a binary predictor, e.g., using `sim_ame()`. Although inference can be at each level of the predictor or between two levels of the predictor, typically a plot of the ADRF is the most useful relevant quantity. These can be requested using `plot.clarify_adrf()`.

The AMEF is the derivative of the ADRF; if we call the derivative of the ADRF at each point a "treatment effect" (i.e., the rate at which the outcome changes corresponding to a small change in the predictor, or "treatment"), the AMEF is a function that relates the size of the treatment effect to the level of the treatment. The shape of the AMEF is usually of less importance than the value of the AMEF at each level of the predictor, which corresponds to the size of the treatment effect at the corresponding level. The AMEF is computed by computing the ADRF at each level of the focal predictor specified in `at`, shifting the predictor value by a tiny amount (control by `eps`), and computing the ratio of the change in the outcome to the shift, then averaging this value across all units. This quantity is related the the average marginal effect of a continuous predictor as computed by `sim_ame()`, but rather than average these treatment effects across all observed levels of the treatment, the AMEF is a function evaluated at each possible level of the treatment. The "tiny amount" used is `eps` times the standard deviation of `var`.

Note that inference on the computed quantities treats the other variables in the model as fixed; that is, it only accounts for model-based uncertainty, not uncertainty due to sampling.

## Value

A `<clarify_adrf>` object, which inherits from `<clarify_est>` and is similar to the output of `sim_apply()`, with the additional attributes "var" containing the variable named in `var`, "by" containing the names of the variables specified in `by` (if any), "at" containing values at which the ADRF or AMEF is evaluated, and "contrast" containing the argument supplied to `contrast`. For an ADRF, the average marginal means will be named  $E[Y(\{v\})]$ , where `{v}` is replaced with the values in `at`. For an AMEF, the average marginal effects will be named  $dY/d(\{x\})|_{\{a\}}$  where `{x}` is replaced with `var` and `{a}` is replaced by the values in `at`.

## See Also

- `plot.clarify_adrf()` for plotting the ADRF or AMEF.
- `sim_ame()` for computing average marginal effects.

- `sim_apply()`, which provides a general interface to computing any quantities for simulation-based inference.
- `summary.clarify_est()` for computing p-values and confidence intervals for the estimated quantities.
- `marginaleffects::avg_slopes()` and `marginaleffects::avg_predictions()` for delta method-based implementations of computing average marginal effects and average marginal means.
- the **adrftools** package, which performs inference, testing, and visualization of the ADRF and AMEF.

## Examples

```
data("lalonde", package = "MatchIt")

# Fit the model
fit <- glm(I(re78 > 0) ~ treat + age + race +
           married + re74,
           data = lalonde, family = binomial)

# Simulate coefficients
set.seed(123)
s <- sim(fit, n = 100)

# ADRF for `age`
est <- sim_adrf(s, var = "age",
               at = seq(15, 55, length.out = 6))
est
plot(est)

# AMEF for `age`
est <- sim_adrf(s, var = "age", contrast = "amef",
               at = seq(15, 55, length.out = 6))
est
summary(est)
plot(est)

# ADRF for `age` within levels of `married`
est <- sim_adrf(s, var = "age",
               at = seq(15, 55, length.out = 6),
               by = ~married)
est
plot(est)

## Difference between ADRFs
est_diff <- est[7:12] - est[1:6]
plot(est_diff) + ggplot2::labs(y = "Diff")
```

sim\_ame

*Compute average marginal effects***Description**

sim\_ame() is a wrapper for [sim\\_apply\(\)](#) that computes average marginal effects, the average effect of changing a single variable from one value to another (i.e., from one category to another for categorical variables or a tiny change for continuous variables).

**Usage**

```
sim_ame(
  sim,
  var,
  subset = NULL,
  by = NULL,
  contrast = NULL,
  outcome = NULL,
  type = NULL,
  eps = 1e-05,
  verbose = interactive(),
  cl = NULL,
  ...
)

## S3 method for class 'clarify_ame'
print(x, digits = 4L, max.ests = 6L, ...)
```

**Arguments**

sim	a <clarify_sim> object; the output of a call to <a href="#">sim()</a> or <a href="#">misim()</a> .
var	either the names of the variables for which marginal effects are to be computed or a named list containing the values the variables should take. See Details.
subset	optional; a vector used to subset the data used to compute the marginal effects. This will be evaluated within the original dataset used to fit the model using <a href="#">subset()</a> , so nonstandard evaluation is allowed.
by	a one-sided formula or character vector containing the names of variables for which to stratify the estimates. Each quantity will be computed within each level of the complete cross of the variables specified in by.
contrast	a string containing the name of a contrast between the average marginal means when the variable named in var is categorical and takes on two values. Allowed options include "diff" for the difference in means (also "rd"), "rr" for the risk ratio (also "irr"), "log(rr)": for the log risk ratio (also "log(irr)"), "sr" for the survival ratio, "log(sr)": for the log survival ratio, "srr" for the switch relative risk (also "grrr"), "or" for the odds ratio, "log(or)" for the log odds ratio, and "nnt" for the number needed to treat. These options are not case sensitive, but the parentheses must be included if present.

outcome	a string containing the name of the outcome or outcome level for multivariate (multiple outcomes) or multi-category outcomes. Ignored for univariate (single outcome) and binary outcomes.
type	a string containing the type of predicted values (e.g., the link or the response). Passed to <code>marginaleffects::get_predict()</code> and eventually to <code>predict()</code> in most cases. The default and allowable option depend on the type of model supplied, but almost always corresponds to the response scale (e.g., predicted probabilities for binomial models).
eps	when the variable named in <code>var</code> is continuous, the value by which to change the variable values to approximate the derivative. See Details.
verbose	logical; whether to display a text progress bar indicating progress and estimated time remaining for the procedure. Default is TRUE for interactive sessions and FALSE otherwise.
cl	a cluster object created by <code>parallel::makeCluster()</code> , or an integer to indicate the number of child-processes (integer values are ignored on Windows) for parallel evaluations. See <code>pbapply::pblapply()</code> for details. If NULL, no parallelization will take place.
...	for <code>sim_ame()</code> , additional arguments passed to <code>marginaleffects::get_predict()</code> (and eventually to <code>predict()</code> ) to compute predictions. For <code>print()</code> , ignored.
x	a <code>&lt;clarify_ame&gt;</code> object.
digits	the minimum number of significant digits to be used; passed to <code>print.data.frame()</code> .
max.ests	the maximum number of estimates to display.

## Details

`sim_ame()` computes average adjusted predictions or average marginal effects depending on which variables are named in `var` and how they are specified. Canonically, `var` should be specified as a named list with the value(s) each variable should be set to. For example, specifying `var = list(x1 = 0:1)` computes average adjusted predictions setting `x1` to 0 and 1. Specifying a variable's values as NULL, e.g., `list(x1 = NULL)`, is equivalent to requesting average adjusted predictions at each unique value of the variable when that variable is binary or a factor or character and requests the average marginal effect of that variable otherwise. Specifying an unnamed entry in the list with a string containing the value of that variable, e.g., `list("x1")` is equivalent to specifying `list(x1 = NULL)`. Similarly, supplying a vector with the names of the variables is equivalent to specifying a list, e.g., `var = "x1"` is equivalent to `var = list(x1 = NULL)`.

Multiple variables can be supplied to `var` at the same time to set the corresponding variables to those values. If all values are specified directly or the variables are categorical, e.g., `list(x1 = 0:1, x2 = c(5, 10))`, this computes average adjusted predictions at each combination of the supplied variables. If any one variable's values is specified as NULL and the variable is continuous, the average marginal effect of that variable will be computed with the other variables set to their corresponding combinations. For example, if `x2` is a continuous variable, specifying `var = list(x1 = 0:1, x2 = NULL)` requests the average marginal effect of `x2` computed first setting `x1` to 0 and then setting `x1` to 1. The average marginal effect can only be computed for one variable at a time.

Below are some examples of specifications and what they request, assuming `x1` is a binary variable taking on values of 0 and 1 and `x2` is a continuous variable:

- `list(x1 = 0:1), list(x1 = NULL), list("x1"), "x1"` – the average adjusted predictions setting `x1` to 0 and to 1
- `list(x2 = NULL), list("x2"), "x2"` – the average marginal effect of `x2`
- `list(x2 = c(5, 10))` – the average adjusted predictions setting `x2` to 5 and to 10
- `list(x1 = 0:1, x2 = c(5, 10)), list("x1", x2 = c(5, 10))` – the average adjusted predictions setting `x1` and `x2` in a full cross of 0, 1 and 5, 10, respectively (e.g., (0, 5), (0, 10), (1, 5), and (1, 10))
- `list(x1 = 0:1, "x2"), list("x1", "x2"), c("x1", "x2")` – the average marginal effects of `x2` setting `x1` to 0 and to 1

The average adjusted prediction is the average predicted outcome value after setting all units' value of a variable to a specified level. (This quantity has several names, including the average potential outcome, average marginal mean, and standardized mean). When exactly two average adjusted predictions are requested, a contrast between them can be requested by supplying an argument to `contrast` (see Effect Measures section below). Contrasts can be manually computed using `transform()` afterward as well; this is required when multiple average adjusted predictions are requested (i.e., because a single variable was supplied to `var` with more than two levels or a combination of multiple variables was supplied).

A marginal effect is the instantaneous rate of change corresponding to changing a unit's observed value of a variable by a tiny amount and considering to what degree the predicted outcome changes. The ratio of the change in the predicted outcome to the change in the value of the variable is the marginal effect; these are averaged across the sample to arrive at an average marginal effect. The "tiny amount" used is `eps` times the standard deviation of the focal variable.

The difference between using `by` or `subset` vs. `var` is that `by` and `subset` subset the data when computing the requested quantity, whereas `var` sets the corresponding variable to given a value for all units. For example, using `by = ~v` computes the quantity of interest separately for each subset of the data defined by `v`, whereas setting `var = list(., "v")` computes the quantity of interest for all units setting their value of `v` to its unique values. The resulting quantities have different interpretations. Both `by` and `var` can be used simultaneously.

Note that inference on the computed quantities treats the other variables in the model as fixed; that is, it only accounts for model-based uncertainty, not uncertainty due to sampling.

### Effect measures:

The effect measures specified in `contrast` are defined below. Typically only "diff" is appropriate for continuous outcomes and "diff" or "irr" are appropriate for count outcomes; the rest are appropriate for binary outcomes. For a focal variable with two levels, 0 and 1, and an outcome `Y`, the average marginal means will be denoted in the below formulas as  $E[Y(0)]$  and  $E[Y(1)]$ , respectively.

contrast	Description	Formula
"diff"/"rd"	Mean/risk difference	$E[Y(1)] - E[Y(0)]$
"rr"/"irr"	Risk ratio/incidence rate ratio	$E[Y(1)] / E[Y(0)]$
"sr"	Survival ratio	$(1 - E[Y(1)]) / (1 - E[Y(0)])$
"srr"/"grrr"	Switch risk ratio	$1 - sr$ if $E[Y(1)] > E[Y(0)]$ $rr - 1$ if $E[Y(1)] < E[Y(0)]$ $0$ otherwise
"or"	Odds ratio	$O[Y(1)] / O[Y(0)]$

"nnt"	Number needed to treat	where $O[Y(\cdot)] = E[Y(\cdot)] / (1 - E[Y(\cdot)])$ 1 / rd
-------	------------------------	-----------------------------------------------------------------

The `log(.)` versions are defined by taking the `log()` (natural log) of the corresponding effect measure.

## Value

A `<clarify_ame>` object, which inherits from `<clarify_est>` and is similar to the output of `sim_apply()`, with the additional attributes "var" containing the variable values specified in var and "by" containing the names of the variables specified in by (if any). The average adjusted predictions will be named  $E[Y(\{v\})]$ , where {v} is replaced with the values the variables named in var take on. The average marginal effect for a continuous var will be named  $E[dY/d(\{x\})]$  where {x} is replaced with var. When by is specified, the average adjusted predictions will be named  $E[Y(\{v\})|\{b\}]$  and the average marginal effect  $E[dY/d(\{x\})|\{b\}]$  where {b} is a comma-separated list of values of the by variables at which the quantity is computed. See examples.

## See Also

- `sim_apply()`, which provides a general interface to computing any quantities for simulation-based inference.
- `plot.clarify_est()` for plotting the output of a call to `sim_ame()`.
- `summary.clarify_est()` for computing p-values and confidence intervals for the estimated quantities.
- `marginaleffects::avg_predictions()`, `marginaleffects::avg_comparisons()` and `marginaleffects::avg_s` for delta method-based implementations of computing average marginal effects.

## Examples

```
data("lalonde", package = "MatchIt")

# Fit the model
fit <- glm(I(re78 > 0) ~ treat + age + race +
          married + re74,
          data = lalonde, family = binomial)

# Simulate coefficients
set.seed(123)
s <- sim(fit, n = 100)

# Average marginal effect of `age`
est <- sim_ame(s, var = "age")
summary(est)

# Contrast between average adjusted predictions
# for `treat`
est <- sim_ame(s, var = "treat", contrast = "rr")
summary(est)
```

```

# Average adjusted predictions for `race`; need to follow up
# with contrasts for specific levels
est <- sim_ame(s, var = "race")

est <- transform(est,
  `RR(h,b)` = `E[Y(hispan)]` / `E[Y(black)]`)

summary(est)

# Average adjusted predictions for `treat` within levels of
# `married`, first using `subset` and then using `by`
est0 <- sim_ame(s, var = "treat", subset = married == 0,
  contrast = "rd")
names(est0) <- paste0(names(est0), "|married=0")
est1 <- sim_ame(s, var = "treat", subset = married == 1,
  contrast = "rd")
names(est1) <- paste0(names(est1), "|married=1")

summary(cbind(est0, est1))

est <- sim_ame(s, var = "treat", by = ~married,
  contrast = "rd")

est
summary(est)

# Average marginal effect of `age` within levels of
# married*race
est <- sim_ame(s, var = "age", by = ~married + race)
est
summary(est, null = 0)

# Comparing AMEs between married and unmarried for
# each level of `race`
est_diff <- est[4:6] - est[1:3]
names(est_diff) <- paste0("AME_diff|", levels(lalonde$race))
summary(est_diff)

# Average adjusted predictions at a combination of `treat`
# and `married`
est <- sim_ame(s, var = c("treat", "married"))
est

# Average marginal effect of `age` setting `married` to 1
est <- sim_ame(s, var = list("age", married = 1))

```

**Description**

sim\_apply() applies a function that produces quantities of interest to each set of simulated coefficients produced by sim(); these calculated quantities form the posterior sampling distribution for the quantities of interest. Capabilities are available for parallelization.

**Usage**

```
sim_apply(sim, FUN, verbose = interactive(), cl = NULL, ...)
```

```
## S3 method for class 'clarify_est'
print(x, digits = 4L, max.ests = 6L, ...)
```

**Arguments**

sim	a <clarify_sim> object; the output of a call to sim() or misim().
FUN	a function to be applied to each set of simulated coefficients. See Details.
verbose	logical; whether to display a text progress bar indicating progress and estimated time remaining for the procedure. Default is TRUE for interactive sessions and FALSE otherwise.
cl	a cluster object created by parallel::makeCluster(), or an integer to indicate the number of child-processes (integer values are ignored on Windows) for parallel evaluations. See pbapply::pblapply() for details. If NULL, no parallelization will take place.
...	for sim_apply(), optional arguments passed to FUN. For print(), ignored.
x	a <clarify_est> object.
digits	the minimum number of significant digits to be used; passed to print.data.frame().
max.ests	the maximum number of estimates to display.

**Details**

sim\_apply() applies a function, FUN, to each set of simulated coefficients, similar to apply(). This function should return a numeric vector containing one or more estimated quantities. This should be a named vector to more easily keep track of the meaning of each estimated quantity. Care should be taken to ensure that the returned vector is the same length each time FUN is called. NAs are allowed in the output but should be avoided if possible.

The arguments to FUN can be specified in a few ways. If FUN has an argument called `coefs`, a simulated set of coefficients will be passed to this argument, and FUN should compute and return a quantity based on the coefficients (e.g., the difference between two coefficients if one wants to test whether two coefficients are equal). If FUN has an argument called `fit`, a model fit object of the same type as the one originally supplied to sim() (e.g., an <lm> or <glm> object) will be passed to this argument, where the coefficients of the fit object have been replaced by the simulated coefficients generated by sim(), and FUN should compute and return a quantity based on the model fit (e.g., a computation based on the output of predict()). If neither `coefs` nor `fit` are the names of arguments to FUN, the model fit object with replaced coefficients will be supplied to the first argument of FUN.

When custom coefficients are supplied to `sim()`, i.e., when the `coefs` argument to `sim()` is not left at its default value, `FUN` must accept a `coefs` argument and a warning will be thrown if it accepts a `fit` argument. This is because `sim_apply()` does not know how to reconstruct the original fit object with the new coefficients inserted. The quantities computed by `sim_apply()` must therefore be computed directly from the coefficients.

If `FUN` is not supplied at all, the simulated values of the coefficients will be returned in the output with a warning. Set `FUN` to `NULL` or `verbose` to `FALSE` to suppress this warning.

#### `sim_apply()` with multiply imputed data:

When using `misim()` and `sim_apply()` with multiply imputed data, the coefficients are supplied to the model fit corresponding to the imputation identifier associated with each set of coefficients, which means if `FUN` uses a dataset extracted from a model (e.g., using `insight::get_data()`), it will do so from the model fit in the corresponding imputation.

The original estimates (see Value below) are computed as the mean of the estimates across the imputations using the original coefficients averaged across imputations. That is, first, the coefficients estimated in the models in the imputed datasets are combined to form a single set of pooled coefficients; then, for each imputation, the quantities of interest are computed using the pooled coefficients; finally, the mean of the resulting estimates across the imputations are taken as the "original" estimates. Note this procedure is only valid for quantities with symmetric sampling distributions, which excludes quantities like risk ratios and odds ratios, but includes log risk ratios and log odds ratios. The desired quantities can be transformed from their log versions using `transform()`.

### Value

A `<clarify_est>` object, which is a matrix with a column for each estimated quantity and a row for each simulation. The original estimates (`FUN` applied to the original coefficients or model fit object) are stored in the attribute `"original"`. The `"sim_hash"` attribute contains the simulation hash produced by `sim()`.

### See Also

- `sim()` for generating the simulated coefficients.
- `summary.clarify_est()` for computing p-values and confidence intervals for the estimated quantities.
- `plot.clarify_est()` for plotting estimated quantities and their simulated posterior sampling distribution.

### Examples

```
data("lalonde", package = "MatchIt")
fit <- lm(re78 ~ treat + age + race + nodegree + re74,
         data = lalonde)
coef(fit)

set.seed(123)
s <- sim(fit, n = 500)

# Function to compare predicted values for two units
```

```

# using `fit` argument
sim_fun <- function(fit) {
  pred1 <- unname(predict(fit, newdata = lalonde[1,]))
  pred2 <- unname(predict(fit, newdata = lalonde[2,]))
  c(pred1 = pred1, pred2 = pred2)
}

est <- sim_apply(s, sim_fun)

# Add difference between predicted values as
# additional quantity
est <- transform(est, `diff 1-2` = pred1 - pred2)

# Examine estimates and confidence intervals
summary(est)

# Function to compare coefficients using `coefs`
# argument
sim_fun <- function(coefs) {
  setNames(coefs["racewhite"] - coefs["racehispan"],
           "wh - his")
}

est <- sim_apply(s, sim_fun)

# Examine estimates and confidence intervals
summary(est)

# Another way to do the above:
est <- sim_apply(s, FUN = NULL)
est <- transform(est,
                 `wh - his` = `racewhite` - `racehispan`)

summary(est, parm = "wh - his")

```

---

sim\_setx

---

*Compute predictions and first differences at set values*


---

## Description

`sim_setx()` is a wrapper for `sim_apply()` that computes predicted values of the outcome at specified values of the predictors, sometimes called marginal predictions. One can also compute the difference between two marginal predictions (the "first difference"). Although any function that accepted `clarify_est` objects can be used with `sim_setx()` output objects, a special plotting function, `plot.clarify_setx()`, can be used to plot marginal predictions.

## Usage

```
sim_setx(
```

```

sim,
x = list(),
x1 = list(),
outcome = NULL,
type = NULL,
verbose = interactive(),
cl = NULL,
...
)

## S3 method for class 'clarify_setx'
print(x, digits = 4L, max.ests = 6L, ...)

```

## Arguments

sim	a <clarify_sim> object; the output of a call to <code>sim()</code> or <code>misim()</code> .
x	a data frame containing a reference grid of predictor values or a named list of values each predictor should take defining such a reference grid, e.g., <code>list(v1 = 1:4, v2 = c("A", "B"))</code> . Any omitted predictors are fixed at a "typical" value. See Details. When x1 is specified, x should identify a single reference unit. For <code>print()</code> , a <clarify_setx> object.
x1	a data.frame or named list of the value each predictor should take to compute the first difference from the predictor combination specified in x. x1 can only identify a single unit. See Details.
outcome	a string containing the name of the outcome or outcome level for multivariate (multiple outcomes) or multi-category outcomes. Ignored for univariate (single outcome) and binary outcomes.
type	a string containing the type of predicted values (e.g., the link or the response). Passed to <code>marginaleffects::get_predict()</code> and eventually to <code>predict()</code> in most cases. The default and allowable option depend on the type of model supplied, but almost always corresponds to the response scale (e.g., predicted probabilities for binomial models).
verbose	logical; whether to display a text progress bar indicating progress and estimated time remaining for the procedure. Default is TRUE for interactive sessions and FALSE otherwise.
cl	a cluster object created by <code>parallel::makeCluster()</code> , or an integer to indicate the number of child-processes (integer values are ignored on Windows) for parallel evaluations. See <code>pbapply::pblapply()</code> for details. If NULL, no parallelization will take place.
...	for <code>sim_setx()</code> , additional arguments passed to <code>marginaleffects::get_predict()</code> (and eventually to <code>predict()</code> ) to compute predictions. For <code>print()</code> , ignored.
digits	the minimum number of significant digits to be used; passed to <code>print.data.frame()</code> .
max.ests	the maximum number of estimates to display.

## Details

When `x` is a named list of predictor values, they will be crossed to form a reference grid for the marginal predictions. Any predictors not set in `x` are assigned their "typical" value, which, for factor, character, logical, and binary variables is the mode, for numeric variables is the mean, and for ordered variables is the median. These values can be seen in the `"setx"` attribute of the output object. If `x` is empty, a prediction will be made at a point corresponding to the typical value of every predictor. Estimates are identified (in `summary()`, etc.) only by the variables that differ across predictions.

When `x1` is supplied, the first difference is computed, which here is considered as the difference between two marginal predictions. One marginal prediction must be specified in `x` and another, ideally with a single predictor changed, specified in `x1`.

## Value

A `<clarify_setx>` object, which inherits from `<clarify_est>` and is similar to the output of `sim_apply()`, with the following additional attributes:

- `"setx"` - a data frame containing the values at which predictions are to be made
- `"fd"` - whether or not the first difference is to be computed; set to `TRUE` if `x1` is specified and `FALSE` otherwise

## See Also

- `sim_apply()`, which provides a general interface to computing any quantities for simulation-based inference.
- `plot.clarify_setx()` for plotting the output of a call to `sim_setx()`.
- `summary.clarify_est()` for computing p-values and confidence intervals for the estimated quantities.
- `marginaleffects::predictions()` and `marginaleffects::comparisons()` for delta method-based implementations of computing predicted values and first differences.

## Examples

```
data("lalonde", package = "MatchIt")

fit <- lm(re78 ~ treat + age + educ + married + race + re74,
         data = lalonde)

# Simulate coefficients
set.seed(123)
s <- sim(fit, n = 100)

# Predicted values at specified values of values, typical
# values for other predictors
est <- sim_setx(s, x = list(treat = 0:1,
                           re74 = c(0, 10000)))

summary(est)
plot(est)
```

```

# Predicted values at specified grid of values, typical
# values for other predictors
est <- sim_setx(s, x = list(age = c(20, 25, 30, 35),
                           married = 0:1))

summary(est)
plot(est)

# First differences of treat at specified value of
# race, typical values for other predictors
est <- sim_setx(s, x = data.frame(treat = 0, race = "hispan"),
               x1 = data.frame(treat = 1, race = "hispan"))

summary(est)
plot(est)

```

---

transform.clarify\_est *Transform and combine <clarify\_est> objects*

---

## Description

transform() modifies a <clarify\_est> object by allowing for the calculation of new quantities from the existing quantities without re-simulating them. cbind() binds two <clarify\_est> objects together.

## Usage

```
## S3 method for class 'clarify_est'
transform(`_data`, ...)
```

```
## S3 method for class 'clarify_est'
cbind(..., deparse.level = 1)
```

## Arguments

`_data` the <clarify\_est> object to be transformed.

`...` for transform(), arguments in the form name = value, where name is the name of a new quantity to be computed and value is an expression that is a function of the existing quantities corresponding to the new quantity to be computed. See Details. For cbind(), <clarify\_est> objects to be combined.

`deparse.level` ignored.

## Details

For transform(), the expression on the right side of the = should use the names of the existing quantities (e.g., ``E[Y(1)]` - `E[Y(1)]``), with ``` appropriately included when the quantity name include parentheses or brackets. Alternatively, it can use indexes prefixed by `.b`, e.g., `.b2 - .b1`, to refer to the corresponding quantity by position. This can aid in computing derived quantities of quantities with complicated names. (Note that if a quantity is named something like `.b1`, it will

need to be referred to by position rather than name, as the position-based label takes precedence). See examples. Setting an existing value to NULL will remove that quantity from the object.

`cbind()` does not rename the quantities or check for uniqueness of the names, so it is important to rename them yourself prior to combining the objects.

## Value

A `<clarify_est>` object, either with new columns added (when using `transform()`) or combining two `<clarify_est>` objects. Note that any type attributes corresponding to the `sim_apply()` wrapper used (e.g., `sim_ame()`) is lost when using either function. This can affect any helper functions (e.g., `plot()`) designed to work with the output of specific wrappers.

## See Also

[transform\(\)](#), [cbind\(\)](#), [sim\(\)](#)

## Examples

```
data("lalonde", package = "MatchIt")

# Fit the model
fit <- lm(re78 ~ treat * (age + educ + race +
  married + re74 + re75),
  data = lalonde)

# Simulate coefficients
set.seed(123)
s <- sim(fit, n = 100)

# Average adjusted predictions for `treat` within
# subsets of `race`
est_b <- sim_ame(s, var = "treat",
  subset = race == "black")
est_b

est_h <- sim_ame(s, var = "treat",
  subset = race == "hispan")
est_h

# Compute differences between adjusted predictions
est_b <- transform(est_b,
  diff = `E[Y(1)]` - `E[Y(0)]`)
est_b

est_h <- transform(est_h,
  diff = `E[Y(1)]` - `E[Y(0)]`)
est_h

# Bind estimates together after renaming
names(est_b) <- paste0(names(est_b), "_b")
names(est_h) <- paste0(names(est_h), "_h")
```

```
est <- cbind(est_b, est_h)
est

# Compute difference in race-specific differences
est <- transform(est,
  `diff-diff` = .b6 - .b3)

summary(est,
  parm = c("diff_b", "diff_h", "diff-diff"))

# Remove last quantity by using `NULL`
transform(est, `diff-diff` = NULL)
```

# Index

`apply()`, 21  
`cbind()`, 27  
`cbind.clarify_est`  
    (transform.clarify\_est), 26  
`confint.clarify_est` (plot.clarify\_est), 5  
  
`ggplot2::geom_density()`, 5, 7, 10  
`ggplot2::geom_line()`, 5, 10  
`ggplot2::geom_ribbon()`, 5, 10  
`glm()`, 11  
  
`insight::get_data()`, 22  
`insight::get_parameters()`, 3, 11  
`insight::get_varcov()`, 3, 11  
  
`lm()`, 11  
`log()`, 19  
  
`marginaleffects::avg_comparisons()`, 19  
`marginaleffects::avg_predictions()`, 15, 19  
`marginaleffects::avg_slopes()`, 15, 19  
`marginaleffects::comparisons()`, 25  
`marginaleffects::get_predict()`, 13, 14, 17, 24  
`marginaleffects::predictions()`, 25  
`misim`, 2  
`misim()`, 11, 13, 16, 21, 22, 24  
  
`parallel::makeCluster()`, 13, 17, 21, 24  
`pbapply::pblapply()`, 13, 17, 21, 24  
`plot.clarify_adrf`, 4  
`plot.clarify_adrf()`, 14  
`plot.clarify_est`, 5  
`plot.clarify_est()`, 9, 10, 19, 22  
`plot.clarify_setx`, 9  
`plot.clarify_setx()`, 23, 25  
`print.clarify_adrf` (sim\_adrf), 12  
`print.clarify_ame` (sim\_ame), 16  
  
`print.clarify_est` (sim\_apply), 20  
`print.clarify_setx` (sim\_setx), 23  
`print.data.frame()`, 14, 17, 21, 24  
  
`rlang::hash()`, 3, 11  
  
`sim`, 10  
`sim()`, 3, 4, 7, 13, 16, 21, 22, 24, 27  
`sim_adrf`, 12  
`sim_adrf()`, 4, 5, 12  
`sim_ame`, 16  
`sim_ame()`, 4, 11, 14  
`sim_apply`, 20  
`sim_apply()`, 2, 4, 7, 8, 10–12, 15, 16, 19, 23, 25  
`sim_setx`, 23  
`sim_setx()`, 4, 9, 12  
`stats::coef()`, 3, 11  
`stats::vcov()`, 3, 11  
`subset()`, 13, 16  
`summary.clarify_est` (plot.clarify\_est), 5  
`summary.clarify_est()`, 5, 9, 10, 15, 19, 22, 25  
  
`transform()`, 18, 22, 27  
`transform.clarify_est`, 26