Package: propensity (via r-universe)

November 3, 2026

Title A Toolkit for Calculating and Working with Propensity Scores

Version 0.1.0.9000

Description Calculates propensity score weights for multiple causal 'estimands' across binary, continuous, and categorical exposures. Provides methods for handling extreme propensity scores through trimming, truncation, and calibration. Includes inverse probability weighted estimators that correctly account for propensity score estimation uncertainty.

License MIT + file LICENSE

URL https://r-causal.github.io/propensity/, https://github.com/r-causal/propensity

BugReports https://github.com/r-causal/propensity/issues

Depends R (>= 4.3)

Imports causalgenerics (>= 0.1.0.9000), cli, deli, generics, lifecycle, rlang, stats, tibble, tidyselect, vctrs (>= 0.6.5)

Suggests bench, broom, dplyr, ggplot2, knitr, MASS, mgcv, mice (>= 3.18.0), nnet, parsnip, probably, PSweight, rmarkdown, spelling, testthat (>= 3.1.9), tidyr, WeightIt, withr

VignetteBuilder knitr

Remotes r-causal/causalgenerics, r-causal/deli

Config/roxygen2/version 8.1.0

Config/testthat/edition 3

Encoding UTF-8

Language en-US

Roxygen list(markdown = TRUE)

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

Date/Publication 2026-10-04 17:09:50 UTC

RemoteUrl https://github.com/r-causal/propensity

RemoteRef HEAD

RemoteSha e9d163a2ac339d5b2c62a650fbbf3bfc5ac5b3c8

Contents

propensity-package . . . 2
augment.ipw . . . 4
dens_normal . . . 6
exposure_type . . . 8
glance.ipw . . . 10
glance.ipw_pooled . . . 12
ipw-methods . . . 13
is_ps_calibrated . . . 41
is_ps_trimmed . . . 42
is_ps_truncated . . . 43
is_refit . . . 44
is_unit_trimmed . . . 45
is_unit_truncated . . . 46
is_wt_truncated . . . 47
joint_wt_models . . . 48
numerator_model . . . 49
print.ipw_diagnostic_se . . . 51
ps_calibrate . . . 52
ps_refit . . . 55
ps_tilt . . . 58
ps_trim . . . 61
ps_trim_meta . . . 69
ps_trunc . . . 70
ps_trunc_meta . . . 76
psw . . . 77
tidy.ipw . . . 83
tidy.ipw_pooled . . . 86
wt_ate . . . 89
wt_joint . . . 108
wt_trunc . . . 111
wt_trunc_sensitivity . . . 115

Index **118**

`propensity-package` *propensity: A Toolkit for Calculating and Working with Propensity Scores*

Description

propensity provides tools for propensity score analysis in causal inference. Calculate propensity score weights for a variety of causal estimands, handle extreme propensity scores through trimming, truncation, and calibration, and estimate causal effects with inverse probability weighting. The package supports binary, categorical, and continuous exposures.

Weight functions

Calculate propensity score weights for different causal estimands:

- `wt_ate()`: Average treatment effect (ATE) weights
- `wt_att()`: Average treatment effect on the treated (ATT) weights
- `wt_atu()`: Average treatment effect on the untreated (ATU) weights (`wt_atc()` is an alias)
- `wt_atm()`: Average treatment effect for the evenly matchable (ATM) weights
- `wt_ato()`: Average treatment effect for the overlap population (ATO) weights
- `wt_entropy()`: Entropy weights for the entropy-tilted population
- `wt_cens()`: Censoring weights

Propensity score modifications

Handle extreme propensity scores before calculating weights:

- `ps_trim()`: Trim observations with extreme propensity scores
- `ps_trunc()`: Truncate (winsorize) extreme propensity scores
- `ps_calibrate()`: Calibrate propensity scores to improve balance
- `ps_refit()`: Re-estimate the propensity score model after trimming

Estimation

- `ipw()`: Inverse probability weighted estimator with variance estimation that accounts for propensity score estimation uncertainty

PSW class

The `psw()` class represents propensity score weights with metadata about the estimand and modifications applied:

- `psw()`, `as_psw()`, `is_psw()`: Create and test propensity score weights
- `estimand()`: Query the causal estimand
- `is_stabilized()`: Check if weights are stabilized

Author(s)

Maintainer: Malcolm Barrett <malcolmbarrett@gmail.com> (ORCID) [copyright holder]

Authors:

- Malcolm Barrett <malcolmbarrett@gmail.com> (ORCID) [copyright holder]

See Also

- `vignette("propensity")` for a getting started guide
- The package website for full documentation

`augment.ipw` *Augment an inverse probability weighted result with per-observation columns*

Description

`augment()` works per observation rather than per estimate: the data an `ipw()` result was produced from, carried through in full, with the propensity score, the weights, the fitted values, and the residuals attached as dot-prefixed columns. Nothing is dropped and no column of the source frame is changed.

The added columns are read from the two models the result holds, so they describe the fit rather than the frame they are attached to. `data` says which frame to carry them on, which is how broom uses the argument: it names the data the fit was produced from, not new data to predict on. Supplying the modeling data rather than leaving the default is how covariates the outcome formula left out arrive beside the fit's own columns.

The weights are reported once, as `.weights`, which is a deliberate departure from broom. A broom `augment()` method carries the model frame's `(weights)` column through and adds no weights column of its own; here the weights are the central per-observation quantity of the analysis, and the `psw()` vector reports them with the estimand they target attached, which the model frame's plain numeric copy of the same values does not. The default frame therefore leaves `(weights)` out. A frame supplied through `data` is the caller's own and is carried as it arrives, so a weight column they hold stays where they put it, whatever it is named.

A column is an answer per observation only while the fit it is read from and the frame it is carried on describe the same observations, so three things are refused rather than reported. A result whose propensity score model and outcome model were fit to different rows is refused, because the propensity score of one observation would arrive beside the fitted value of another. Two models of the same data most often part this way over missing values, each dropping the rows a variable of its own is missing on. What is compared is the number of observations each model produced an answer for and, when the outcome model's frame names the exposure, the exposure values of the two model frames position by position, reading a factor and the numbers its labels spell as one encoding of one exposure. Exposure values that disagree prove the two models hold different observations; exposure values that agree prove nothing, since two different sets of rows are free to carry the same sequence of values, and two encodings that cannot be read onto each other, such as a factor of `"a"` and `"b"` against a recoding of it as 0 and 1, prove nothing either way. The check is one-sided in that direction on purpose: it refuses only what it can prove, so a fit whose models do line up is never refused over the labels the rows of either frame carry or the encoding its exposure is written in. A frame that already holds a column this call would add is refused as well, because the fit's column has nowhere to go: added beside the frame's, the two would be told apart by nothing, and written over it, the frame's data would be gone. Which names are in the way depends on the fit and the frame, `.resid` being among them only for a frame it would be added to and the propensity columns being the ones the propensity score model produced. Last, a result whose outcome model was fit without weights is refused under the class `ipw()` refuses the same fact with: `.weights` has nothing to report for such a fit.

Usage

```
## S3 method for class 'ipw'
```

```
augment(x, data = NULL, ...)
```

Arguments

x	An ipw object, as returned by ipw().
data	A data frame with one row for each observation the fit used, or NULL, the default, to use the outcome model's own model frame. That frame holds the response and the terms of the outcome formula, and both are kept. The (weights) column stats::model.frame() records is the one thing left out of it, because those weights are reported as .weights.
...	These dots are for future extensions and must be empty.

Value

A tibble with one row per observation, holding every column of the source frame in its own order followed by:

.propensity The propensity score, as the propensity score model predicts it: the probability of exposure for a binary exposure, and the conditional mean of the exposure for a continuous one. This is the propensity score the weight functions take, so feeding it back with the exposure returns the weights the outcome model was fit with.

.propensity_<level> For a categorical exposure, which has a probability for every level rather than a single number, one column per level in place of .propensity, named for the level and ordered as the propensity score model orders its levels.

.weights The weights the outcome model was fit with, as the psw() vector they were supplied as, so the estimand they record travels with them.

.fitted The outcome model's fitted values, on the response scale.

.resid The observed outcome less .fitted. Present only when the source frame holds the outcome the model was fit on, which the default frame always does. A factor or logical outcome is differenced on the 0/1 scale its fitted values are on.

See Also

ipw() for the estimator, tidy() for its estimates, and glance() for a one-row summary of the fit.

Examples

```
set.seed(123)
n <- 200
x1 <- rnorm(n)
z <- rbinom(n, 1, plogis(0.5 * x1))
y <- rbinom(n, 1, plogis(-0.5 + 0.8 * z + 0.3 * x1))
dat <- data.frame(x1, z, y)

ps_mod <- glm(z ~ x1, data = dat, family = binomial())
wts <- wt_ate(ps_mod)
outcome_mod <- glm(y ~ z, data = dat, family = quasibinomial(), weights = wts)
result <- ipw(ps_mod, outcome_mod)
```

```
augment(result)

# The modeling data carries through the covariates the outcome formula left
# out
augment(result, data = dat)
```

`dens_normal` *Density specifications for continuous exposures*

Description

Weights for a continuous exposure are a ratio of densities. Both densities are evaluated on the standardized residual $z_i = (A_i - \mu_i)/\sigma$, where A_i is the exposure, μ_i the fitted conditional mean, and σ the residual spread, so a specification describes a density on a standardized scale rather than on the scale of the exposure itself.

These constructors name the family and record the parameters that identify it. Pass one to the `.density` argument of `wt_ate()` or `wt_cens()`, which also accept the strings `"normal"`, `"laplace"`, and `"kernel"` for the families that need no parameters, and a bare function of one argument, which is wrapped with `dens_fn()`.

- `dens_normal()` is the standard normal density, the default.
- `dens_laplace()` is the standard Laplace density, $\exp(-|z|)/2$, which puts more mass in the tails than the normal. Its scale is estimated under the Laplace itself, where it is the mean absolute residual, or by the root mean square of the residuals, which is the spread every family without an estimator of its own is read at.
- `dens_t()` is Student's t density with `df` degrees of freedom, heavier tailed still, and heavier the smaller `df` is. Its scale is estimated under the t itself, by maximum likelihood, or by the root mean square of the residuals.
- `dens_kernel()` is a kernel density estimate of the standardized residuals, fit with `stats::density()` and interpolated to each observation. It assumes no family at all, at the cost of a density that is not a smooth function of the model's parameters: weights built from it have no closed-form standard error, and `ipw()` reports none for them at all. Bootstrap the whole fit by hand to put an interval around such an estimate.
- `dens_fn()` is a density you write yourself.

Usage

```
dens_normal()

dens_laplace(sigma_method = c("mle", "rms"))

dens_t(df, sigma_method = c("mle", "rms"))

dens_kernel(bw = "nrd0", adjust = 1, kernel = "gaussian", n = 512)

dens_fn(f)
```

Arguments

sigma_method	How the spread of the conditional density is estimated from the residuals of the propensity score model, either "rms", the root mean square, or "mle", the scale estimated under the family itself, which both dens_laplace() and dens_t() take by default. See the section below.
df	Degrees of freedom for Student's t, a single positive, finite number.
bw	The bandwidth passed to stats::density(): a single positive number, or the name of one of its selection rules ("nrd0", "nrd", "ucv", "bcv", "SJ", "SJ-ste", or "SJ-dpi").
adjust	A single positive number the bandwidth is multiplied by, as in stats::density(). Values above 1 smooth the estimate further.
kernel	The smoothing kernel, one of the kernels stats::density() accepts.
n	The number of grid points stats::density() evaluates, at least 2. The estimate is interpolated between them, so a larger n follows the shape of the residuals more closely.
f	A function of one argument, the standardized residual, returning one non-negative, finite density value for each element it is given.

Value

An object of class propensity_density: a list with the elements family, the name of the family; params, the parameters that identify it, which is an empty list for a family that takes none; and fn, the function that evaluates the density, which is NULL for dens_kernel(). dens_t() and dens_laplace() carry a fourth element, sigma_method, the name of the estimator their scale is read with.

The spread of the conditional density

Both densities are evaluated on a residual standardized by a spread, and for the conditional density that spread is estimated from the residuals of the propensity score model. A family with no scale estimator of its own takes the root mean square of those residuals, which is sigma_method = "rms" and is the maximum likelihood estimator of the spread of a normal density.

It is not the maximum likelihood estimator of the scale of a heavier-tailed family. The scale of such a family is smaller than its standard deviation, and the root mean square is pulled outward by the large residuals a heavy tail produces, which is what the family was chosen to accommodate. sigma_method = "mle" estimates the scale under the family itself, and both dens_laplace() and dens_t() take it by default, since the spread the density is divided by is the scale parameter of the family reading it. For dens_laplace() it is the mean absolute residual. For dens_t() it is the root of $\sum_i \left[(\nu+1)r_i^2/(\nu\sigma^2+r_i^2)\right] = n$, where r_i is the residual and ν is df. Each residual enters that sum through a bounded term, so a residual far out in the tail moves the estimate by less than it moves the root mean square. Prefer it when the residuals are heavy tailed, which is the case these families are for; the two estimators answer the same question, and answer it alike, as df grows and the t approaches the normal.

dens_normal() offers no such choice, because the root mean square is already its maximum likelihood estimator. Neither do dens_kernel() and dens_fn(): a kernel is fit to the standardized

residuals and divided by the same spread, so the spread cancels up to the grid the estimate is interpolated on, and a density you write yourself names no family an estimator could be derived from.

The choice describes both densities of the ratio. The marginal density that stabilizes the weights is the exposure's own, read at the exposure's mean and at the spread the same estimator gives for it, so the two halves of the ratio are densities of the same width. A scale estimated by maximum likelihood is recorded by `density_meta()` as `sigma = "mle"`, and `ipw()` estimates it alongside the propensity score model's coefficients, solving the equation it is the root of as part of its stacked system so that the standard errors account for it. Supplying a `.sigma` says the spread is a number of your own rather than one estimated from the residuals, so the two cannot be given together.

Examples

```
dens_normal()

dens_t(df = 4)

dens_t(df = 4, sigma_method = "mle")

dens_laplace(sigma_method = "rms")

dens_kernel(adjust = 1.5)

dens_fn(function(z) stats::dt(z, df = 4))
```

`exposure_type` *What a set of weights records about the exposure*

Description

Weights carry two records describing the exposure they were built for. `exposure_type()` returns the type of exposure the weight function was given, and `density_meta()` returns the record left by weights whose value is a ratio of densities.

Both describe the exposure rather than the units, so neither holds a length of its own: both survive subsetting, arithmetic, and anything else that changes the number of weights. Combining two sets of weights that record either of them differently drops the record they disagree on, with a warning of class `propensity_metadata_conflict_warning`. A result that no longer records an exposure type is not described as continuous, so it drops any density record along with it.

Usage

```
exposure_type(wt)

density_meta(wt)

## S3 method for class 'propensity_density_meta'
format(x, ...)
```

```
## S3 method for class 'propensity_density_meta'
print(x, ...)
```

Arguments

wt	A psw or causal_wts object.
x	A density record, as returned by density_meta().
...	Not used.

Details

exposure_type() returns "binary", "categorical", or "continuous". It returns NULL for weights that were built without an exposure to describe, such as those written with psw() directly.

density_meta() returns a record of what the ratio was built from, and NULL for weights that are not a ratio of densities, which is every set of weights for a binary or categorical exposure:

- density, the specification of the conditional density family, as built by dens_normal() and its relatives.
- numerator, what stabilized the weights: "marginal" for the marginal density of the exposure, "integrated" for the conditional density marginalized over the units, "model" for the conditional density a fitted model supplied to stabilize estimates, "score" for a stabilization_score the caller supplied, and "none" for weights that were not stabilized.
- numerator_model, that fitted model itself, and NULL under every other numerator. The model is kept whole rather than the numerator it evaluates to, since ipw() rebuilds that numerator at every value of its parameter vector, which takes the model's design and its coefficients. Two sets of weights stabilized on models that read different terms, or that were fit to different values of the same terms, record different numerators and are combined the way any other disagreeing records are.
- sigma, where the residual spread of the conditional density came from: "pooled" for the pooled residual root mean square, "mle" for a scale estimated under the family that reads it, which dens_t() and dens_laplace() take with sigma_method = "mle", and "supplied" for a .sigma the caller gave. Weights built from a dose model trimmed with ps_trim() record where the trim's spread came from.
- sigma_value, the single spread the weights were read at when it was not estimated by the weight function itself: one the caller supplied, or the spread a dose model trimmed with ps_trim() holds in its record, which is recorded whatever its source. It is NULL for a spread estimated from the residuals, by either estimator, and for one supplied per observation. A spread that is one number is a constant the weights can be rebuilt from, which is what ipw() needs of it; a spread that changes with the observation is not, so the record holds where it came from and nothing more.

Weights that carry a density record print it under their values, as the lines format() renders: the density family, the numerator, and the spread, followed by the formula of the model supplied to stabilize when a model estimated the numerator. Weights that carry none, which is every set of weights for a binary or categorical exposure, print exactly as they always have.

Value

- `exposure_type()`: a single string, or `NULL`.
- `density_meta()`: a list of class `propensity_density_meta` with the elements `density`, `numerator`, `sigma`, `sigma_value`, and `numerator_model`, or `NULL`.
- `print()` and `format()`: the record, invisibly, and a character vector of one line per choice the record describes.

See Also

`psw()` for the weight vector class and the rest of what it records, and `wt_ate()` for the functions that write these records.

Examples

```
set.seed(1)
ps <- runif(20, 0.2, 0.8)
trt <- rbinom(20, 1, ps)

# A binary exposure is weighted by propensity scores rather than by a
# density, so there is no density to record.
exposure_type(wt_ate(ps, trt))
density_meta(wt_ate(ps, trt))

dose <- rnorm(20)
mu <- 0.3 * ps
w <- wt_ate(mu, dose, exposure_type = "continuous")

exposure_type(w)
density_meta(w)
```

---

glance.ipw *Glance at an inverse probability weighted result*

Description

`glance()` describes an `ipw()` result rather than its estimates: one row naming the estimand and counting the observations and the residual degrees of freedom of the system the standard errors came from. A fit reporting several effect measures, or several contrasts of a categorical exposure, still returns exactly one row.

Under M-estimation that system is the stacked estimating equations, which hold the propensity score model, the outcome model, and the effect measures at once. Its residual degrees of freedom are the observations it was solved on less the parameters it solves for, so a multinomial propensity score model leaves fewer of them than a binary one does on the same data. Linearization and the robust diagnostic stack nothing and record no parameter count, so the observations are the outcome model's and there is no count to subtract from them.

The columns and their types are the same on every route `ipw()` takes, so the rows of several results stack into one table.

Usage

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

Arguments

x	An ipw object, as returned by ipw().
...	These dots are for future extensions and must be empty.

Value

A one-row tibble with the columns:

estimand The estimand the weights target, such as "ate".

nobs The number of observations the outcome model was fit on, which is also what the stacked estimating equations are solved on under M-estimation. Reported by stats::nobs().

df.residual The residual degrees of freedom of the stacked estimating equations, nobs less the number of parameters the system solves for. NA under linearization, which records no parameter count. Reported by stats::df.residual().

See Also

ipw() for the estimator, tidy() for its estimates, and augment() for its per-observation columns.

Examples

```
set.seed(123)
n <- 200
x1 <- rnorm(n)
z <- rbinom(n, 1, plogis(0.5 * x1))
y <- rbinom(n, 1, plogis(-0.5 + 0.8 * z + 0.3 * x1))
dat <- data.frame(x1, z, y)

ps_mod <- glm(z ~ x1, data = dat, family = binomial())
wts <- wt_ate(ps_mod)
outcome_mod <- glm(y ~ z, data = dat, family = quasibinomial(), weights = wts)
result <- ipw(ps_mod, outcome_mod)

glance(result)
```

`glance.ipw_pooled` *Glance at a pooled inverse probability weighted result*

Description

`glance()` describes a `pool_ipw()` result rather than its estimates: one row naming the estimand, counting the observations and the results that were pooled, and reporting the complete-data degrees of freedom the small-sample adjustment used. A pooled result reporting several effect measures, or several contrasts of a categorical exposure, still returns exactly one row.

Usage

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

Arguments

`x`	An `ipw_pooled` object, as returned by `pool_ipw()`.
`...`	These dots are for future extensions and must be empty.

Value

A one-row tibble with the columns:

`estimand` The causal estimand every pooled result targeted.

`nobs` The smallest number of observations any pooled result was estimated from. The imputed datasets are the same size, so this is that size unless a result was fitted on a subset of one. Reported by `stats::nobs()`.

`m` The number of results pooled.

`dfcom` The complete-data degrees of freedom the Barnard-Rubin adjustment used.

See Also

`pool_ipw()` for the pooling, `tidy()` for the pooled estimates, and the Multiple imputation section of `ipw()` for the workflow the two belong to.

Examples

```
set.seed(2024)
n <- 150
x1 <- rnorm(n)
z <- rbinom(n, 1, plogis(0.3 * x1))
y <- rbinom(n, 1, plogis(-0.4 + 0.9 * z + 0.5 * x1))
dat <- data.frame(x1, z, y)
dat$x1[rbinom(n, 1, 0.2) == 1] <- NA

imp <- mice::mice(dat, m = 2, print = FALSE, seed = 1)
```

```
fits <- with(imp, {
  ps <- glm(z ~ x1, family = binomial())
  w <- wt_ate(ps)
  om <- glm(y ~ z, family = quasibinomial(), weights = w)
  ipw(ps, om)
})

glance(pool_ipw(fits))
```

`ipw-methods` *Inverse Probability Weighted Estimation*

Description

The `joint_wt_models` method estimates the effects of a joint intervention on two treatments from the pair of fitted treatment models `joint_wt_models()` records and a weighted outcome model that reads both treatments. Standard errors are computed by M-estimation. Neither `se_method = "linearization"` nor `se_method = "robust"` is available here: neither path solves a stacked system, and the cell means and every contrast built from them are parameters of one.

The only supported estimand is `"ate"`, which is what the product weights `wt_joint()` builds target.

Either treatment may be categorical rather than binary, fit with `nnet::multinom()`, in which case its block is the multinomial score that fit solves and the surface reports the whole crossing its levels make. Such a component may be stabilized like any other, its numerator estimated in a block of its own; see **Joint exposures**.

The second treatment may be a dose, whatever the first one is, in which case the surface is the marginal structural model's own coefficients rather than the cells of a crossing, with one level contrast per non-reference level of the first treatment and one dose slope at its reference level. The dose model is read through the same registry the single-treatment route reads, so an `stats::lm()`, a gaussian `stats::glm()` at an identity or a log link, and a `MASS::rlm()` fit with one of the psi functions MASS supplies are stacked, and the dose component's weights may be any density ratio `wt_ate()` builds. A dose model or a density the stacked system cannot differentiate is refused rather than resampled, since this route has no resampling method; see **Joint exposures**.

Either component may be stabilized, including on a numerator model the caller fit and passed to `wt_ate()`'s `stabilize`. Each component's numerator is estimated in a block of its own, so the standard errors account for it having been fitted; see **Joint exposures** for what each block holds and for what a numerator may condition on. A numerator the caller computed and passed as `wt_ate()`'s `stabilization_score` is read back off the product and held fixed, contributing no block, as it is for a single treatment.

Every score this route stacks is unweighted, so a treatment model fit with prior case weights is refused by the name it was recorded under in `joint_wt_models()`.

This is the second of the two routes to a joint intervention. The other declares the crossing with `causalgenerics::joint_exposure()` and weights it with one multinomial model over the cells; see **Joint exposures**. Both report the same surface, so the choice between them is a modeling choice rather than a reporting one. Prefer this route when the two treatments call for different adjustment

sets, or when the dependence of the second treatment on the first is what you want to model directly: it weights through the sequential factorization `f(A | L) f(E | A, L)`, so each treatment gets its own model and its own covariates. Prefer the declared crossing when one model over the cells is the natural specification.

The factorization is validated when the container is built rather than here: `joint_wt_models()` refuses a second model that does not condition on the first treatment, and a pair that condition on each other. By the time a container reaches `ipw()` those questions are settled.

The `multinom` method estimates causal effects for a categorical exposure from a fitted `nnet::multinom()` propensity score model and a weighted outcome model. Standard errors are computed by M-estimation. Neither `se_method = "linearization"` nor `se_method = "robust"` is available for a categorical exposure, and both are refused with an error of class `propensity_method_error`: the stacked system has a sandwich for every fit this exposure type accepts, and the diagnostic sandwich describes two fitted cells that a categorical result does not report.

For a K-level exposure, the counterfactual mean at each level is reported first, under the effect label `"mean"`, and then the effects for each non-reference level against the reference (first) factor level. The `contrast` column names the level each mean row belongs to and the pair of levels each effect row compares, as it does for a binary exposure.

Two fits are refused with an error of class `propensity_model_family_error`. A `nnet::multinom()` fit to two levels reports the single probability of a binary exposure rather than one probability per level; fit such an exposure with `stats::glm()` and `family = binomial()` and pass it to the `glm` method instead. A fit to a matrix response records no levels at all, since `nnet::multinom()` reads a matrix as counts; refit it with the exposure factor on the left-hand side.

The `lm` method estimates the causal dose-response effect for a continuous exposure from a fitted `stats::lm()` (or gaussian-family `stats::glm()`) propensity score model of the exposure and a weighted marginal structural outcome model. A `MASS::rlm()` fit with one of the psi functions MASS supplies and an `mgcv::gam()` of a gaussian family reach this method by inheritance and are stacked at their own scores, the additive fit with its smoothing parameters held at the values it reports. A gaussian model is read through its link, which may be identity or log; the other gaussian links, and every class the stacked system has no score for, are refused, as are weights built with a `"kernel"` density, which have no closed-form standard error. See **Continuous propensity score models** in `ipw()` for what each refusal says, what to do about it, and what the additive route conditions on. The only supported estimand is `"ate"`.

Standard errors are computed by M-estimation; neither the linearization method nor the robust diagnostic is available for continuous exposures. The package offers no resampling method, so a fit the stacked system cannot write has no standard error here, and the refusal points at a bootstrap of the whole pipeline written by hand. See **Continuous propensity score models** in `ipw()`.

Every term of the marginal structural model that reads the exposure must read the exposure and nothing else, and the reported effects are the coefficients those terms contribute, labeled `"log(or)"` for a logit-link outcome and `"log(rr)"` for a log-link one. A model with one exposure coefficient keeps the eight-column contract with no contrast column and labels its row `"slope"` at an identity link; a dose-response curve such as `y ~ A + I(A^2)` reports one row per coefficient under `"coef"`, gaining a contrast column that names them.

A curve is also the one shape of fit that has a single reading. An exposure entering the outcome model through more than one design column, whether written as `y ~ A + I(A^2)` or as a basis such as `poly(A, 2)`, `splines::ns(A, 3)`, or `splines::bs(A, 3)`, has no coefficient that is the effect of the exposure, because the dose response has a different slope at every dose. `ipw()` records the

conditional reading for such a fit and reports it once as a message. Asking `ipw()` itself for `effects = "marginal"` is refused with an error of class `propensity_ipw_effects_error`; the result declares the conditional reading as the only one it supports, so `causalgenerics::as_marginal()`, `stats::coef()`, `stats::vcov()`, `stats::confint()`, `generics::tidy()`, and `as.data.frame()` refuse it from the result class instead, with an error of class `causalgenerics_unsupported_reading_marginal`. Marginalizing the curve over the observed doses is a separate estimand that this package does not compute; use the **marginaleffects** package on the conditional result: `avg_slopes()` for slopes, `avg_comparisons()` for contrasts, and `avg_predictions()` for causal dose-response functions. A basis term reading a covariate rather than the exposure is a covariate term however many columns it expands to, so it contributes no row and leaves the reading marginal.

A model frame records the term rather than the variables inside it, so the frame of a fit whose exposure enters through `poly()` or a spline holds no exposure column and `.data` must supply one. The propensity score design is then rebuilt through the model's terms object, whose `predvars` attribute records the basis the model was fit with, so the rebuilt columns are the fitted ones.

`ipw()` is a bring-your-own-model (BYOM) inverse probability weighted estimator for causal inference. You supply a fitted propensity score model and a fitted weighted outcome model; `ipw()` computes causal effect estimates with standard errors that correctly account for the two-step estimation process.

`ipw()` supports binary, categorical, and continuous exposures. For a binary or categorical exposure, a binary outcome returns the risk difference, log risk ratio, and log odds ratio, and a continuous outcome returns the difference in means. A continuous exposure is supplied through an `stats::lm()` or gaussian `stats::glm()` propensity score model with a weighted marginal structural outcome model whose link is identity, logit, or log, and `ipw()` reports the single exposure coefficient of that model. A `MASS::rlm()` fit with one of the psi functions MASS supplies and an `mgcv::gam()` fit of a gaussian family are read as the equations their own coefficients solve; every other subclass errors.

Usage

```
## S3 method for class 'joint_wt_models'
ipw(
  wt_mod,
  outcome_mod,
  ...,
  .data = NULL,
  .by = NULL,
  estimand = NULL,
  ps_link = NULL,
  conf_level = 0.95,
  se_method = c("mestimation", "linearization", "robust"),
  effects = c("marginal", "conditional")
)

## S3 method for class 'multinom'
ipw(
  wt_mod,
  outcome_mod,
  ...,
  .data = NULL,
```

```
  .by = NULL,
  estimand = NULL,
  ps_link = NULL,
  conf_level = 0.95,
  se_method = c("mestimation", "linearization", "robust"),
  .focal_level = NULL,
  effects = c("marginal", "conditional")
)

## S3 method for class 'lm'
ipw(
  wt_mod,
  outcome_mod,
  ...,
  .data = NULL,
  .by = NULL,
  estimand = NULL,
  ps_link = NULL,
  conf_level = 0.95,
  se_method = c("mestimation", "linearization", "robust"),
  effects = c("marginal", "conditional")
)

## S3 method for class 'glm'
ipw(
  wt_mod,
  outcome_mod,
  ...,
  .data = NULL,
  .by = NULL,
  estimand = NULL,
  ps_link = NULL,
  conf_level = 0.95,
  se_method = c("mestimation", "linearization", "robust"),
  effects = c("marginal", "conditional")
)
```

Arguments

wt_mod	The weighting object: a fitted propensity score model that produced the weights, typically a logistic regression of class stats::glm() with the exposure as the left-hand side of the formula. ipw() is an S3 generic that dispatches on this object. The left-hand side must be the exposure column itself, for every exposure type: a matrix response such as cbind(successes, failures) and a transformed response such as log(a) or factor(z) both error, because ipw() reads the exposure's name from that expression. Compute a transformed exposure into its own column and fit wt_mod on that column.
outcome_mod	A fitted weighted outcome model of class stats::glm() or stats::lm(), with

the outcome as the dependent variable and propensity score weights supplied via the `weights` argument. The weights should be created with a propensity weight function such as `wt_ate()`. Supported outcome models are an `stats::lm()`, a gaussian `stats::glm()` with an identity link, and a binomial or quasibinomial `stats::glm()` with a logit, probit, cloglog, log, or identity link; any other family (such as poisson or Gamma), an unsupported link (such as cauchit), or a non-identity gaussian link errors. A factor or logical outcome response is converted to `0`/`1` following glm's coding (the first factor level is the failure, every other level is the success).

`...` These dots are for future extensions and must be empty. They separate the two models from the remaining arguments, which must therefore be supplied by name. Anything left in them, such as a `.data` argument given by position or a misspelled argument name, errors.

`.data` A data frame containing the exposure, outcome, and covariates. If `NULL` (the default), `ipw()` extracts data from the model objects. Supply `.data` explicitly if the outcome model formula contains transformations that prevent extraction of the exposure variable from `stats::model.frame()`, or if the propensity score model cannot reconstruct its design (for example, fit with `model = FALSE` with the fitting data no longer available); `ipw()` then rebuilds the propensity design from `.data`. `.data` must be the data the models were fit to. It must have one row per observation the models were fit to, and each column must carry the type its model was fit on: a column fit as a factor may also be supplied as character or as an ordered factor, since every design rebuilt from `.data` is rebuilt under the levels the fit recorded, but a factor supplied as numeric, or a numeric supplied as a factor or a logical, errors. A row count or a column type that disagrees with the fitted models errors rather than rebuilding a design the models were not fit to.

The levels a categorical column declares are its fit's to decide, so a column releveled after fitting is rebuilt in the fitted order, and one that declares a level no observation carries rebuilds the fitted design, since the fits drop unused levels themselves. A value the fit never saw has no coefficient to multiply and errors. The exposure column and the outcome model's response are read rather than rebuilt, and both are rejected when the order they declare contradicts the fit: `ipw()` treats the second level of a binary exposure as the exposed group, and codes a factor response as an indicator for its non-first levels, so either one releveled the other way describes the opposite contrast.

Supplying `.data` also decides how a term that transforms the exposure is handled. The counterfactual designs are built by setting the exposure column to each level in turn and rebuilding the outcome design from `.data`, so a term such as `as.numeric(a == "c")`, `I(z^2)`, or `factor(z)` is re-evaluated at the level being set rather than held at its observed value, and such a model is g-computation on the model as specified. Without `.data` there is nothing to re-evaluate: the designs come from the outcome model's own model frame, which holds each such term at the values it was fit on. A term that has to derive the exposure's levels from the values it sees, such as `factor(a)`, is therefore rejected on that route, with `.data` named as the remedy.

`.by` A single unquoted variable naming a modifier to report effects within, or `NULL` (the default) to report the effect for the sample as a whole and nothing else. The

variable is looked up in `.data` when you supply one and in the outcome model's own model frame otherwise, and it must be a factor or a character vector with no missing values. The result then reports the effects it reports without `.by`, keyed by the group `"overall"`, followed by one set within each level of the modifier and one set contrasting each level against the first, and the `estimates` frame gains a `group` column naming the subgroup each row belongs to. See **Effect modification** for what those rows estimate and what they require.

`estimand` A character string specifying the causal estimand: one of `"ate"`, `"att"`, `"atu"`, `"atm"`, `"ato"`, or `"entropy"`. The available estimands depend on the exposure type: a binary or categorical exposure supports all six, while a continuous exposure supports only `"ate"`. For a categorical exposure, `"att"` and `"atu"` require a focal level (see `.focal_level`). See the **Estimands** section for the full support matrix. If `NULL`, the estimand is inferred from the weights in `outcome_mod`, which requires weights created with a propensity weight function such as `wt_ate()`.

`ps_link` **[Deprecated]** A character string specifying the link function used in the propensity score model: `"logit"`, `"probit"`, or `"cloglog"`. The link is always read from `wt_mod`, so `ps_link` can only restate what the fitted model already supplies and can simply be dropped. Supplying it warns.

The argument applies only to a binomial `stats::glm()` propensity score model on the binary path. A multinomial or continuous propensity score model has no link for `ps_link` to override, so a non-`NULL` value errors on both of those paths; leave it `NULL` there. `ps_link` cannot name a link other than the one `wt_mod` was fit with: `se_method = "linearization"` and `se_method = "robust"` error, and `se_method = "mestimation"` reports the resulting weights as inconsistent with the propensity score model.

`conf_level` Confidence level for intervals. Default is `0.95`.

`se_method` Method for standard error estimation. `"mestimation"` (the default) stacks the propensity score, outcome, and estimand estimating equations and returns the empirical sandwich variance. `"linearization"` uses the influence-function linearization of Kostouraki et al. (2024). Both account for the uncertainty of estimating the propensity scores.

`"robust"` does not, and is a diagnostic rather than an inference. It is the linearization route with the correction for having estimated the propensity score dropped, so it reports the sandwich the weighted outcome model computes for itself: the delta-method reading of `sandwich::vcovHC(outcome_mod, type = "HC0")`, exactly. Ignoring that the weights were estimated generally understates the variance, and understates it most where the weights are least stable, so this is offered for reading beside one of the other two rather than in place of it. A result fit this way says so when printed and marks the tables it coerces to; see **Standard errors as a diagnostic** below.

Which exposure types support which: `"mestimation"` supports every one of them; `"linearization"` and `"robust"` support a binary exposure alone. A binary or a categorical exposure has a sandwich for every fit it accepts. A few continuous fits have no sandwich, and the package computes no standard error for them: weights built with a `"kernel"` density are refused, and so are a

MASS::rlm() and an mgcv::gam() fit whose score the stacked system cannot write. See **Continuous propensity score models** below.

"linearization" and "robust" support only an outcome model of the exposure alone, fit with an intercept; a covariate-adjusted outcome model requires "mestimation". For a binary or categorical exposure, every method requires an outcome model that can represent a baseline, so a numeric no-intercept coding such as y ~ z - 1 errors on any of them. The standard errors of the conditional reading described under effects require "mestimation" as well: the linearization and robust routes store the outcome model unwrapped, so stats::vcov(), stats::confint(), and tidy() refuse effects = "conditional" there with a classed error rather than reporting the covariance the outcome model computed for itself.

effects — The presentation mode the result records, either "marginal" (the default) or "conditional". The marginal reading reports the population-averaged causal contrasts; the conditional reading reports the outcome model's coefficient surface. Both surfaces are computed whichever mode is named, so the argument settles which one the result presents and nothing else. causalgenerics::as_marginal() and causalgenerics::as_conditional() move a result between the two readings afterwards, and the accessors take an effects argument of their own for a single call.

One kind of fit has a single reading rather than two. A continuous exposure entering outcome_mod through more than one design column, such as a poly() or spline dose response, has no coefficient that is the effect of the exposure, because a curve has a different slope at every dose. Such a result records the conditional reading and says so once. ipw() refuses effects = "marginal" itself with an error of class propensity_ipw_effects_error, and the result declares the conditional reading as the only one it supports, so the accessors refuse it from the result class with an error of class causalgenerics_unsupported_reading_marginal. See the continuous exposure section below.

The covariance the conditional reading reports is the outcome block of the jointly estimated sandwich, which every route that stacks estimating equations attaches to the outcome model it stores: se_method = "mestimation" for a binary exposure, the categorical and joint routes, which run on M-estimation alone, and the continuous route under that method. A linearization or robust fit stacks no system and records no covariance of either kind, so stats::vcov() and stats::confint() error on its conditional reading, and printing it writes the coefficients under a note saying the standard errors are not reported rather than beside the ones the outcome model computed for itself.

.focal_level — For the categorical (multinom) method with the att or atu estimand, the focal exposure level. If NULL, it is taken from the focal_category attribute of the weights in outcome_mod. An explicit value overrides the attribute.

Value

Methods of ipw() return an S3 object of class ipw. That result class is shared across packages and its components, its print() method, and its as.data.frame() method are documented at causalgenerics::new_ipw(). propensity's methods fill every component; three of them take propensity-specific values:

`estimand` One of `"ate"`, `"att"`, `"atu"`, `"atm"`, `"ato"`, or `"entropy"`.

`se_method` One of `"mestimation"`, `"linearization"`, or `"robust"`.

`fit` The fitted M-estimator object when `se_method` is `"mestimation"`. Calling `stats::vcov()` or `generics::tidy()` on this object exposes the full stacked system of estimating equations, including the propensity score, outcome, and estimand parameters. It is `NULL` under `"linearization"` and `"robust"`, neither of which solves a system.

The result answers the standard model accessors, which causalgenerics registers for the shared class: `stats::coef()` and `stats::confint()` for the reported effects, `stats::vcov()` for their covariance, `stats::nobs()` and `stats::df.residual()` for the counts describing the fit, and `stats::weights()` for the `psw()` vector the outcome model was fit with. Coefficients are named for the effect measure and the contrast together, the contrast being the level a counterfactual mean belongs to or the pair of levels an effect measure compares, and for the subgroup as well where `.by` reports one row per subgroup. Which surface `stats::coef()`, `stats::vcov()`, and `stats::confint()` report follows the presentation mode the result records, described under `effects` above.

Under `se_method = "mestimation"` the stored `wt_mod` and `outcome_mod` are the models supplied, carrying their block of the joint sandwich. Calling `stats::vcov()` on either reports a covariance that accounts for the whole system having been estimated from the same data, rather than the one the model's own fitting routine reports. The models supplied are left as they were fit. Linearization solves no such system, so its component models are stored unchanged.

Workflow

`ipw()` is designed around a three-step workflow:

1. Fit a propensity score model (e.g., logistic regression of exposure on confounders).
2. Calculate propensity score weights for your estimand (e.g., `wt_ate()`) and fit a weighted outcome model.
3. Pass both models to `ipw()` to obtain causal effect estimates with correct standard errors.

You are responsible for specifying and fitting both models. `ipw()` handles the variance estimation.

Estimands

The available estimands depend on the exposure type:

Estimand	Binary	Categorical	Continuous
`"ate"`	yes	yes	yes
`"att"`	yes	yes (needs focal level)	no
`"atu"`	yes	yes (needs focal level)	no
`"atm"`	yes	yes	no
`"ato"`	yes	yes	no
`"entropy"`	yes	yes	no

A continuous exposure supports only `"ate"`; any other request errors. For a categorical exposure, `"att"` and `"atu"` require a focal level, supplied through `.focal_level` or taken from the weights (see the `multinom` method).

A binary exposure takes no focal level: `ipw()` always treats the second sorted exposure level as the exposed group, which is the second level of a factor, `TRUE` for a logical, and the larger of two numeric values. Because `"att"` and `"atu"` weights are mirror images of each other, weights built with `wt_att()` or `wt_atu()` on the other level carry values that are correct for the reversed roles, and `ipw()` rejects them rather than correcting them as though the roles matched. To target the other level, relevel the exposure so that it sorts second and refit both models.

The estimand a weighted outcome model was fit for is not necessarily the one a tool that averages its per-row predictions reports, so see **Standardizing model predictions** in `ps_tilt()` before taking such an average of a model fit for anything other than `"ate"`.

Effect measures

A binary or categorical exposure reports the counterfactual mean at each exposure level before the contrasts built from those means. Those rows carry the effect label `"mean"` and lead the table, one per level in level order with the reference level first. Each is the outcome model's prediction with every unit set to that level, averaged over the sample under the estimand's tilt, so a binomial outcome reports the marginal risk under each level and a continuous one reports the marginal mean. Reading a risk difference beside the two risks it is a difference of is what those rows are for: a difference of 0.05 is a different finding at risks of 0.02 and 0.07 than at 0.50 and 0.55.

For a binary exposure, the reported measures depend on the outcome model. A binary outcome (`stats::glm()` with `family = binomial()`) returns three measures:

- `rd`: Risk difference (marginal risk in exposed minus unexposed)
- `log(rr)`: Log risk ratio
- `log(or)`: Log odds ratio

A continuous outcome (`stats::lm()` or `stats::glm()` with `family = gaussian()`) returns only the difference in means (`diff`).

For a categorical exposure, the same measures are reported for each non-reference level against the reference (first) factor level, so a K-level exposure produces one block of measures per non-reference level.

The estimates table of either exposure carries a `contrast` column naming what each row is about: the level a mean row belongs to, and the pair of levels a contrast row compares, written `"1 vs 0"` or `"b vs a"` in the levels the models were fit on. A binary exposure names its one pair the same way a categorical one names each of its several, so the two tables are read alike.

For a continuous exposure, `ipw()` reports the exposure coefficients of the weighted marginal structural outcome model. A model with one exposure coefficient reports one row and no `contrast` column, and a dose-response curve such as `y ~ A + I(A^2)` reports one row per coefficient, with the `contrast` column naming the coefficient each row belongs to.

The `effect` label names the scale: `log(or)` for a logit link and `log(rr)` for a log link, since a coefficient there is a log odds ratio or a log risk ratio per unit of whatever column it multiplies. At an identity link the word depends on what the row claims. A model with one exposure coefficient reports `slope`, because that coefficient is the slope of the dose response everywhere. A model with several reports `coef`, because a curve has a different slope at every dose and no one of its coefficients is that slope; the row says which coefficient it is and leaves the dose response to be built downstream from `coef()` and `vcov()`.

Every term of the marginal structural model that reads the exposure must read the exposure and nothing else. The requirement is on what a term reads rather than on how it is written, so `A + I(A^2)` and `A + sin(A)` are admitted and `A * x1` is not: a term reading a covariate as well contributes a coefficient that depends on that covariate, and no row could name the effect it stands for. Read the whole coefficient vector from the returned `fit` object for a model this surface cannot report.

A basis matrix reads the exposure alone, so `poly(A, 2)`, `splines::ns(A, 3)`, and `splines::bs(A, 3)` are admitted on the same footing. One such term contributes one coefficient per basis column and each of them is a row, reported under `coef` at an identity link like any other multi-coefficient surface. The `contrast` column names the coefficient rather than the term, a distinction a basis makes and `A + I(A^2)` does not: the single term `poly(A, 2)` reports the contrasts `poly(A, 2)1` and `poly(A, 2)2`. A basis reading a covariate is a covariate term and contributes no row, and one reading both, such as `poly(A, 2):x1`, is refused with every other mixed term.

A marginal structural model built on a basis of the exposure requires `.data`. A model frame records the term rather than the variables inside it, so the frame of such a fit carries the basis matrix and no column holding the exposure, which is where `ipw()` reads the exposure from when `.data` is absent. Omitting it errors, and the message names the exposure and directs to `.data`. Supplying it does not refit the basis: the outcome design is read from the fit, and a design rebuilt from `.data` is rebuilt through the fitted terms object, whose `predvars` attribute records the basis that term was fit with.

Use `as.data.frame()` with `exponentiate = TRUE` to obtain risk ratios and odds ratios on their natural scale.

Effect modification

`.by` names a modifier and asks for the effect within each of its levels alongside the effect for the sample as a whole. The reported rows come in four blocks, in this order:

- the overall rows, unchanged from the fit without `.by`, the counterfactual means and then the contrasts, under the group label `"overall"`;
- the counterfactual mean at each exposure level within each level of the modifier, under a `"var = value"` label such as `"sex = female"`;
- one block of contrasts per level of the modifier, under the same labels;
- one block of contrasts per non-reference level against the reference level, which is the modifier's first level, under a `"var = value vs var = value"` label.

The last block reports no means of its own. Its rows compare the effects in two strata, and a difference of two effects belongs to neither stratum's population, so there is no level whose mean it would be.

A stratum's effect is g-computation restricted to that stratum: the outcome model predicts each unit's outcome at each exposure level, and the predictions are averaged over the units of the stratum, weighted by the estimand's tilt. What a stratum row estimates is therefore the effect in that stratum's own covariate distribution, and the last block is a difference of two such effects. Those two populations differ whenever the covariates are distributed differently across the strata, so a difference between strata is a difference between two conditional effects and not, on its own, evidence that the modifier changes the effect. It is the estimand's population that the tilt fixes, and the modifier's stratum that the indicator fixes, and the row reports the effect where those two meet: an `att` fit reports the effect among the exposed units of a stratum, not among all of its units. For a categorical exposure the tilt an `att` fit standardizes over is the focal level's own propensity score, so a stratum

row there reports the effect among the units of that stratum who look like the ones assigned the focal level.

A categorical exposure already reports one contrast per non-reference level, and `.by` crosses those contrasts with the strata: every block holds each contrast on each reported measure, so a K-level exposure with S strata reports the whole-sample block and then S + (S - 1) blocks of (K - 1) contrasts, beside S blocks of K means. A row is named by three things, the measure, the contrast, and the subgroup, and the `estimates` frame carries `contrast` and `group` in that order after `effect`. Within a contrast block the ordering is the one an ungrouped fit already uses, contrast-major and measure-minor, so a contrast's measures sit together; the blocks themselves run group-major.

The stratum rows and the stratum contrasts report the risk difference and the log risk ratio for a binary outcome, and the difference in means for a continuous one. They report no odds ratio. An odds ratio is noncollapsible, so a difference of two of them moves with the outcome distribution in each stratum whether or not the effect does, and reporting it as effect modification would invite exactly the reading it does not support. The overall rows keep all three measures.

Three requirements apply. The modifier must be a factor or a character vector with no missing values: a missing value names no subgroup, and a continuous variable names no fixed set of them, so cut it into groups first. A level no unit carries is dropped rather than refused, since the fits drop it too and an empty stratum has no mean to standardize. Every one of the modifier's remaining levels must hold every exposure level, since a stratum in which nobody took one of them identifies no contrast against it and the outcome model would extrapolate one from the strata that do; the remedy is a coarser modifier rather than a refit. Finally, `.by` requires `se_method = "mestimation"` and a binary or categorical exposure: the stratum means and their contrasts are parameters of the stacked estimating equations, which is what puts them in the same sandwich as everything else, and a continuous exposure reports its marginal structural model's own coefficient rather than a set of standardized means to modify. Each of those errors.

An outcome model with no term reading the exposure and the modifier together warns rather than errors. The stratum effects are g-computation on the model as specified, so such a model forces the same effect on every stratum up to the covariate distribution each one has, and the modification it reports is a property of the model rather than of the data. The fit still returns.

Stabilizing the weights on the modifier is worth doing here. A stabilized weight divides by the propensity score and multiplies by a numerator, and the numerator may condition on any variable the outcome model the effects are computed from already reads, without changing what is estimated. The modifier is such a variable by construction, so a numerator conditioning on it leaves the estimator consistent for the same stratum effects and changes only how variable the weights are. That is where it pays: the part of the exposure's variation the modifier explains cancels between the numerator and the propensity score, and the weights tighten by more than the default numerator tightens them.

The converse is the constraint, and it is the reason this is a statement about the model rather than a free choice. A numerator conditioning on a variable the outcome model does not read changes the estimand rather than the precision, because the pseudo-population it builds is one in which that variable still predicts the exposure, and confounding by it remains (Robins et al. 2000; Cole and Hernán 2008; Hernán and Robins 2020, Chapter 12). Condition the numerator on the modifier and on nothing the model does not hold. The same rule governs an exposure-by-modifier interaction reported without `.by`, since what matters is what the reported model reads and not which argument asked for the rows.

Build the numerator with `wt_ate()`'s `stabilization_score`, evaluated at the exposure each unit actually took; see **Stabilization** in `wt_ate()` for the recipe. The sandwich treats a supplied score as

a known constant, so the standard errors do not account for the numerator model having been fitted. The default stabilizer is estimated in the stacked system instead, which is one parameter wider as a result. There is a third way: pass the fitted numerator model to `stabilize` itself, a `stats::glm()` of a binary exposure or an `nnet::multinom()` of a categorical one, and its own equations join the stack, which is what a supplied score has none of. That accounting has something to say only where the reported model is not saturated in the variables the numerator reads; a numerator on a modifier the reported model interacts with fully divides out of every cell, so the estimates and the standard errors are the ones the unstabilized weights give.

A stabilized fit reporting effects within the strata of a modifier, and carrying the default numerator, warns with class `propensity_ipw_by_stabilizer_warning` and returns: the default numerator conditions on nothing, which is consistent for the same stratum effects and leaves precision on the table. A numerator model the caller supplied is read the same way, since a fitted model carries the terms it read: one that never reads the modifier warns with the same class, naming the terms it was built on, and one that reads the modifier, on its own or through a transformation of it, is silent. A `stabilization_score` says nothing either way, being a vector of numbers that does not carry what it conditions on.

Every reported row is a parameter of one stacked system solved on one sample, so the covariance couples them: `stats::vcov()` reports nonzero entries between the overall rows, the stratum rows, and the stratum contrasts, and the standard error of a stratum contrast accounts for the two strata having been estimated together. Fitting each stratum on its own subset would report the same stratum effects with no covariance between them, leaving the difference between two strata untestable.

Joint exposures

`causalgenerics::joint_exposure()` crosses two discrete treatments into one categorical exposure and records the crossing on the vector. The result is a factor, so `ipw()` fits it exactly as it fits any exposure over the same cells. What the declaration changes is what gets reported: instead of each cell against the reference cell, the result is written in the two treatments.

- One row per cell, under the effect label `"mean"`, carrying the counterfactual risk (or mean) had everyone been set to that cell. The cell is named in the `contrast` column and the group is `"overall"`.
- The simple effects: each treatment's effect within a fixed level of the other, with the `contrast` naming the treatment and the level it is set to (`"a: 1 vs 0"`) and the `group` naming the level the other treatment is held at (`"e = 1"`). These include the comparisons that are not against the reference cell, which the vs-reference reporting cannot express at all.
- The interaction: the difference between two of the first treatment's simple effects, keyed by the two levels of the second being compared (`"e = 1 vs e = 0"`). On the risk difference scale this is the additive interaction, and on the log risk ratio scale the multiplicative one, which is the log of the ratio of the two risk ratios.

A 2-by-2 crossing therefore reports fourteen rows for a binary outcome and nine for a continuous one. The interaction is reported once, under the first treatment's framing, rather than twice: interaction is symmetric in the two treatments, so the difference between the first treatment's two simple effects is the difference between the second treatment's two, and reporting both would be one quantity under two labels.

No odds ratio is reported anywhere on this surface. It is noncollapsible, so neither a simple effect reported beside one nor a difference of two of them says what it appears to.

Every row is a parameter of the same stacked system, so the covariance couples them: the interaction covaries with each simple effect it is built from, and with each cell mean underneath those. The cell means are the marginal-mean block the categorical path already estimates, and the simple effects and interaction rows are contrasts over that block, so an interaction equals the corresponding double difference of the cell means exactly rather than to within a solver residual.

Two restrictions apply. A declared crossing is reported for the `"ate"` estimand alone: every cell mean on this surface standardizes to one population, and a focal estimand standardizes each of them to a population the simple effects and the interaction are not defined over, so weights built for anything else error. And `.by` cannot be combined with a declared crossing: effect modification of a joint intervention is a three-way question, the interaction between two treatments within the levels of a third variable, which this surface neither answers nor sensibly projects. Either restriction is escaped the same way, by dropping the declaration with `factor(x)` and reporting each cell against the reference cell.

Two treatment models instead of a declared crossing:

A joint intervention can also be weighted through the sequential factorization it really has, $f(A|L)f(E|A,L)$: one treatment model per treatment, recorded with `joint_wt_models()`, and the product of their weights, built with `wt_joint()`. Pass the container as `wt_mod` and an outcome model that reads both treatments as separate columns.

The reported surface is the one above, row for row and label for label: the same cell means, simple effects, and interaction, under the same `effect`, `contrast`, and `group` values. The two routes estimate the same estimand through different models of the same propensity score, so they agree as estimators rather than to the last bit, and they coincide when the two parameterizations are saturated in the same covariates.

Either treatment may have more than two levels, fit with `nnet::multinom()` and weighted with categorical `wt_ate()` weights. Such a component is stacked under the multinomial score its own fit solves, which reads no order into the pair, so a categorical treatment sits in either position. The surface grows with the crossing rather than changing shape: a 2-by-3 crossing reports twenty-four rows for a binary outcome and fifteen for a continuous one, being the six cell means, each treatment's simple effects within each level of the other, and the first treatment's interaction against each non-reference level of the second. A categorical component needs no stabilization, and a caller who asks for it anyway is answered: the marginal proportions of its levels, a `stabilization_score`, and a fitted `nnet::multinom()` numerator each get the block described under **Stabilizing numerators, one per component**. A categorical treatment may also be paired with a dose, whose vocabulary names one level contrast per non-reference level; see the dose section below.

A `nnet::multinom()` fit to two levels is a logistic regression solved by a different optimizer, and it is stacked as one. `joint_wt_models()` records such a component as `"categorical"`, after the model's class, but the block written for it is the binomial score, the weight factor is the binary one `wt_ate()` already builds from such a fit, and the pair reports the same fourteen-row two-by-two surface a pair of binomial `stats::glm()` fits reports.

Prefer the two-model route when the two treatments call for different adjustment sets, or when the dependence of the second treatment on the first is what you want to model directly, since each treatment then gets its own model and its own covariates. Prefer the declared crossing when one model over the cells is the natural specification. The factorization itself is validated when the container is built rather than here: `joint_wt_models()` refuses a second model that does not condition on the first treatment, and a pair that condition on each other.

Both treatment models are stacked alongside the outcome model, so the weights entering the outcome score are rebuilt from both propensity score blocks on every evaluation and the reported standard errors account for having estimated both. The same two restrictions apply: the `"ate"` estimand only, and no `.by`. Standard errors come from M-estimation alone.

A joint intervention with a dose:

The second treatment may be a dose rather than a second discrete treatment, recorded with any of the classes the single-treatment route reads a conditional density from, and weighted with a stabilized density ratio, which `wt_joint()` requires of a continuous component. A dose has no cells, so there is no crossing to report counterfactual risks over: the surface is the marginal structural model's own coefficients, which for an identity-link model are exactly the weighted fit's coefficients.

The first treatment may be binary or categorical. What the position settles is the dose: it is the second factor of the factorization, the treatment whose model conditions on the other, and nothing here carries the density of a first factor that is one, so a dose in the first slot and a pair of doses are both refused by type with an error of class `propensity_ipw_exposure_error`.

Which of two surfaces a fit reports is decided by the marginal structural model alone. A model written in bare treatment terms and fit with an intercept reports the vocabulary surface below, whose rows name the treatment each one varies and where it is evaluated. Every other treatment-reading model reports the coefficient surface, whose rows are named after the coefficients they report.

The vocabulary surface:

For `y ~ a * e` with `a` binary and `e` a dose, three coefficients carry causal content and each is a row:

- the binary treatment's effect at a dose of zero, with the `contrast` naming the treatment and its levels (`"a: 1 vs 0"`) and the `group` naming where it is evaluated (`"e = 0"`);
- the dose's slope at the binary treatment's reference level (`"e: per unit"`, `"a = 0"`);
- the interaction, the change in the binary treatment's effect per unit of the dose, keyed by the one-unit step of the other treatment (`"a: 1 vs 0"`, `"e + 1 vs e"`).

The `effect` column names the scale, keeping the vocabulary each treatment type already uses: a level contrast is a `diff` and a per-unit change in the dose is a `slope` under an identity link, and both are `log(or)` under a logit link and `log(rr)` under a log link. An additive model, `y ~ a + e`, forces one effect for the binary treatment at every dose and one slope at either of its levels, so it reports two rows under the group `"overall"` and no interaction.

A categorical first treatment reports those same three readings with the level contrasts run out over the levels it has: one contrast row per non-reference level at a dose of zero, one slope row at the reference level, and one interaction row per non-reference level. For `y ~ z * e` with `z` taking `"lo"`, `"mid"`, and `"hi"`, that is five rows, in the outcome design's column order:

- `"z: mid vs lo"` at `"e = 0"` and `"z: hi vs lo"` at `"e = 0"`;
- `"e: per unit"` at `"z = lo"`;
- `"z: mid vs lo"` and `"z: hi vs lo"`, each at `"e + 1 vs e"`.

A binary treatment is the one non-reference level of that rule rather than a case of its own, so the three rows above are this surface at two levels. An additive model reports the level contrasts and the slope under the group `"overall"` and no interaction, K rows for K levels. Every row is one coefficient of the weighted fit, so the surface grows by a row per level rather than by a table of contrasts, and per-arm dose slopes, which are not coefficients of this model, are built downstream from `coef()` and `vcov()`. No guard here reads how thin an arm is: a level the

dose model rarely sees has its conditional density read far from any data, and the product weight carries that as a heavy tail. Check the weights' effective sample size and their spread within each level, as for any joint weight, before reading the rows.

Those three readings hold of a model that is linear in each treatment and carries an intercept, and of no other, which is why such a model reports this surface and no other model does. A model fit without one, written `- 1` or `+ 0`, reports the coefficient surface: dropping the intercept expands a factor treatment to an indicator for every level rather than for the non-reference levels alone, and where the columns survive it, as a 0/1 numeric treatment's do, the forced zero moves what the coefficients mean instead, so no row of this vocabulary would be true of the fit either way.

The columns are read as well as the formula: a factor treatment under a coding other than treatment contrasts leaves the terms bare while rescaling or recentering the column, so its coefficients are no longer the effects these rows name, and such a fit is refused rather than reported in this vocabulary. Refit the outcome model with the treatment as an unordered factor under treatment contrasts. A treatment with two levels also has a numeric coding whose bare term contributes that column, 0 for the reference level and 1 for the other; a treatment with more than two levels has none, so there it is the unordered factor alone. The refusal belongs to this surface rather than to the fit. A model the vocabulary has no reading for reports the coefficient surface instead, and a model written without an intercept is one such, so an ordered factor there contributes an `a.L:e` row named after the column it multiplies, claiming nothing about a coding.

An interaction of the treatment with the dose written where the dose has no term of its own, as `y ~ a + a:e`, is refused here too, as is a crossing carrying no treatment term either, `y ~ a:e`. R codes a factor in such an interaction with one indicator per level rather than with the contrasts it carries, so the design gains a reference-level column no row names and each column after it holds the level before it. Write the crossing as `y ~ a * e`, which gives the dose a term of its own and reports on this vocabulary.

The dose column is read the same way, against the dose the treatment models were fit to. A dose transformed in the outcome model's own formula, as `I(e / 10)` or `scale(e)`, leaves that column alone and is no longer a bare term, so the fit reports the coefficient surface. A dose rescaled or recentered in the data after the treatment models were fit is refused on this surface instead: the models have to agree on what the variable is, and the weights the outcome model carries were built for the dose those models hold. Working on a rescaled dose is supported either by rescaling it before every model is fit or by writing the transformation into the outcome formula. Neither refusal reaches the coefficient surface, which reports whatever columns the model holds and names each row after its own column: a bare-term model on a rescaled dose fit without an intercept, `y ~ a * e - 1`, reports the coefficients and standard errors that reparameterization leaves.

The coefficient surface:

A marginal structural model carrying a transformed or a basis treatment term, such as `y ~ a * sin(e)`, `y ~ a + e + I(e^2) + a:e`, or `y ~ a * splines::ns(e, 3)`, reports one row per treatment-reading coefficient. The `contrast` column carries the coefficient name exactly as the fit writes it, and there is no `group` column at all: the surface makes no claim about where a row is evaluated, because for a curve there is no one place. Build an interpretable dose response downstream from `coef()` and `vcov()`, whose covariance between the rows is what that takes. Marginalizing the curve over the observed doses is a separate estimand that this package does not compute, here as on the single-treatment route: use the **marginaleffects** package on the result, `avg_slopes()` for slopes, `avg_comparisons()` for contrasts, and `avg_predictions()` for causal dose-response functions.

Unlike the single-treatment route, this one announces nothing and refuses nothing for such a fit. There the surface a basis exposure leaves is the only reading the result has, so the reading is declared; here both readings are computed whatever the marginal structural model is, and `effects` is answered either way.
The `effect` label follows the outcome link, `coef` at an identity link and the same `log(or)` and `log(rr)` as everywhere else. `coef` is a word of its own rather than `diff` or `slope`, both of which carry an evaluated-at claim on the vocabulary surface.

Covariates and mixed terms:
A covariate entering on its own is admitted on either surface and contributes no row, however many columns it expands to: it adjusts the marginal structural model and the surface has nothing to say about its coefficient. A term reading a treatment and a covariate together is refused on both, since its coefficient is a change in an effect per unit of that covariate and no row could name the effect it stands for. Nothing here is standardized over the covariates, which is what the declared crossing does instead.
Unlike the single-treatment route, this one needs no `.data` for a basis marginal structural model. Without one, each treatment and each treatment design is read off the model that owns it, so the outcome model frame is never asked for a column it does not hold.
A `.data` the caller does supply is the frame every design here is rebuilt from, on the terms the single-treatment routes rebuild theirs: both treatment columns, both treatment designs, each component's numerator design, and the counterfactual designs are built over the rows every model read and under the coding each fit recorded. A column one of those fits reads that is missing from `.data` errors with class `propensity_columns_exist_error`, and one supplied as another type errors with class `propensity_ipw_data_error`, naming the fits that recorded it, a treatment model by the treatment it fits. A set of fits made under `na.action = na.exclude` is reconciled with the frame they were given by restricting `.data` to the rows they analyzed.
The dose model contributes its coefficients and the conditional variance its density ratio divides by, all as parameters of the same stacked system. Each component's stabilizing numerator is estimated in a block of its own; see **Stabilizing numerators, one per component**.

Stabilizing numerators, one per component:
Either component may be stabilized, and each component's numerator is estimated in its own block of the stacked system, so the standard errors account for that numerator having been fitted rather than reading it as a constant. What the block holds follows what the component's weights record. A binary component stabilized by the default numerator contributes the one marginal proportion that numerator is, a categorical one the $k - 1$ free proportions its levels leave, and a dose stabilized by the default numerator the exposure's two marginal moments. A component stabilized on a fitted model passed to `wt_ate()`'s `stabilize` contributes that model instead: one parameter per coefficient for a binary component, one run of them per non-reference level for a categorical one, and for a dose one per coefficient plus one for the spread its density is read at. A dose stabilized with `numerator = "integrated"` is built from the dose block and the data alone and contributes nothing, and an unstabilized discrete component contributes nothing either.
Those parameters are named `stab_<component>_<parameter>`, where `<component>` is the name the component was recorded under in `joint_wt_models()`: a joint system carries two components, and a name saying only which role a parameter plays would not say whose. A categorical component composes that prefix with the multinomial layout its own block takes, `<level>:<term>` for a fitted numerator and the level alone for a marginal one, so a treatment `z` with levels `lo`, `mid`, and `hi` reports `stab_z_mid` and `stab_z_hi` for the marginal proportions and `stab_z_mid:(Intercept)`, `stab_z_mid:v`, `stab_z_hi:(Intercept)`, and `stab_z_hi:v`

for a numerator of v.

A numerator model is read through the guards its own single-treatment route reads it through. A binary component's is read as an unpenalized binomial fit at a logit, probit, or cloglog link; a categorical component's as an nnet::multinom() fit that declares the treatment's levels in the order the component's own model declares them; and a dose's goes through the registry under **Continuous propensity score models**. A multinomial numerator fit under another level order, a model of a response other than the treatment it stabilizes, or one fit to a different set of observations errors with class propensity_ipw_numerator_error; one fit with case weights errors with class propensity_ipw_ps_weights_error. Every numerator design here is one of the designs a .data the caller supplies rebuilds, on the terms the single-treatment routes rebuild theirs, so a numerator that keeps no model frame and whose fitting call can no longer be evaluated errors with class propensity_ipw_data_error without one, and is rebuilt from .data with one.

Both components' numerators arrive in the same argument, one stabilize per component, so every refusal about one names the component it was built for beside that argument, stabilize for z. wt_mod here is the container the two treatment models arrived in rather than a model anyone could refit, so a refusal reading a numerator against its treatment model names that component too.

What a numerator may condition on is a statement about the reported model rather than a free choice, and it is the same statement here as for a single treatment. A numerator may condition on any variable the marginal structural model already reads, such as a modifier v: $P(Z|V)$ for a discrete treatment and $f(A|V)$ for a dose leave the estimator consistent for the same effects and only tighten the weights. A numerator conditioning on a variable the reported model does not read changes what is estimated rather than the precision, because the pseudo-population it builds is one in which that variable still predicts the treatment, and the estimates are then read with v or not at all. The accounting a stacked numerator adds has something to say only where the reported model is not saturated in the variables the numerator reads: a numerator that divides out of every cell of the reported model moves neither the estimates nor their standard errors.

The dose model and the ratio its weights are:

The dose model goes through the registry described under **Continuous propensity score models**, so the classes and links this route stacks are the ones the single-treatment route stacks: an stats::lm(), a gaussian stats::glm() fit with an identity or a log link, a MASS::rlm() fit with one of the psi functions MASS supplies, and an mgcv::gam() of a gaussian family read at its own penalized score. A gaussian dose model fit with another link errors with class propensity_ipw_link_error, and a robust or an additive fit whose score the system cannot write raises the same conditions it raises for a single dose. The limitations of the additive route are the ones listed under **Continuous propensity score models**, and they apply here unchanged. Every score this route stacks is the unweighted one, so a component model fit with prior case weights is not at the root of the block written for it and errors with class propensity_ipw_ps_weights_error, naming the component rather than the container the two models arrived in. Refit that component without weights.

The dose component's weights may be any ratio wt_ate() builds: a heavier-tailed .density, numerator = "integrated", or a single .sigma the caller fixed. wt_joint() records each component's ratio on the product, and the stacked system rebuilds the dose factor from that record, so what is differentiated is the weight the outcome model was fit with. An integrated numerator is built from the dose block and the data alone, so it adds no stabilization parameters; it is the conditional density averaged over the units the dose model was fit to and read at points spanning their dose, so it is rebuilt over those rows the way a single dose's is, and a dose model

that keeps no model frame and whose fitting call can no longer be evaluated errors with class `propensity_ipw_data_error` under it, `.data` or no `.data`. A fixed scalar `.sigma` is a known constant, so the dose block is its coefficients alone and none of that number's uncertainty is carried. A spread supplied for each observation has no counterpart in the system and errors with class `propensity_ipw_sigma_error`.

A component built with a `stabilization_score` is rebuilt from the score itself. `wt_joint()` records each component's score on the product, and the sandwich treats it as a known constant here exactly as it does for a single treatment, so it adds no stabilization parameters and none of its uncertainty is carried. A product built by a version of this package that recorded no score, one assembled by hand, or one subset after it was built, which drops a score held per observation, says a component was stabilized without saying what by: the system stands that component's own marginal numerator in, and the weight-consistency check reports the difference, naming the components whose numerator was stood in for. Rebuild such weights with `wt_joint()` to have the score read back. A record left whole over fewer rows is refused instead: a model frame that drops incomplete rows re-attaches the record as it was, leaving a component's score at the length the product was built at, and a score that describes other observations than the ones being weighted errors with class `propensity_ipw_stabilization_score_error`.

Dose weights built with a `"kernel"` density are refused with class `propensity_ipw_se_method_unavailable_error`. The bandwidth is not a function of the parameters that the sandwich could differentiate, and the package has no second method to fall back on, so the refusal points at building the dose weights from a density the stacked system can write, or at bootstrapping the whole fit by hand.

Variance estimation

By default (`se_method = "mestimation"`), standard errors are computed by M-estimation. The propensity score model, the weighted outcome model, the marginal means, and the effect contrasts are stacked as a single system of estimating equations, and the empirical sandwich variance of that joint system is used. The stacked marginal means are standardized to the estimand's tilted target population, so a non-`ate` estimand with a covariate-adjusted outcome model reports the contrast for that target population rather than the full-sample average. Stacking the models accounts for the uncertainty introduced by estimating the propensity scores, avoiding the underestimated standard errors that arise from treating estimated weights as fixed. See Stefanski and Boos (2002) for the M-estimation framework.

M-estimation standard errors are available for all exposure types: binary (from a `stats::glm()` propensity score model), categorical (from a `nnet::multinom()` model), and continuous (from an `stats::lm()`, a gaussian `stats::glm()`, a `MASS::rlm()`, or an `mgcv::gam()` model; see **Continuous propensity score models** below). The `atm` weight `pmin(e, 1 - e)` is not differentiable at a propensity score of `0.5`; deli's central finite difference straddles the kink there and averages the one-sided slopes. Its effect on the variance is negligible unless many observations sit at exactly `0.5`.

A binary exposure's propensity score model is read as an unpenalized logistic fit, whatever the standard error method: the stacked system solves that score, and both the linearization and the robust diagnostic differentiate it. An `mgcv::gam()` fit of a binomial family is not at the root of that score but of a penalized one, so it is refused with class `propensity_ipw_se_method_unavailable_error` rather than described by a standard error for a model that was not fit. What is refused is the standard error and not the weights: `wt_ate()` reads the same fit and builds them from its fitted probabilities. Write the shape each smooth term carries out as columns of a `stats::glm()` formula, or bootstrap the whole fit by hand. An additive dose model is stacked at its penalized score instead, as described under **Continuous propensity score models**.

M-estimation standard errors for a categorical exposure allocate memory roughly linearly in the number of observations, on the order of 70 to 90 kilobytes per observation for a single fit, so a sample of 10,000 observations allocates on the order of hundreds of megabytes. The cost comes from the stacked estimating-equation machinery rather than any single term, and for very large samples you can expect long, garbage-collection-heavy fits. This is expected behavior.

Stabilizing numerators for a discrete exposure:

A numerator estimated by a model the caller passed to `wt_ate()`'s `stabilize` is estimated here as well, whichever type the exposure is. A binary exposure's numerator model contributes one parameter per coefficient, named `stab_<term>`, and a categorical exposure's contributes one per coefficient of its `nnet::multinom()` fit, named `stab_<level>:<term>`, so that the level a coefficient belongs to is read off the name. The multinomial numerator's own score is stacked alongside the propensity score model's, written as the same multinomial estimating equation, so the standard errors account for the numerator having been fitted rather than reading it as a constant. The default stabilizer contributes what it estimates instead: the single marginal proportion `stab_pi` for a binary exposure, and the K - 1 free marginal proportions `stab_<level>` for a categorical one. A `stabilization_score` contributes nothing at all, being a number the caller fixed. What a dose's numerator contributes is described under **Continuous propensity score models**.

Reading an estimated numerator as known would be conservative for these weights rather than wrong, since stabilization is offered for the `ate` estimand alone (Lunceford and Davidian 2004; Kostouraki et al. 2024), and Shu et al. (2020) stack the numerator and the denominator of stabilized weights for that reason. Stacking the multinomial score therefore tightens the inference rather than rescuing it. Neither result isolates the numerator's own contribution, and Kostouraki et al.'s tutorial is written for a binary exposure, so what they give is the direction of the correction rather than a bound on its size.

A numerator model's design is one of the designs `ipw()` rebuilds when `.data` is supplied: it is rebuilt over the rows every model read and under the coding the fit recorded, so a column only the numerator reads is asked of `.data` and is held to the type the fit recorded for it. A `.data` missing such a column errors with class `propensity_columns_exist_error`, and one supplying it as another type with class `propensity_ipw_data_error`. Without `.data` the design is read off the fit itself, and a fit that keeps no model frame rebuilds one by re-evaluating its fitting call. A `stats::glm()` fit with `model = FALSE` is such a fit, so one made inside a function whose frame is gone errors with class `propensity_ipw_data_error`, and the remedy is to supply `.data`.

A multinomial numerator model is held to the propensity score model's levels in the propensity score model's own order. Everything its block contributes is positional: the coefficients are stacked level-major, and the softmax the numerator is rebuilt from reads the first level as the reference the others are contrasted with, so a fit made under another order is a different parameterization rather than a permutation of this one. Such a fit is refused with an error of class `propensity_ipw_numerator_error`, which is also what a model of a response other than the exposure, or one fit to a different number of observations, errors with. One fit with case weights errors with class `propensity_ipw_ps_weights_error`, for the reason a weighted propensity score model does. `nnet::multinom()` keeps no model frame, so the design its coefficients were fit over is recovered by re-evaluating the fitting call; a fit whose call can no longer be evaluated, such as one made inside a function whose frame is gone, errors with class `propensity_ipw_data_error`, and the remedy is to supply `.data`, which the numerator's design is rebuilt from.

A categorical exposure reports M-estimation standard errors alone, so a multinomial numerator model meets neither of the other two methods. The linearization's refusal of a fitted numerator model, below, and the robust diagnostic's reading of every weight as a known constant are both

matters for a binary exposure.

Continuous propensity score models:

A continuous exposure's weights are a ratio of densities centered on the conditional mean the propensity score model fits, so the stacked system carries that model's own score. Four classes supply one: an `stats::lm()`, a gaussian `stats::glm()` fit with an identity or a log link, a `MASS::rlm()` fit with one of the psi functions MASS supplies, and a gaussian `mgcv::gam()` fit with one of the same two links. A model fit with a family that is neither gaussian nor binomial is neither a dose model nor a binary propensity score model, and errors with class `propensity_model_family_error` naming the family it has. A gaussian model fit with any other link is refused by name, because the coefficients its iteration stops at are not a tight enough root to seed the stacked solve from; refit it with one of the two links or as an `stats::lm()`. Any other class errors, since solving it here would report a propensity score model that was never fit. Each class is read as the class it is rather than by what it inherits from, so a subclass of a supported fit errors as well.

A `MASS::rlm()` is stacked as the psi score its coefficients are the root of, clipped where the fit itself clipped. Each of the three psi functions MASS supplies has a loss of its own in the stacked system: `MASS::psi.huber()`, `MASS::psi.bisquare()`, and `MASS::psi.hampel()`. `rlm()` clips the residual divided by its scale estimate at the psi's own constants, and the stacked score clips the raw residual, so each constant is multiplied by the scale the fit settled on. Those constants are read off the psi the fit carries, which is where `rlm()` records a caller's choice of them, so a retuned psi is stacked at the values it was retuned to. That scale enters the location score as a known constant. It solves an equation of its own that the stacked system does not write, so none of its sampling variability is propagated, and the standard errors are those of a psi score read at a scale treated as fixed. The spread of the conditional density is a separate quantity, and comes from the estimator its family asks for, there as for every other class.

`MASS::psi.bisquare()` and `MASS::psi.hampel()` redescend, and that is worth knowing about the standard errors they get. A redescending psi's estimating equation has more than one root, so there is no single set of coefficients it identifies and the solve reports whichever root it is seeded at. The seed here is the fit's own coefficients, so what comes back is the root the user has, and the sandwich is the covariance of that root read locally. It carries no claim about the other roots, and it does not describe the sampling behavior of a procedure that might have converged to one of them instead. Use a bootstrap of the whole fit if that behavior is what you need.

Weights built with `.sigma = fit$s`, the robust scale, are read as the fixed spread they are, but that combination is usually a poor choice: the robust scale resists the very residuals the density is meant to spread over, so on contaminated data the conditional density is far narrower than the residuals are, and the ratio can collapse onto an effective sample size of about one. Leave `.sigma` unset unless you have a reason to hold the spread at a particular value.

Three robust fits are refused. One fit with a psi function of the caller's own has no loss here to write its score as, and one fit with `method = "MM"` reaches its coefficients through a high-breakdown start that decides which root of a redescending psi the fit finishes at and supplies the scale it clips at, neither of which is an equation this system writes. A sandwich read locally at those coefficients would therefore not describe how an MM fit behaves across samples. Both error with class `propensity_ipw_robust_psi_error`. One whose iteration stopped short of its own tolerance is the root of nothing, and errors with class `propensity_ipw_convergence_error`. The first points to a refit with one of the three psi functions or to a bootstrap of the whole fit written by hand, the second to the default `method = "M"`, and the last to a larger `maxit` or a looser `acc`, which is what a fit needs to reach its own tolerance.

An `mgcv::gam()` is stacked as the penalized least squares it is, with its smoothing parameters held

at the values the fit reports. Its design is the smooth basis it was built on rather than the columns its formula names, and its coefficients are the root of the least-squares score of that basis less the penalty those smoothing parameters define. An `mgcv::bam()` is the same fit computed for larger data and is read the same way. The additive part of an `mgcv::gamm()` fit is not: it carries the `gam` class alone, without the `glm` and `lm` classes `ipw()` reads a model through, and errors with class `propensity_method_error`. Refit it with `mgcv::gam()` or `mgcv::bam()`. A penalty attached to a parametric term, such as one from `paraPen`, belongs to no smooth term and is not one this route can place, so such a fit errors with class `propensity_ipw_se_method_unavailable_error`. A fit made under a smoothing floor from `min.sp` errors with the same class, because mgcv adds that floor to the penalty and leaves it out of the smoothing parameters the fitted object reports, so nothing in the fit records the penalty it was made under; refit without `min.sp`, which `mgcv::bam()` ignores in any case.

Four limitations come with the additive route, and each is a property of the method rather than of a particular fit.

1. The variance conditions on the smoothing parameters rather than estimating them, so none of the uncertainty of having chosen them is carried. A simulation put that at roughly three percent of the standard error at `n = 500`, roughly seven percent at `n = 200`, and less at larger samples. A fit handed its smoothing parameters and a fit that chose them report the same standard error here, which is the same limitation stated as an equality.
2. What the propensity score block reports is the frequentist variance of a penalized fit rather than the Bayesian covariance mgcv reports by default, so `vcov()` on the stacked fit reads like `vcov(fit, freq = TRUE)` and will not match `vcov(fit)`.
3. The envelope is the one every other continuous fit has: a gaussian family, an identity or a log link, and no prior weights. A fit outside it is refused where every other model of the wrong family, link, or weighting is.
4. Coverage degrades as the weights grow heavier tails, and the interval is optimistic once the effective sample size falls much below 60 percent of the observations. That holds whatever the dose model is: the same simulation found it for a correctly specified `stats::glm()` on the same data.

Every density `wt_ate()` builds a ratio in is stacked as it was built: the normal, Laplace, and Student t families, and a density you wrote yourself, under either the marginal or the integrated numerator. `ipw()` reads the family, the numerator, and the spread off the record the weights carry and rebuilds the same ratio at each value of the parameter vector, so the weights it differentiates are the weights the outcome model was fit with. A marginal numerator contributes the exposure's two marginal moments to the parameter vector; an integrated one is built from the propensity score block and the data alone and contributes none. That numerator is the conditional density averaged over the units the propensity score model was fit to, read at points spanning their exposure, so it is rebuilt over those rows even where `.data` leaves the rest of the system standing on fewer of them. Rebuilding it needs the design those rows enter through, which `.data` cannot stand in for: a propensity score model that keeps no model frame, made inside a function whose frame is gone, errors with class `propensity_ipw_data_error` under `numerator = "integrated"`, and the remedy is to refit it where its data is available. Weights carrying no such record, including any written with `psw()` by hand, are read as the normal ratio the package built before the record existed.

A numerator estimated by a model the caller passed to `wt_ate()`'s `stabilize` contributes that model: one parameter per coefficient, named with a `stab_` prefix so the terms it shares with the propensity score model stay apart from it, and one for the spread its density is read at, `sigma2_n`.

That parameter holds a squared scale rather than a variance, which are the same number only for a family whose scale is its standard deviation. Its score, and the equation that spread is the root of, which is the residual moment or the family's own score depending on what the density asks for, are stacked alongside the propensity score model's, so the standard errors account for the numerator having been fitted, which is what separates this from a `stabilization_score`: a score the caller computed is carried as a known constant and contributes no parameter at all. The numerator model is read through the registry above, so a class, family, or link whose score this system cannot write is refused there in the terms that registry refuses a propensity score model in, naming `stabilize`. A model of a response other than the exposure, or one fit to a different set of observations, errors with class `propensity_ipw_numerator_error`. One fit with case weights errors with class `propensity_ipw_ps_weights_error`, for the reason a weighted propensity score model does: the block written for it is its unweighted score, and its coefficients are not at the root of that.

A dose numerator model's design is one of the designs `ipw()` rebuilds when `.data` is supplied, on the terms a discrete exposure's is: it is rebuilt over the rows every model read and under the coding the fit recorded, so a column only the numerator reads is asked of `.data` and is held to the type the fit recorded for it. A `.data` missing such a column errors with class `propensity_columns_exist_error`, and one supplying it as another type with class `propensity_ipw_data_error`. Without `.data` the design is read off the fit itself, and a fit that keeps no model frame rebuilds one by re-evaluating its fitting call. A `stats::lm()` or `stats::glm()` fit with `model = FALSE` is such a fit, so one made inside a function whose frame is gone errors with class `propensity_ipw_data_error`, and the remedy is to supply `.data`.

One choice is refused for the standard error rather than for the model. A `"kernel"` density chooses its bandwidth from the residuals of the propensity score model, which is not a function of the parameters that the sandwich could differentiate, so weights built from it have no closed-form standard error and raise an error of class `propensity_ipw_se_method_unavailable_error`. The package offers no resampling method of its own, so the remedy the refusal names is a bootstrap of the whole pipeline written by hand: resample the rows, refit the propensity score model, rebuild the weights with `wt_ate()`, and refit the outcome model on each resample, then read the spread of the coefficient across the resamples. Doing it that way rather than inside `ipw()` keeps what each replicate re-estimates, such as a kernel bandwidth, under the writer's control. It is also how to carry a smoothing choice that the additive route conditions on: a bootstrap that refits the `mgcv::gam()` each time re-chooses the smoothing parameters along with everything else.

Standard errors by linearization:

Setting `se_method = "linearization"` instead uses the influence-function linearization of Kostouraki et al. (2024). It is available only for a binary exposure and only for the `ate`, `att`, `ato`, and `atm` estimands; a categorical or continuous exposure, or the `atu` or `entropy` estimand, requires `se_method = "mestimation"`. The linearization also supports only an outcome model whose formula contains the exposure alone. Its point estimates come from g-computation on the fitted outcome model, while its influence functions are those of the Hajek weighted-mean estimator; the two agree only for an exposure-only outcome model. A covariate-adjusted outcome model (any term beyond the exposure, including covariates, interactions, or transformed terms) errors on the linearization path and requires `se_method = "mestimation"`, which stacks adjusted outcome models correctly. The conditional reading is restricted the same way: linearization stacks no system, so the outcome model it stores carries no block of one, and the standard errors that reading reports are refused with a classed error rather than replaced by the covariance the outcome model computed for itself, which treats the estimated weights as fixed.

Weights stabilized on a fitted numerator model are refused here as well, with an error of class `propensity_ipw_se_method_unavailable_error`. The influence functions read the stabilizer as a known constant, which is exact for the default stabilizer and for a `stabilization_score` the caller fixed and is not exact for a model fit to the same data, so the method is refused rather than a standard error reported for a numerator nobody fit. `se_method = "mestimation"` estimates the numerator model alongside everything else. `se_method = "robust"` reads every weight as a known constant by design, that being what the diagnostic is, so it is not refused.

The linearization outcome model must also carry an intercept, which is a stricter requirement than the M-estimation path imposes. Under a numeric coding such as `y ~ z - 1` the linear predictor under no exposure is fixed at the link's zero point rather than estimated, so the g-computation marginal means no longer match the Hajek means the influence functions describe. The rejection is broader than that mechanism: a saturated factor coding such as `y ~ 0 + zf` does estimate both cell means, and the M-estimation path accepts it, but the linearization influence functions are derived for the intercept parameterization, so every no-intercept outcome model errors here. See **Model requirements** for the baseline contract both methods impose.

Standard errors as a diagnostic:

Setting `se_method = "robust"` reports the sandwich the weighted outcome model computes for itself, with the weights entering as known constants. It is the linearization route with the correction for having estimated the propensity score dropped, so the point estimates are the linearization ones and only the standard errors move. What it reports for the two counterfactual means and the contrasts built from them is the delta-method reading of `sandwich::vcovHC(outcome_mod, type = "HC0")`, exactly: the influence variance is read as a plain mean of squares rather than as a sample variance, so what comes back is the HC0 sandwich itself rather than a finite-sample rescaling of it.

This is offered as a diagnostic and not as an inference. It ignores that the weights were estimated, and the correction it drops is generally what a propensity score model contributes to the variance, so the standard errors it reports generally understate it, and understate it most where the weights are least stable. Use `"mestimation"` or `"linearization"` to report a standard error. Reach for the diagnostic to see what the propensity correction is worth on a given fit, by reading it beside one of them.

A result fit this way carries the class `"ipw_diagnostic_se"` ahead of `"ipw"`, and printing it writes one line naming the method, so the numbers are not read as accounting for the propensity score model. `as.data.frame()` and `tidy()` carry the same mark as an `ipw_se_diagnostic` attribute on the table. Every requirement of the linearization route applies here: a binary exposure, the `ate`, `att`, `ato`, and `atm` estimands, an outcome model of the exposure alone fit with an intercept, no `.by`, and no conditional reading. The categorical, continuous, and joint treatment routes refuse it as they refuse linearization.

The printed mark does not survive `pool_ipw()`: a pooled object is a summary of results rather than a result, and what says its standard errors are the diagnostic ones is the method it records, `se_method = "robust"`.

Multiple imputation

With missing data, fit the whole analysis once per imputed dataset and pool the results. Everything the analysis needs is rebuilt inside a single expression, so each imputation gets its own propensity model, its own weights, and its own outcome model:

```
imp <- mice::mice(dat, m = 20, print = FALSE)
```

```
fits <- with(imp, {
  ps <- glm(z ~ x1 + x2, family = binomial())
  w <- wt_ate(ps)
  om <- glm(y ~ z, family = quasibinomial(), weights = w)
  ipw(ps, om)
})
pool_ipw(fits)
```

`pool_ipw()` is the recommended verb. It reads the results themselves rather than a tidied table of them, so the effect labels survive, a categorical result keeps the contrast each row reports, and the complete-data degrees of freedom fall back to the outcome models whenever a result records none of its own. That fallback is written for the condition rather than for one route: a fit under `se_method = "linearization"` or `se_method = "robust"` records none, and so does a result another package built on the same class from estimating equations that report no residual degrees of freedom. It also takes the smallest degrees of freedom across the pooled results, which is the conservative choice when they differ. `tidy()` and `glance()` report what it returns.

`mice::pool()` also works, for every exposure type, and is the right choice when a result has to travel through the same pipeline as other analyses. Four things are worth knowing about that route. It reaches the result through `tidy()`, which pools a categorical result grouped by `term` and `contrast` together, a grouping mice has supported since 3.15.0. `mice::pool()` groups correctly but its `summary()` prints only `term`, so a categorical pooled table shows each effect measure repeated with no contrast label beside it, and the labels have to be read off `pooled$pooled` instead. A result that records no complete-data degrees of freedom of its own, a fit under `se_method = "linearization"` or `se_method = "robust"` among them, leaves the pooled degrees of freedom missing unless `dfcom` is passed explicitly, as in `mice::pool(fits, dfcom = df.residual(fits$analyses[[1]]$outcome_mod))`. Every result of that kind has to be told, where `pool_ipw()` reads the outcome models for all of them unasked. That remedy is what the package's requirement of mice 3.18.0 or later is for: mice 3.17.0 introduced a regression in the `dfcom` argument of `pool()`, and 3.18.0 is the version that repairs it. And the `exponentiate` argument of `summary()` on a pooled `mipo` is not the one these methods take: a `mipo` records no scale for the rows it holds, so it exponentiates every one of them, returning a risk difference as its exponential and still labeling the row `rd`. `tidy()` and the pooled result's own frame exponentiate the rows reported on the log scale alone and relabel those.

To pool the conditional reading, record it inside the same expression with `as_conditional()`, and the pooled result reports the outcome models' coefficients rather than the causal contrasts.

Recording the reading beforehand is one route to it rather than the only one. `pool_ipw()` pools both readings of the results it is given, storing the one they record and keeping the other beside it, so `as_marginal()` and `as_conditional()` move a pooled result between them after the pooling and report a table the pooling already built rather than pooling a second time. `tidy()`, `stats::coef()`, `stats::vcov()`, `stats::confint()`, and the pooled result's own frame each take an `effects` argument naming a reading for the one call, which leaves the pooled result as it is. `mice::pool()` offers neither: it reaches a result through `tidy()` and returns a `mipo` holding the one table it asked for, so on that route the reading still has to be recorded before the pooling.

A reading the pooling could not compute is refused rather than reported under the other one's name. The conditional reading needs the covariance the joint estimation of the weights and the outcome implies, and a fit under `se_method = "linearization"` or `se_method = "robust"` stacks no such system and records no such block, so a set of those fits pools the marginal reading alone. The pooled result records why the other reading is missing, and asking for it, whether by flip-

ping the result or by naming it for one call, errors with that recorded explanation under the classes `causalgenerics_pool_missing_surface_conditional` and `causalgenerics_pool_missing_surface`.

The analysis belongs inside `with()` rather than outside it because the propensity score model has to be estimated once per imputation. Weights built from propensity scores averaged across imputations are identical in every imputation, which leaves no between-imputation variance for the weights to contribute and no per-imputation estimation for the corrected standard errors to account for; both components of the variance are lost. Fitting within each imputation keeps the uncertainty of having estimated the weights inside each result, and the pooling adds the uncertainty the imputation itself contributed. Leyrat et al. (2019) compare the approaches and recommend this one.

Model requirements

`ipw()` cannot yet account for propensity scores that were trimmed (`ps_trim()`), truncated (`ps_trunc()`), or calibrated (`ps_calibrate()`) before weighting. The stacked estimating equations rebuild the weights from the propensity score model on every evaluation, so a weight that is no longer a deterministic function of that model breaks the sandwich variance. Supplying weights built from a modified score errors on either standard error method; refit the weights from the unmodified propensity score model. Density-trimmed weights for a continuous exposure (`ps_trim()` with `method = "density"` or `method = "resid"`) carry the trim record and are refused as trimmed weights.

Weights bounded with `wt_trunc()` are refused as well, with their own error, for a different reason. A bound on the weights is not a smooth function of the propensity score model's parameters, so the stacked sandwich would report an interval for the truncated estimand without saying so. The four methods also differ in kind. A fixed bound (`"wt"`, or `"adaptive"`, which depends only on the sample size) is known before the weights are seen, while a percentile bound (`"pctl"`, or `"count"`, which bounds at an observed weight) is estimated from the weights it modifies and would need an estimating equation of its own. Only the fixed case is a candidate for support, and relaxing the refusal for it first requires a simulation that measures the coverage of the M-estimation interval with the bound inside the stack. Until then, rebuild the weights without `wt_trunc()` and report the M-estimation interval, or fit the outcome model with the truncated weights and report a fixed-weight sandwich, stating that it conditions on the bound.

An outcome model fit without weights also errors on either method. The outcome model must not carry an offset term on either method, since neither the stacked outcome score nor the linearization influence functions thread an offset; supplying one errors.

For a binary or categorical exposure, the outcome model formula must contain the exposure. The counterfactual designs are built by setting the exposure to each level in turn, so a model fit on covariates alone gives one identical design per level; it errors on either standard error method.

For those two exposure types, the outcome model must also be able to represent a baseline at every level, for example through an intercept or through a saturated factor coding of the exposure. A numeric no-intercept coding such as `y ~ z - 1` errors on both standard error methods, for different reasons. On the M-estimation path the counterfactual design at the zero-coded level is identically zero, so the marginal mean there is fixed by the outcome link (`0.5` under a logit or probit link, `0` for a linear model) rather than estimated from the data. On the linearization and robust paths the intercept is required outright, because without it the g-computation means stop matching the Hajek means the influence functions describe. A saturated factor coding such as `y ~ 0 + zf` is a reparameterization whose designs are the level indicators, so the M-estimation path accepts it and reproduces the with-intercept fit; the linearization and robust paths still require the intercept. A no-intercept model that keeps a covariate, such as `y ~ z + x1 - 1`, still estimates the marginal mean from that covariate and

runs on the M-estimation path. A continuous exposure has no counterfactual designs, since `ipw()` reports the marginal structural model's own exposure coefficient, and is unaffected.

Three further requirements apply to the weights and the propensity score model. The outcome model weights must match the values implied by the propensity score model; a mismatch errors, on both standard error methods.

For a continuous exposure that requirement bears on the spread of the conditional density. `ipw()` stacks a single conditional spread alongside the propensity score coefficients, by the moment the pooled spread is the root of or, for a density built with `sigma_method = "mle"`, by the score of the family it names, so the weights it rebuilds are the ones `wt_ate()` produces when no `.sigma` is given. A single `.sigma` is taken instead as a known constant: the weights are rebuilt at the number that was supplied, and the stacked system carries none of that number's uncertainty, which is what fixing a spread says. Weights built with an observation-level `.sigma`, such as `influence(model)$sigma`, are a different function of the data with no counterpart in the stacked system, and are refused before anything is solved, with an error of class `propensity_ipw_sigma_error`. Rebuild the weights with the pooled default or with one number to use `ipw()`.

It bears on a `stabilization_score` in the same way. A score written per observation is one value per unit. Subsetting or reordering the weights with `[` carries it along, but a slice that cannot place it, such as `vctrs::vec_slice()` or `dplyr::arrange()`, drops it, and a model frame that drops incomplete rows leaves it at the length the weights were built at. Either way the record still names a score as the numerator and the score in hand no longer describes the observations being weighted, so `ipw()` refuses before anything is solved, with an error of class `propensity_ipw_stabilization_score_error`. Rebuild the weights on the rows being analyzed, or stabilize on a single `stabilization_score`, which scales every weight and survives any restriction.

Weights stabilized on a fitted numerator model carry a further requirement, on `outcome_mod` rather than on the weights. A numerator fit on a covariate divides every weight by a quantity that varies with it, so the pseudo-population the weights build is one in which that covariate still predicts the exposure. Confounding by it remains there, and what the weighted fit identifies is the effect within its levels rather than the marginal one, which only a marginal structural model carrying it reports (Robins et al. 2000; Cole and Hernán 2008). `ipw()` compares the two: the variables on the right-hand side of the numerator model's formula against the variables `outcome_mod`'s formula reads anywhere, and any the second does not read are reported once for the call, in a warning of class `propensity_ipw_stabilizer_coverage_warning` naming each of them. The comparison is of variables rather than of terms, so a model reading `factor(v)`, `poly(v, 2)`, or `splines::ns(v, 3)` reads `v`; a marginal structural model conditional on a transformation of a covariate is conditional on the covariate. The joint route asks the same question of each component's numerator and reports the answers together. The estimates are the ones the fit gives either way, so this is a warning and not a refusal: add the variables to `outcome_mod`, or refit the model given to `stabilize` without them. The default stabilizer, an intercept-only numerator model, and a `stabilization_score` all condition on nothing this can name and say nothing.

The propensity score model must not separate the exposure. A model whose covariates predict the exposure without error has no finite maximum likelihood estimate, and the propensity scores its coefficients imply reach exactly zero or one, leaving the corresponding weights undefined. For a binary exposure, whose propensity score model is a binomial `glm()`, both standard error methods reject such a fit at the same threshold: the fitted linear predictors, put through the link's inverse, saturate for at least one observation. Nothing short of saturation is rejected. A fitted model's `predict()` cannot show this on its own, because the inverse link `glm` uses is bounded away from zero and one, so a separated fit otherwise yields finite weights and a standard error that looks

ordinary. For a categorical exposure, which runs on the M-estimation path alone, the threshold is narrower: only a probability of exactly zero at the level a unit was actually assigned is rejected, since that alone leaves the unit's own weight undefined, and a softmax column for a level the unit was not assigned may reach zero without the fit being refused. A continuous exposure has no saturating inverse link and is not checked.

The weight functions apply a stricter rule to the propensity scores they are handed: a categorical matrix holding an exact 0 or 1 anywhere is refused, whichever level the cell belongs to. A separated `nnet::multinom()` fit is therefore stopped where its weights are built rather than here. `ps_trunc()` repairs such a matrix, its categorical matrix method reading the closed interval so that a cell at an endpoint is bounded rather than refused, while `ps_trim()` holds the open interval as the weight functions do. Weights built from a trimmed or truncated score are their own refusal here, as are weights bounded with `wt_trunc()`, since `ipw()` cannot yet account for a modified propensity score or a truncated weight, so a separated fit is repaired for the weight functions rather than for this one. See **Propensity scores at 0 and 1** in `wt_ate()` for the remedy.

The propensity score model must also be fit without case weights, since the stacked propensity score equations are unweighted and a weighted fit would not sit at the score root; that requirement applies to `se_method = "mestimation"` alone, because neither the linearization path nor the robust one restacks the propensity score model. Linearization still corrects for the uncertainty of estimating the propensity scores; it does so through the influence functions rather than through a stacked score. The robust diagnostic drops that correction, which is what makes it a diagnostic.

References

Stefanski LA, Boos DD. The calculus of M-estimation. *The American Statistician*. 2002;56(1):29–38. doi:10.1198/000313002753631330

Kostouraki A, Hajage D, Rachet B, et al. On variance estimation of the inverse probability-of-treatment weighting estimator: A tutorial for different types of propensity score weights. *Statistics in Medicine*. 2024;43(13):2672–2694. doi:10.1002/sim.10078

Lunceford JK, Davidian M. Stratification and weighting via the propensity score in estimation of causal treatment effects: a comparative study. *Statistics in Medicine*. 2004;23(19):2937–2960. doi:10.1002/sim.1903

Shu D, Young JG, Toh S, Wang R. Variance estimation in inverse probability weighted Cox models. *Biometrics*. 2020. doi:10.1111/biom.13332

Robins JM, Hernán MA, Brumback B. Marginal structural models and causal inference in epidemiology. *Epidemiology*. 2000;11(5):550–560. doi:10.1097/00001648200009000011

Cole SR, Hernán MA. Constructing inverse probability weights for marginal structural models. *American Journal of Epidemiology*. 2008;168(6):656–664. doi:10.1093/aje/kwn164

Hernán MA, Robins JM. *Causal Inference: What If*. Boca Raton: Chapman & Hall/CRC; 2020.

Leyrat C, Seaman SR, White IR, et al. Propensity score analysis with partially observed covariates: How should multiple imputation be used? *Statistical Methods in Medical Research*. 2019;28(1):3–19. doi:10.1177/0962280217713032

See Also

`wt_ate()`, `wt_att()`, `wt_atu()`, `wt_atm()`, `wt_ato()`, `wt_entropy()` for calculating propensity score weights.

ps_trim(), ps_trunc() for handling extreme propensity scores before weighting.

Examples

```
# Simulate data with a confounder, binary exposure, and binary outcome
set.seed(123)
n <- 200
x1 <- rnorm(n)
z <- rbinom(n, 1, plogis(0.5 * x1))
y <- rbinom(n, 1, plogis(-0.5 + 0.8 * z + 0.3 * x1))
dat <- data.frame(x1, z, y)

# Step 1: Fit a propensity score model
ps_mod <- glm(z ~ x1, data = dat, family = binomial())

# Step 2: Calculate ATE weights and fit a weighted outcome model
wts <- wt_ate(ps_mod)
outcome_mod <- glm(y ~ z, data = dat, family = quasibinomial(), weights = wts)

# Step 3: Estimate causal effects with correct standard errors
result <- ipw(ps_mod, outcome_mod)
result

# Exponentiate log-RR and log-OR to get RR and OR
as.data.frame(result, exponentiate = TRUE)

# Continuous outcome example
y_cont <- 2 + 0.8 * z + 0.3 * x1 + rnorm(n)
dat$y_cont <- y_cont
outcome_cont <- lm(y_cont ~ z, data = dat, weights = wts)
ipw(ps_mod, outcome_cont)

# Effect modification: the effect within each level of a factor, and the
# difference between them. The outcome model carries the exposure-by-modifier
# term that lets the effect differ across the strata.
dat$grp <- factor(rep(c("a", "b"), length.out = n))
ps_grp <- glm(z ~ x1 + grp, data = dat, family = binomial())
wts_grp <- wt_ate(ps_grp)
outcome_grp <- glm(
  y ~ z * grp + x1,
  data = dat,
  family = quasibinomial(),
  weights = wts_grp
)
ipw(ps_grp, outcome_grp, .by = grp)

# The same fit with the weights stabilized on the modifier. The numerator is
# the probability of the exposure each unit took given `grp`, which is the
# shape the default stabilizer has and conditions only on a variable the
# outcome model reads, so it tightens the weights without moving the estimand.
num_grp <- glm(z ~ grp, data = dat, family = binomial())
p_grp <- fitted(num_grp)
wts_stab <- wt_ate(
```

```
  ps_grp,
  stabilize = TRUE,
  stabilization_score = ifelse(dat$z == 1, p_grp, 1 - p_grp)
)
sd(wts_grp)
sd(wts_stab)
outcome_stab <- glm(
  y ~ z * grp + x1,
  data = dat,
  family = quasibinomial(),
  weights = wts_stab
)
ipw(ps_grp, outcome_stab, .by = grp)

# Continuous exposure: an lm propensity model of the dose on covariates,
# weights, which are stabilized by default for a continuous exposure, and a
# weighted marginal structural outcome model
a <- 0.5 + 0.8 * x1 + rnorm(n)
y_dose <- 1 + 0.6 * a + 0.3 * x1 + rnorm(n)
dat$a <- a
dat$y_dose <- y_dose
ps_cont <- lm(a ~ x1, data = dat)
wts_cont <- wt_ate(ps_cont)
msm <- lm(y_dose ~ a, data = dat, weights = wts_cont)
ipw(ps_cont, msm)

# A dose-response curve written as a basis matrix reports one row per basis
# coefficient. The model frame of such a fit holds the basis rather than the
# dose, so `.data` supplies the exposure.
msm_curve <- lm(y_dose ~ poly(a, 2), data = dat, weights = wts_cont)
ipw(ps_cont, msm_curve, .data = dat)


# Categorical exposure: a multinomial propensity model and per-level contrasts
set.seed(2)
n <- 300
x1 <- rnorm(n)
a <- factor(sample(c("a", "b", "c"), n, replace = TRUE))
y <- rbinom(n, 1, plogis(-0.5 + 0.4 * (a == "b") + 0.8 * (a == "c") + 0.3 * x1))
dat_cat <- data.frame(x1, a, y)

ps_cat <- nnet::multinom(a ~ x1, data = dat_cat, trace = FALSE)
ps_mat <- predict(ps_cat, type = "probs")
wts_cat <- wt_ate(ps_mat, dat_cat$a, exposure_type = "categorical")
outcome_cat <- glm(y ~ a, data = dat_cat, family = quasibinomial(), weights = wts_cat)
ipw(ps_cat, outcome_cat)
```

---

is_ps_calibrated *Check if propensity scores are calibrated*

Description

`is_ps_calibrated()` tests whether `x` was calibrated, rather than whether it is still a `ps_calib` object: it is `TRUE` for a calibrated propensity score, for weights derived from one, and for scores that were calibrated before `ps_trim()` or `ps_trunc()` bounded them, each of which records the calibration in its own metadata.

Usage

```
is_ps_calibrated(x)
```

Arguments

`x`	An object to test.

Value

A single `TRUE` or `FALSE`.

See Also

`ps_calibrate()` to calibrate propensity scores.

Examples

```
ps <- runif(100)
exposure <- rbinom(100, 1, ps)

is_ps_calibrated(ps)

calibrated <- ps_calibrate(ps, exposure, smooth = FALSE)
is_ps_calibrated(calibrated)
```

`is_ps_trimmed` *Test whether propensity scores have been trimmed*

Description

`is_ps_trimmed()` returns `TRUE` if `x` is a `ps_trim` object or a `psw` object created from trimmed propensity scores, and `FALSE` otherwise. This tests whether the *object* carries trimming information, not which individual units were trimmed; see `is_unit_trimmed()` for that.

Usage

```
is_ps_trimmed(x)
```

Arguments

`x`	An object to test.

Value

A logical scalar (TRUE or FALSE).

See Also

ps_trim() for trimming propensity scores, is_unit_trimmed() to identify which units were trimmed, ps_trim_meta() to retrieve full trimming metadata.

Examples

```
ps <- c(0.05, 0.3, 0.6, 0.95)
trimmed <- ps_trim(ps, method = "ps", lower = 0.1, upper = 0.9)

is_ps_trimmed(trimmed)
is_ps_trimmed(ps)
```

---

is_ps_truncated *Test whether propensity scores have been truncated*

Description

is_ps_truncated() returns TRUE if x is a ps_trunc object or a psw object derived from truncated propensity scores. Use is_unit_truncated() to find out *which* observations were modified.

Usage

```
is_ps_truncated(x)
```

Arguments

x	An object.

Value

A single TRUE or FALSE.

See Also

ps_trunc(), is_unit_truncated(), ps_trunc_meta()

Examples

```
ps <- c(0.02, 0.3, 0.5, 0.7, 0.98)
is_ps_truncated(ps)

ps_t <- ps_trunc(ps, method = "ps", lower = 0.05, upper = 0.95)
is_ps_truncated(ps_t)
```

`is_refit` *Check if propensity scores have been refit*

Description

`is_refit()` tests whether `x` is a `ps_trim` object whose propensity model has been refit on the retained (non-trimmed) observations via `ps_refit()`.

Usage

```
is_refit(x)
```

Arguments

`x`	An object to test (typically a ps_trim vector).

Details

On a psw vector built from trimmed propensity scores, the answer comes from the trimming record carried on the weights. `is_refit()` reads a single flag out of that record rather than a position, so unlike `is_unit_trimmed()` it answers from any record present, whatever length the weights have since taken on. It raises an error of class `propensity_missing_meta_error` only when weights marked as trimmed carry no record at all, which is what a subset of such weights leaves behind. Weights that were never trimmed have nothing missing and return `FALSE`.

Value

A single `TRUE` or `FALSE`.

See Also

`ps_refit()` to refit a propensity model after trimming, `ps_trim()` to trim propensity scores.

Examples

```
set.seed(2)
n <- 30
x <- rnorm(n)
z <- rbinom(n, 1, plogis(0.4 * x))
fit <- glm(z ~ x, family = binomial)
ps <- predict(fit, type = "response")

trimmed <- ps_trim(ps, lower = 0.2, upper = 0.8)
is_refit(trimmed)

refit <- ps_refit(trimmed, fit)
is_refit(refit)
```

`is_unit_trimmed` *Identify which units were trimmed*

Description

`is_unit_trimmed()` returns a logical vector indicating which observations were removed by trimming. This is a per-unit query, as opposed to `is_ps_trimmed()`, which tests whether the object has been trimmed at all.

The answer comes from the trimming record, which is written for a fixed set of observations and can both be lost and outlive them. On a `ps_trim`, `vctrs::vec_slice()` and `c()` drop it, and subassignment that grows the vector carries it across a length change; see `ps_trim()` for the whole contract. On a psw vector built from trimmed propensity scores, a subset drops it, while `model.frame()` re-attaches it to the shortened weights column of an outcome model fit on these weights.

`is_unit_trimmed()` therefore checks that the record covers the object it is given, and raises an error of class `propensity_missing_meta_error` when it does not, or when an object marked as trimmed carries no record at all, rather than name trimmed units at stale positions. Query the `ps_trim` object the record was written for instead.

That check counts observations, which a reordering does not change, so a `ps_trim` or `psw` reordered through vctrs rather than through `[`, by `vctrs::vec_slice(x, 5:1)` or `dplyr::arrange()`, drops the positions instead, and `is_unit_trimmed()` refuses the result. See `ps_trim()` and psw for the whole contract.

Usage

```
is_unit_trimmed(x)
```

Arguments

`x`	A `ps_trim` object created by `ps_trim()`, or a psw vector built from one.

Value

A logical vector the same length as `x`, where `TRUE` marks a trimmed unit.

See Also

`ps_trim()` for trimming propensity scores, `is_ps_trimmed()` to test whether an object has been trimmed, `ps_trim_meta()` to retrieve full trimming metadata.

Examples

```
ps <- c(0.05, 0.3, 0.6, 0.95)
trimmed <- ps_trim(ps, method = "ps", lower = 0.1, upper = 0.9)

is_unit_trimmed(trimmed)

# Use to subset data to retained observations
```

```
kept <- !is_unit_trimmed(trimmed)
ps[kept]
```

`is_unit_truncated` *Identify which units were truncated*

Description

`is_unit_truncated()` returns a logical vector indicating which observations had their propensity scores modified by truncation. Use `is_ps_truncated()` to test whether an object has been truncated at all.

The answer comes from the truncation record, which is written for a fixed set of observations and can both be lost and outlive them. On a `ps_trunc`, `vctrs::vec_slice()` and `c()` drop it, and subassignment that grows the vector carries it across a length change; see `ps_trunc()` for the whole contract. On a psw vector built from truncated propensity scores, a subset drops it, while subassignment that grows the weights carries it across the length change.

`is_unit_truncated()` therefore checks that the record covers the object it is given, and raises an error of class `propensity_missing_meta_error` when it does not, or when an object marked as truncated carries no record at all, rather than name truncated units at stale positions. Query the `ps_trunc` object the record was written for instead.

That check counts observations, which a reordering does not change, so a `ps_trunc` or psw reordered through vctrs rather than through `[`, such as by `vctrs::vec_slice(x, 5:1)` or `dplyr::arrange()`, drops the positions instead, and `is_unit_truncated()` refuses the result. See `ps_trunc()` and psw for the whole contract.

Usage

```
is_unit_truncated(x)
```

Arguments

x	A `ps_trunc` object created by `ps_trunc()`, or a psw vector built from one.

Value

A logical vector the same length as `x` (or number of rows for matrix input). `TRUE` marks observations whose values were winsorized.

See Also

`ps_trunc()`, `is_ps_truncated()`, `ps_trunc_meta()`

Examples

```
ps <- c(0.02, 0.3, 0.5, 0.7, 0.98)
ps_t <- ps_trunc(ps, method = "ps", lower = 0.05, upper = 0.95)
is_unit_truncated(ps_t)
```

`is_wt_truncated` *Test whether the weights themselves have been truncated*

Description

`is_wt_truncated()` returns `TRUE` for a psw vector whose own values were bounded, and `is_unit_wt_truncated()` returns which of its weights the bound moved.

Truncating the weights is recorded apart from truncating the propensity scores they were built from, which `is_ps_truncated()` and `is_unit_truncated()` report. The two are different operations, and a set of weights can carry either, both, or neither.

Usage

```
is_wt_truncated(x)

is_unit_wt_truncated(x)
```

Arguments

`x`	An object. `is_unit_wt_truncated()` accepts only a psw vector.

Details

The truncation leaves a record on the weights, the `psw_trunc_meta` attribute, holding the method, the bound as given and as applied, and the positions of the weights it moved. `is_unit_wt_truncated()` answers from those positions, so the record follows the rules psw describes for the records a modified propensity score leaves: it is kept through arithmetic and subassignment, re-indexed through the subscript of `[`, and loses its positions to any other slice and to a combine, which keeps the method and bounds. The `wt_truncated` flag describes the weights as a whole and is kept through all of those, so `is_wt_truncated()` keeps its answer where `is_unit_wt_truncated()` has none to give.

`is_unit_wt_truncated()` therefore checks that the record covers the weights it is given, and raises an error of class `propensity_missing_meta_error` when it does not, or when weights marked as truncated carry no record at all, rather than name truncated weights at stale positions. Weights that were not truncated have no record to read, and every weight is reported as untouched.

Value

- `is_wt_truncated()`: a single `TRUE` or `FALSE`. It is `FALSE` for anything that is not a psw vector.
- `is_unit_wt_truncated()`: a logical vector the same length as `x`, `TRUE` for each weight the bound moved.

See Also

psw for the weight vector class, and `is_ps_truncated()` for truncation of the propensity scores.

Examples

```
w <- psw(c(1.2, 0.8, 1.5), estimand = "ate")
is_wt_truncated(w)
is_unit_wt_truncated(w)
```

`joint_wt_models` *Record the two treatment models of a joint exposure*

Description

`joint_wt_models()` records the two fitted treatment models a joint weight is built from, in the order the joint density factorizes: the first treatment's model, then the second treatment's model, which conditions on the first. `is_joint_wt_models()` reports whether an object is such a record.

The container exists to refuse. It checks that the pair describes a sequential factorization at all, which is the one thing about a joint weight that cannot be detected from the weight vector afterwards.

Usage

```
joint_wt_models(...)

is_joint_wt_models(x)
```

Arguments

`...`	Exactly two fitted treatment models, each named for the treatment it fits. The order is the factorization's order. Supported models are a binomial `stats::glm()` for a binary treatment, a `nnet::multinom()` for a categorical one, and an `stats::lm()`, a gaussian `stats::glm()`, a gaussian `mgcv::gam()`, or a `MASS::rlm()` for a continuous one. Each class is recognized by name rather than by inheritance, so a subclass of one of them is not read as its parent.
`x`	An object to test.

Value

`joint_wt_models()` returns an S3 object of class `joint_wt_models`, a list with three fields:

`names` The two treatment names, in the factorization's order.

`models` The two fitted models, named for their treatments.

`exposure_type` Each model's exposure type, named for its treatment: `"binary"`, `"categorical"`, or `"continuous"`. The type follows the model's class, so a `nnet::multinom()` is recorded as `"categorical"` whatever its level count. One fit to two levels is a binary treatment model, and `ipw()` stacks it as one.

`is_joint_wt_models()` returns a single logical.

What is checked

Each argument is named for the treatment its model fits, and the name has to be that model's response: the factorization check looks for the first treatment by name in the second model's formula, so a model recorded under the wrong name would be checked for the wrong variable.

The second model must condition on the first treatment. The product of two marginal models, $f(A|L)f(E|L)$, is not the joint weight $f(A|L)f(E|A,L)$ for any data in which the second treatment depends on the first, and nothing downstream can tell the two apart. What is checked is whether the second model's right-hand side *mentions* the first treatment, so `e ~ factor(a) * x1` and `e ~ x1 + a:x1` both pass, as does the additive `e ~ a + x1`.

Passing is not the same as modeling the dependence well. An additive term satisfies the factorization and may still model it badly, and a badly modeled dependence biases the joint weight invisibly. Model the dependence on the first treatment flexibly, with an interaction or a spline, unless you have a reason to believe it is additive.

Neither may the two models condition on each other. A sequential factorization has a first factor that is marginal in the second treatment, so a pair that each read the other's treatment are not the two factors of any factorization, in either order.

See Also

`wt_joint()` for the product weight itself. These are used by `ipw()` methods for joint treatment models.

Examples

```
set.seed(4)
n <- 200
x1 <- rnorm(n)
a <- rbinom(n, 1, plogis(0.3 * x1))
e <- rbinom(n, 1, plogis(-0.2 + 0.5 * x1 - 0.8 * a))
dat <- data.frame(x1, a, e)

models <- joint_wt_models(
  a = glm(a ~ x1, data = dat, family = binomial()),
  e = glm(e ~ a * x1, data = dat, family = binomial())
)
models$names
models$exposure_type
is_joint_wt_models(models)
```

`numerator_model` *The model a set of weights was stabilized on*

Description

`numerator_model()` returns the fitted model supplied to `stabilize` that estimated the numerator of a set of weights, and `NULL` for weights whose numerator was not estimated by one.

Usage

```
numerator_model(wt)
```

Arguments

wt	A psw or causal_wts object.

Details

A model supplied to wt_ate()'s stabilize argument estimates the numerator of the weights, and what it estimates follows the exposure. A binomial() fit of a binary exposure reports the probability of the level each unit took; an nnet::multinom() fit of a categorical exposure reports one probability per level, and the numerator is the column named for the level each unit took; a model of a dose reports the conditional mean the density family is read at, and the numerator is that density. The model itself is recorded rather than the numerator it evaluates to, because ipw() rebuilds that numerator at every value of its parameter vector, which takes the model's design and its coefficients.

Where the record is kept differs by exposure type, and this accessor is the one place to read it either way. Weights for a continuous exposure are a ratio of densities and keep the model inside the density record density_meta() returns; weights for a binary or categorical exposure are a ratio of probabilities, leave no density record, and keep the model on its own. Both come back here.

The record describes the numerator rather than the units, so it holds no length of its own and survives subsetting, arithmetic, and anything else that changes the number of weights. Two sets of weights record the same numerator when their models write the same formula and were fit to the same coefficients; combining two that record different ones drops the record, with a warning of class propensity_metadata_conflict_warning. Weights stabilized without a model were divided by a numerator no model estimated, which is a different numerator again: combining those with weights stabilized on a model drops the model the same way, and the result is stabilized on nothing it can name.

A stabilization_score is a numerator the caller computed rather than one a model estimated, so weights stabilized on a score record no model, and stabilization_score() returns NULL for weights stabilized on a model.

Value

The fitted model supplied to stabilize, or NULL.

See Also

wt_ate() for the stabilize argument that writes this record, stabilization_score() for the numerator a caller supplies as a vector, and density_meta() for the rest of what a continuous exposure's weights record.

Examples

```
set.seed(1)
x <- rnorm(50)
v <- rbinom(50, 1, 0.5)
```

```
z <- rbinom(50, 1, plogis(0.4 * x - 0.7 * v))

ps_mod <- glm(z ~ x + v, family = binomial())
num_mod <- glm(z ~ v, family = binomial())

w <- wt_ate(ps_mod, stabilize = num_mod)
numerator_model(w)

# Weights stabilized on the marginal probability record no model.
numerator_model(wt_ate(ps_mod, stabilize = TRUE))
```

`print.ipw_diagnostic_se` *Print a result whose standard errors are a diagnostic*

Description

`se_method = "robust"` reports the sandwich the weighted outcome model computes for itself, which treats the estimated weights as known. Printing such a result writes what print() writes for any `ipw()` result and then one line naming the method, so that the number beside each estimate is not read as one accounting for the propensity score model. See **Standard errors as a diagnostic** in `ipw()` for what the diagnostic does and does not cover.

Usage

```
## S3 method for class 'ipw_diagnostic_se'
print(x, ...)

## S3 method for class 'ipw_diagnostic_se'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

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

Arguments

`x`	An `ipw` result fit with `se_method = "robust"`.
`...`	Passed to the next method.
`row.names, optional`	Passed to the next method.

Value

`x`, invisibly.

`as.data.frame()` returns what it returns for any `ipw()` result, carrying an `ipw_se_diagnostic` attribute naming the method.

`tidy()` returns what it returns for any `ipw()` result, carrying an `ipw_se_diagnostic` attribute naming the method.

See Also

ipw() for the estimator and the standard error methods.

Examples

```
set.seed(2)
n <- 300
x <- rnorm(n)
z <- rbinom(n, 1, plogis(0.4 * x))
y <- rbinom(n, 1, plogis(-0.3 + 0.7 * z + 0.5 * x))
dat <- data.frame(x, z, y)

ps_mod <- glm(z ~ x, data = dat, family = binomial())
wts <- wt_ate(ps_mod)
outcome_mod <- glm(y ~ z, data = dat, family = quasibinomial(), weights = wts)

print(ipw(ps_mod, outcome_mod, se_method = "robust"))
```

---

ps_calibrate *Calibrate propensity scores*

Description

ps_calibrate() adjusts estimated propensity scores so they better reflect true treatment probabilities. This can improve the accuracy of inverse probability weights derived from a misspecified propensity score model.

Usage

```
ps_calibrate(
  .propensity,
  .exposure,
  method = c("logistic", "isoreg"),
  smooth = TRUE,
  .focal_level = NULL,
  .reference_level = NULL,
  ...,
  .treated = NULL,
  .untreated = NULL,
  ps = lifecycle::deprecated()
)
```

Arguments

.propensity	A numeric vector of propensity scores in [0, 1]. Unlike the rest of the package, calibration accepts scores of exactly 0 and exactly 1, since repairing scores a model pushed to the ends of the interval is part of what calibration is for. The

logistic calibration curve maps them back inside the interval; isotonic calibration can return a score at an endpoint when its pooled block is pure, and such scores are rejected by the weight functions. Must not already be calibrated.

A data frame of predicted probabilities, one column per exposure level, is reduced to a single column the way `ps_trim()` and `ps_trunc()` reduce one: the second column of a pair, or the only column of a frame that has one. The column read is announced. A frame holding a `.pred_class` column, which a fitted tidymodels classification model returns when no prediction type is named, carries predicted levels rather than probabilities and is refused with an error of class `propensity_df_class_column_error`.

`.exposure` A binary vector of observed treatment assignments, the same length as `.propensity`.

`method` Calibration method. One of:

`"logistic"` (Default) Logistic calibration, also called Platt scaling. Fits a logistic regression of `.exposure` on `.propensity`, yielding a smooth, parametric correction. Works well with small samples and when the bias in `.propensity` is approximately monotone.

`"isoreg"` Isotonic regression. Fits a non-parametric, monotone step function. More flexible than logistic calibration because it makes no distributional assumption, but needs larger samples for stable estimates.

`smooth` Logical. When `method = "logistic"`, controls the form of the calibration model. If `TRUE` (default), fits a GAM with a spline on `.propensity` via `mgcv::gam()`; if `FALSE`, fits a simple logistic regression via `stats::glm()`. Ignored when `method = "isoreg"`. A spline needs enough distinct scores to place its knots, so with fewer than 10 distinct values of `.propensity` among the observations the fit reads, those with both a score and an exposure recorded, the fit falls back to logistic regression without a spline. The fallback is announced and recorded in the returned metadata.

`.focal_level` The value of `.exposure` representing the focal group (typically the treated group). Every binary coding honors it: 0/1 numeric, logical, two-level factor, and two-level character exposures are all coded with the named level as focal, and a level the exposure never takes is an error. With no level named, the exposure defaults to its higher level, which is 1 for a 0/1 exposure, `TRUE` for a logical one, and the second of the two levels a factor or character exposure takes. Levels a factor declares but never takes are not candidates. Naming any other level reverses the coding, so `.propensity` must then hold the probability of the named level.

`.reference_level`

The value of `.exposure` representing the reference group (typically the control group). Naming it makes the exposure's other level focal, with the same consequence for `.propensity`, and a level the exposure never takes is an error. Automatically detected if not supplied.

`...` Additional arguments passed to methods.

`.treated` **[Deprecated]** Use `.focal_level` instead.

`.untreated` **[Deprecated]** Use `.reference_level` instead.

`ps` **[Deprecated]** Use `.propensity` instead. A call that names `ps` must name the arguments after it as well, since a positional argument binds to `.propensity`.

Details

Calibration is useful when the propensity score model is correctly specified in terms of variable selection but produces probabilities that are systematically too high or too low. Unlike `ps_trim()` and `ps_trunc()`, which handle extreme scores by removing or bounding them, calibration reshapes the entire distribution of scores.

Choosing a method:

- Use `"logistic"` (the default) as a first choice. It is stable and fast, and the `smooth = TRUE` option adds flexibility via a spline.
- Use `"isoreg"` when you suspect a non-smooth or irregular relationship between estimated and true probabilities and have a sufficiently large sample.

Fitted models: `.propensity` can be the fitted propensity score model instead of the scores it reports, and the exposure is then read off the model as well unless `.exposure` is supplied. The variable read is announced; `options(propensity.quiet = TRUE)` silences the announcement. A binomial `stats::glm()` and a two-level `nnet::multinom()` report the one score per unit calibration reads. A `nnet::multinom()` of three or more levels reports one column per level and no single score to calibrate, so it is refused with an error of class `propensity_model_family_error`; calibrate the columns one at a time against the exposure indicator each of them belongs to.

Missing values: A unit with a missing exposure tells the calibration model nothing, so it is dropped from the fit. What happens to that unit afterwards depends on the method. Logistic calibration, smoothed or not, fits a curve from propensity score to calibrated score and reads every unit off it, so a unit with a missing exposure but an observed score is still calibrated. Isotonic calibration fits separately within each exposure group and reads each unit from the fit for the group it belongs to; a unit with a missing exposure belongs to neither group and is returned as `NA`. Under both methods a missing propensity score yields a missing calibrated score.

The calibrated scores are returned as a `ps_calib` object, which can be passed directly to weight functions such as `wt_ate()`.

Value

A `ps_calib` vector the same length as `.propensity`. The attribute `ps_calib_meta` stores calibration metadata (method and whether smoothing was applied). Use `is_ps_calibrated()` to test whether an object has been calibrated.

References

Platt, J. (1999). Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in Large Margin Classifiers*, 61–74.

Zadrozny, B., & Elkan, C. (2002). Transforming classifier scores into accurate multiclass probability estimates. *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 694–699. doi:10.1145/775047.775151

See Also

`is_ps_calibrated()` to test for calibrated scores; `ps_trim()` and `ps_trunc()` for alternative approaches to extreme propensity scores; `wt_ate()` and other weight functions that accept `ps_calib` objects.

Examples

```
# Simulate data
set.seed(42)
ps <- runif(200)
exposure <- rbinom(200, 1, ps)

# Logistic calibration without smoothing (simple Platt scaling)
cal <- ps_calibrate(ps, exposure, smooth = FALSE)
cal

# Use calibrated scores to calculate weights
wt_ate(cal, exposure)

# Isotonic regression calibration
cal_iso <- ps_calibrate(ps, exposure, method = "isoreg")

if (rlang::is_installed("mgcv")) {
  # Logistic calibration with spline smoothing (default)
  cal_smooth <- ps_calibrate(ps, exposure)
}

# Calibrate the scores a fitted model reports, reading the exposure off it
x <- rnorm(200)
fit <- glm(exposure ~ x, family = binomial())
ps_calibrate(fit, smooth = FALSE)

if (rlang::is_installed("nnet")) {
  exposure_factor <- factor(exposure)
  multinomial_fit <- nnet::multinom(exposure_factor ~ x, trace = FALSE)
  ps_calibrate(multinomial_fit, smooth = FALSE)
}
```

---

ps_refit *Refit a Propensity Score Model on Retained Observations*

Description

Re-estimates a propensity score model using only the observations retained after trimming. This is the recommended intermediate step between `ps_trim()` and weight calculation (e.g. `wt_ate()`):

`ps_trim()` **->** `ps_refit()` **->** `wt_*()`

Trimming changes the target population by removing observations with extreme propensity scores. Refitting the model on the retained subset produces propensity scores that better reflect this population, improving both model fit and downstream weight estimation. Weight functions warn if a trimmed propensity score has not been refit.

Usage

```
ps_refit(trimmed_ps, model, .data = NULL, ...)
```

Arguments

trimmed_ps	A ps_trim object returned by ps_trim(). Refitting reads the retained positions out of the trimming record, so an object whose record was dropped or no longer covers it raises an error of class propensity_missing_meta_error; see ps_trim(). A single column of a data frame made from a trimmed score matrix raises an error of class propensity_method_error; refit the matrix before converting it. Truncated scores from ps_trunc() keep every unit, so refitting them would return the original model's scores, and they raise an error of class propensity_method_error.
model	The original fitted model used to estimate the propensity scores (e.g. a glm or multinom object). The model is refit via update() on the retained subset. Trimmed propensity scores are refit only by a model of the probability of the exposure; a model of a conditional mean, such as a stats::lm() fit or a gaussian stats::glm(), never produced them and raises an error of class propensity_model_family_error. A dose model trimmed with method = "density" or method = "resid" is refit only by a model of the dose's conditional mean, such as a gaussian stats::glm(), stats::lm(), mgcv::gam(), or MASS::rlm() fit, and any other model, such as one of a probability, raises the same error.
.data	A data frame with one row per observation in trimmed_ps, in the same order. If NULL (the default), the data are recovered from model: its model.frame() when that already holds every variable the refit reads, and otherwise the data the model names, restricted by row name to the rows the model analyzed. A model fit without a data argument names none, and its variables are read out of the formula's environment instead. A formula that transforms a term, such as z ~ log(x) or a spline basis, stores that term already computed, so only the underlying variables let the transformation be recomputed from the retained rows. Pass .data when the data the model was fit on can no longer be reached.
...	Additional arguments passed to update(), such as subset or weights. Each is evaluated against the retained rows, the data the refit is handed: a name is read from their columns first and otherwise from the environment ps_refit() was called from, so subset = x1 > 0 refits on the retained rows where x1 is positive. A logical or index vector supplied instead indexes the retained rows, not the full data. The .data and .env pronouns are available to tell a column from a variable of the same name. Without .data, the retained rows hold only the variables model reads, so an argument naming any other column needs the data frame passed to .data.

Details

Composing with a subset:

A subset in the original call has already chosen the sample the propensity scores are about, and the trimming record indexes that sample rather than every row the data carry. Refitting narrows that sample further, to the retained rows, so the original subset is dropped from the call rather than put to work a second time on rows it was never about. A subset passed through ... is an instruction of its own and is honored.

Which level a refit predicts:

For a vector of scores, the refit predicts the probability of the level the scores describe: when ps_trim() was given a model with its first level named as focal, the refit reports one minus the model's prediction, and for scores supplied directly it reports the level the model predicts by default.

Arguments read from outside the formula:

`weights`, `offset`, and `na.action` in the original call are re-evaluated against the retained rows. A `weights` or `offset` naming a column of the data the model was fit on is read from that column and follows the retained rows, whether the data are recovered from `model` or passed to `.data`. A vector held outside the data cannot follow them: it keeps the length it had and raises an error about differing variable lengths.

Scores predicted from a fit with `na.action = na.exclude` are padded back to the full length of the data, so they describe more observations than the fit read and the trimming record indexes a sample the model never analyzed. `ps_refit()` refuses such scores. Trim scores from a fit whose `na.action` drops those rows instead.

Value

A `ps_trim` object with re-estimated propensity scores for retained observations and `NA` for trimmed observations. Use is_refit() to confirm refitting was applied. Refitting replaces the values a calibrated score held with predictions from the refit model, so a score trimmed after ps_calibrate() is no longer calibrated and is_ps_calibrated() answers `FALSE` for the result.

For a trimmed dose model, the retained values are the refit model's conditional means, and the spread in the record is re-estimated from the retained residuals under the recorded density family, unless the trim was made at a `.sigma` the caller supplied, which is kept. The recorded threshold describes the cut that was made and is left as it was, so the refit model can place a retained unit's conditional density below it; nothing is trimmed again.

See Also

ps_trim() for the trimming step, is_refit() to check refit status, wt_ate() and other weight functions for the next step in the pipeline.

Examples

```
set.seed(2)
n <- 200
x <- rnorm(n)
z <- rbinom(n, 1, plogis(0.4 * x))

# fit a propensity score model
ps_model <- glm(z ~ x, family = binomial)
ps <- predict(ps_model, type = "response")

# trim -> refit -> weight pipeline
trimmed <- ps_trim(ps, lower = 0.1, upper = 0.9)
refit <- ps_refit(trimmed, ps_model)
wts <- wt_ate(refit, .exposure = z)

is_refit(refit)
```

`ps_tilt` *Propensity score tilting functions*

Description

Every estimand this package supports targets a population whose covariate distribution is the study population reweighted by a tilting function h of the propensity score. `ps_tilt()` evaluates h at each observation's propensity score.

The tilt is the numerator of every weight: a weight is h divided by the propensity score of the exposure level the unit actually received. It is also what standardizes a g-computation estimate to a target population, which for `"atm"`, `"ato"`, and `"entropy"` is the only route to the estimand, since those populations are not subsets of the rows and cannot be reached by filtering.

Usage

```
ps_tilt(
  .propensity,
  estimand,
  ...,
  .focal_level = NULL,
  ps = lifecycle::deprecated()
)
```

Arguments

`.propensity`	Propensity scores. A numeric vector of $P(Z = \text{focal} \mid X)$ for a binary exposure, or a matrix or data frame with one column per exposure level, named for that level, for a categorical exposure. A data frame holding a `.pred_class` column, which a fitted tidymodels classification model returns when no prediction type is named, carries predicted levels rather than probabilities and is refused with an error of class `propensity_df_class_column_error`.
`estimand`	One of `"ate"`, `"att"`, `"atu"`, `"atm"`, `"ato"`, or `"entropy"`.
`...`	These dots are for future extensions and must be empty.
`.focal_level`	The exposure level the `"att"` and `"atu"` tilts target, matched against the column names of `.propensity`. A column named for the level exactly wins; failing that, a `.pred_` prefix is stripped, so `.pred_a` matches the level `a`. Required for those two estimands with a categorical `.propensity`, and accepted nowhere else: a numeric `.propensity` is already the probability of the focal level, and the remaining tilts treat every level alike.
`ps`	**[Deprecated]** Use `.propensity` instead. A call that names `ps` must name the arguments after it as well, since a positional argument binds to `.propensity`.

Value

A plain double vector, unnamed, with one element per observation: the length of `.propensity` for the numeric method, and the number of rows of `.propensity` for the matrix and data frame methods.

Tilting functions

For a binary exposure with propensity score $e = P(Z = \text{focal} \mid X)$:

estimand	$h(e)$	target population
`"ate"`	1	everyone
`"att"`	e	the focal group
`"atu"`	$1 - e$	the reference group
`"atm"`	$\min(e, 1 - e)$	the evenly matchable
`"ato"`	$e(1 - e)$	the overlap population
`"entropy"`	$-e \log(e) - (1 - e) \log(1 - e)$	the entropy-tilted population

For a categorical exposure with propensity score vector $(e_1, \ldots, e_K)$ and focal level f: `"ate"` is 1, `"att"` is e_f, `"atu"` is $1 - e_f$, `"atm"` is $\min_k e_k$, `"ato"` is $(\sum_k 1/e_k)^{-1}$, and `"entropy"` is $-\sum_k e_k \log(e_k)$.

Stabilization and censoring weights are not tilts. Stabilization multiplies a weight by a marginal quantity that does not depend on the covariates, and `wt_cens()` reuses the `"ate"` formula, so neither has an entry here.

Standardizing model predictions

A g-computation estimate averages an outcome model's per-row counterfactual predictions, and where those predictions vary from row to row it is the weights of that average that standardize the estimate to a target population. The tilt is those weights, which is what the second example below supplies by hand. An average taken with equal weight standardizes a covariate-adjusted outcome model to the whole sample, so such a model targets the ATE however its own fitting weights were built: fit for a non-`"ate"` estimand and averaged that way, it reports the full-sample contrast rather than the one it was weighted for. Tools that average per-row model predictions, such as the `avg_comparisons()` function in the marginaleffects package, average with equal weight unless they are told otherwise, and the remedy is to hand them the tilt as the weights of the average: `ps_tilt(ps, "att")` for a binary exposure and `ps_tilt(ps, "att", .focal_level = "b")` for a categorical one.

The requirement is easy to miss because an outcome model saturated in the exposure hides it. Such a model predicts one value per exposure level, so every average of its predictions returns the same contrast, and there the estimand is settled by the weights the model was fit with rather than by the weights of the average: a model fit with `wt_att()` weights reports the ATT whether the average is taken with equal weight or with the tilt. The two averages come apart at the first covariate the outcome model adjusts for, where the predictions vary from row to row and the weights the average is taken under are what decides the population the estimate describes.

Propensity score range

Every propensity score in `.propensity` must lie strictly inside $(0, 1)$, the same requirement `wt_ate()` and the rest of the weight family impose before they tilt. The bound holds for a matrix or data frame `.propensity` entry by entry, and each row must sum to one on top of it. A fitted model that separates the exposure can return a probability of exactly zero or one; those scores have no weight to divide and are rejected here rather than tilted.

A missing propensity score gives a missing tilt under every estimand, `"ate"` included, so an observation whose propensity score is unknown never counts toward a tilted mean. A numeric `.propensity` propagates `NA` position by position, and a matrix `.propensity` gives `NA` for any row holding one: a probability vector with a missing entry is not one this can tilt on, whichever level the tilt reads.

Fitted models

`.propensity` can be the fitted propensity score model instead of the scores it reports. A binomial `stats::glm()` and a two-level `nnet::multinom()` report one score per unit and are tilted as a vector; a `nnet::multinom()` of three or more levels reports one column per level and is tilted as a matrix, with `.focal_level` naming the column `"att"` and `"atu"` read. Those are the shapes `predict(fit, type = "response")` and `fitted(fit)` give, and tilting a fit tilts exactly what tilting those values would. A tilt reads no exposure, so nothing but the scores is taken off the model.

Modified propensity scores

`ps_tilt()` takes plain propensity scores. A score modified by `ps_trim()`, `ps_trunc()`, or `ps_calibrate()` carries a class of its own and has no method here; pass the scores underneath it, with `as.numeric(x)` for a binary exposure or `as.matrix(x)` for a categorical one. Units `ps_trim()` set to `NA` stay `NA` through that extraction and take an `NA` tilt.

See Also

`wt_ate()` and the rest of the weight family, which divide the tilt by the propensity score of the received exposure level.

Examples

```
set.seed(1)
n <- 500
x <- rnorm(n)
z <- rbinom(n, 1, plogis(0.6 * x))
y <- rbinom(n, 1, plogis(-0.5 + 0.8 * z + x))
sim <- data.frame(x = x, z = z, y = y)
ps <- unname(
  predict(glm(z ~ x, data = sim, family = binomial()), type = "response")
)

# a weight is the tilt over the propensity of the received exposure level
received <- z * ps + (1 - z) * (1 - ps)
all.equal(
  as.numeric(wt_ato(ps, z, exposure_type = "binary")),
  ps_tilt(ps, "ato") / received
)
```

```
# tilted g-computation: standardize counterfactual predictions to the
# overlap population with an h-weighted mean, no weights in the outcome model
fit <- glm(y ~ z * x, data = sim, family = binomial())
m1 <- predict(fit, transform(sim, z = 1), type = "response")
m0 <- predict(fit, transform(sim, z = 0), type = "response")

h <- ps_tilt(ps, "ato")
weighted.mean(m1, h) - weighted.mean(m0, h)

# the same predictions standardized to everyone give the ATE instead
mean(m1) - mean(m0)

# the tilt of the scores a fitted model reports
ps_tilt(glm(z ~ x, data = sim, family = binomial()), "ato")

if (rlang::is_installed("nnet")) {
  sim$trt <- factor(sample(c("a", "b", "c"), n, replace = TRUE))
  multinomial_fit <- nnet::multinom(trt ~ x, data = sim, trace = FALSE)
  head(ps_tilt(multinomial_fit, "att", .focal_level = "b"))
}
```

`ps_trim` *Trim Propensity Scores*

Description

Trim observations with extreme propensity scores by replacing them with `NA`, effectively removing those units from downstream analyses. The returned object has the same length (or dimensions) as the input, with trimmed entries set to `NA`. After trimming, refit the propensity score model on the retained observations with `ps_refit()`.

Usage

```
ps_trim(
  .propensity,
 method = c("ps", "adaptive", "pctl", "pref", "cr", "optimal", "density", "resid"),
  lower = NULL,
  upper = NULL,
  .exposure = NULL,
  .focal_level = NULL,
  .reference_level = NULL,
  ...,
  .sigma = NULL,
  .density = "normal",
  .treated = NULL,
  .untreated = NULL,
  ps = lifecycle::deprecated()
)
```

Arguments

`.propensity` A numeric vector of propensity scores in (0, 1) for binary exposures, a matrix / data frame where each column gives the propensity score for one level of a categorical exposure, or a fitted model (see **Fitted models** and **Trimming a dose model** in Details). A data frame trimmed for a binary exposure is reduced to a single column: the second column of a two column data frame, which is the probability of the second level in the layout model predictions come in, and the first column otherwise. The column taken is announced; `options(propensity.quiet = TRUE)` silences the announcement. A matrix is held to the same open interval as a vector, so a score of exactly 0 or 1 in any cell is refused and a separated multinomial fit cannot be repaired by trimming it: setting an extreme score to missing gains nothing from a score already at an endpoint. `ps_trunc()` reads the closed interval for a categorical matrix and is the repair; see **Propensity scores at 0 and 1** in `wt_ate()`. A data frame holding a `.pred_class` column, which a fitted tidymodels classification model returns when no prediction type is named, carries predicted levels rather than probabilities and is refused with an error of class `propensity_df_class_column_error`.

`method` Trimming method. One of:

- `"ps"` (default): Fixed threshold. Observations with propensity scores outside `[lower, upper]` are trimmed. For categorical exposures, observations where *any* column falls below `lower` (the symmetric threshold delta) are trimmed.
- `"adaptive"`: Data-driven threshold that minimizes the asymptotic variance of the IPW estimator (Crump et al., 2009). The `lower` and `upper` arguments are ignored. The threshold adapts to the estimated scores, tightening under poor overlap, and the estimand becomes the population it keeps. `ps_trunc(method = "adaptive")` instead adapts to the sample size alone and keeps every unit.
- `"pctl"`: Quantile-based. Observations outside the `[lower, upper]` quantiles of the propensity score distribution are trimmed. Defaults: `lower = 0.05`, `upper = 0.95`.
- `"pref"`: Preference score trimming. Transforms propensity scores to the preference scale (Walker et al., 2013) and trims outside `[lower, upper]`. Requires `.exposure`. Binary exposures only. Defaults: `lower = 0.3`, `upper = 0.7`.
- `"cr"`: Common range (clinical equipoise). Trims to the overlap region of the propensity score distributions across exposure groups. Requires `.exposure`. Binary exposures only. The `lower` and `upper` arguments are ignored. When the two distributions do not overlap at all, so that the lowest score among the focal units sits above the highest among the reference units, the overlap region is empty and every observed unit is trimmed. That is a truthful record of an empty region rather than an error, and it differs deliberately from `ps_trunc()`, which refuses the same data with an error of class `propensity_no_overlap_error` because there is no range left to bound the scores to.
- `"optimal"`: Multi-category optimal trimming (Yang et al., 2016). Categorical exposures only. Requires `.exposure`.

- `"density"`: Quantile floor on the conditional density of a continuous exposure. Units whose conditional density at their observed dose falls below its `lower` quantile are trimmed. Default: `lower = 0.01`.
- `"resid"`: Bound on the absolute standardized residual of a continuous exposure. Units more than `upper` spreads from their predicted dose are trimmed. `upper` has no default.

`"density"` and `"resid"` need the residuals and the family of the model that fit the exposure's conditional mean, so they accept only a dose model (see **Trimming a dose model** in Details). They refuse a vector or matrix of values, a binomial or quasibinomial `glm`, and a `multinom` with an error of class `propensity_method_error`. A dose model accepts only these two methods, and refuses every other one, including the default `"ps"` when `method` is not supplied, with the same class.

For categorical exposures, only `"ps"` and `"optimal"` are supported.

`lower`, `upper` Numeric thresholds whose interpretation depends on `method`:

- `"ps"`: absolute propensity score bounds (defaults: 0.1, 0.9). For categorical exposures, only `lower` is used, as the symmetric threshold delta, and it defaults to 0.1, the default `ps_trunc()` uses as well. With `k` exposure levels, a threshold of `1/k` or larger cannot be met by every column of a row that sums to one, and is an error.
- `"pctl"`: quantile probabilities (defaults: 0.05, 0.95).
- `"pref"`: preference score bounds (defaults: 0.3, 0.7).
- `"adaptive"`, `"cr"`, `"optimal"`: ignored (thresholds are data-driven).
- `"density"`: `lower` is the quantile probability of the floor, in (0, 0.5) (default 0.01). `upper` is refused with an error of class `propensity_unsupported_arg_error`.
- `"resid"`: `upper` is the bound on the absolute standardized residual, a single positive finite number, and is required (an error of class `propensity_missing_arg_error` without it). `lower` is refused with an error of class `propensity_unsupported_arg_error`.

A `"density"` `lower` or `"resid"` `upper` out of range is refused with an error of class `propensity_range_error`.

`.exposure` An exposure variable. Required for `"pref"`, `"cr"` (binary vector), and `"optimal"` (factor or character). For `"density"` and `"resid"`, the dose is read from the model's response unless supplied here. Not required for other methods.

`.focal_level` The value of `.exposure` representing the focal (treated) group, used by `"pref"` and `"cr"`. Every binary coding honors it: 0/1 numeric, logical, two-level factor, and two-level character exposures are all coded with the named level as focal, and a level the exposure never takes is an error. With no level named, a binary exposure defaults to its higher level, which is `1` for a 0/1 exposure, `TRUE` for a logical one, and the second of the two levels a factor or character exposure takes. Levels a factor declares but never takes are not candidates. Naming any other level reverses the coding, so `.propensity` must then hold the probability of the named level.

`.reference_level`

The value of `.exposure` representing the reference (control) group. Naming it makes the exposure's other level focal, with the same consequence for `.propensity`, and a level the exposure never takes is an error. Automatically detected if not supplied.

`...`	Additional arguments passed to methods.
`.sigma`	For `"density"` and `"resid"`, a single residual spread to read the conditional density at, recorded as `"supplied"`. With none supplied, the spread is the one the density family estimates from the residuals, as in `wt_ate()`: the root mean square, unless the family estimates a scale of its own. A spread for each unit is refused with an error of class `propensity_sigma_error`, and a spread supplied with a family that estimates its own scale is refused with an error of class `propensity_density_error`. The other methods refuse any `.sigma` with an error of class `propensity_sigma_error`.
`.density`	For `"density"` and `"resid"`, the family of the conditional density, in any form `wt_ate()` accepts: `"normal"` (the default), `"laplace"`, `"kernel"`, a specification such as `dens_t()`, or a function of the standardized residual. `"resid"` accepts only `dens_normal()`, `dens_t()`, and `dens_laplace()`, and refuses a kernel or user-written density with an error of class `propensity_density_error`. The other methods refuse any family but the normal with the same class.
`.treated`	**[Deprecated]** Use `.focal_level` instead.
`.untreated`	**[Deprecated]** Use `.reference_level` instead.
`ps`	**[Deprecated]** Use `.propensity` instead. A call that names `ps` must name the arguments after it as well, since a positional argument binds to `.propensity`.

Details

How trimming works:

Trimming identifies observations with extreme (near 0 or 1) propensity scores and sets them to `NA`. These observations are excluded from subsequent weight calculations and effect estimation. The goal is to remove units that lack sufficient overlap between exposure groups, which would otherwise receive extreme weights and destabilize estimates.

Choosing a method:

- Use `"ps"` when you have a specific threshold in mind or want a simple default.
- Use `"adaptive"` for a principled, data-driven cutoff that targets variance reduction.
- Use `"pctl"` to trim a fixed percentage of extreme values from each tail.
- Use `"pref"` when you want to restrict to the region of clinical equipoise based on the preference score.
- Use `"cr"` to restrict to the common support region where both exposure groups have observed propensity scores.
- Use `"optimal"` for multi-category (3+) exposures; this is the only data-driven method available for categorical treatments.
- Use `"density"` or `"resid"` for a continuous exposure, on its dose model.

For a binary or categorical exposure, trimming and refitting is the first recourse when the analysis can accept a change of estimand. When every unit must be kept, bound the scores with `ps_trunc()` instead, for a binary exposure with `method = "adaptive"`, or bound the weights with `wt_trunc()`.

Typical workflow:

1. Fit a propensity score model

2. Apply ps_trim() to flag extreme values
3. Call ps_refit() to re-estimate propensity scores on the retained sample
4. Compute weights with wt_ate() or another weight function

Fitted models:

.propensity can be the fitted propensity score model instead of the scores it reports. A binomial stats::glm() and a two-level nnet::multinom() are read as one score per unit; a nnet::multinom() of three or more levels is read as one column per level. Those are the shapes predict(fit, type = "response") and fitted(fit) give, and trimming a fit trims exactly what trimming those values would.

The methods that read an exposure ("pref", "cr", and every method on the categorical route) take it from the model when .exposure is not supplied, announcing the variable they read; options(propensity.quiet = TRUE) silences the announcement. An .exposure you supply is used instead, and a categorical model's columns are matched to its levels by name, so an exposure whose levels are ordered differently is still trimmed against the right column.

Trimming a dose model:

For a continuous exposure, .propensity can be the model of the exposure's conditional mean: a stats::lm(), a glm of the gaussian() family (or another family whose variance is constant), a MASS::rlm(), or an mgcv::gam(), the models wt_ate() reads a dose from. A glm whose spread changes with its mean, such as poisson(), is refused with an error of class propensity_model_family_error. Only "density" and "resid" apply.

For a dose, the generalized propensity score is the conditional density $f(a \mid x)$ (Hirano and Imbens, 2004), and a unit whose observed dose has a small conditional density gets a large weight, as a unit with a propensity score near 0 does for a binary exposure. Both methods read the fitted conditional mean mu, the spread sigma of the residuals under the family in .density, the standardized residual z = (a - mu) / sigma, and the conditional density f = g(z) / sigma, where g is the family's standardized density. "density" keeps the units whose f is at least its lower quantile. "resid" keeps those whose |z| is at most upper; for the symmetric unimodal families it accepts, that is the floor g(upper) / sigma on f, which is the threshold it records. The trim is one-sided, at the low end of the density, since only a small density makes a large weight (Branson et al., 2024).

The retained values are the conditional means, and the trimmed ones are NA. A unit with a missing mean or dose takes no part in the trim. .focal_level, .reference_level, and their deprecated forms are refused with an error of class propensity_focal_level_error, since a dose has no levels.

A trim changes the estimand. The analysis describes the units whose observed dose was plausible under the model, not the full sample, and which units those are depends on the threshold. Branson et al. (2024) define the trimmed dose response at each dose among the units whose conditional density at that dose exceeds the threshold, a population that changes with the dose; this trim is its sample analogue at each unit's own observed dose rather than the same population. A dose-response curve fit to the trimmed sample is a curve for the units it keeps.

After trimming, ps_refit() refits the conditional mean on the retained rows and re-estimates the spread there under the recorded family, and wt_ate() builds the weights from the refit under the family and spread the record holds. ipw() refuses those weights, as it refuses every weight built from a trimmed score.

The other way to hold down extreme weights for a dose is wt_trunc(), which bounds the weights and keeps every unit; see **Truncating weights or trimming the density** there for the simulation

results behind this summary. With a density family that fits the residuals, a density trim at `lower = 0.01` did not help: at an effective sample size of 64% of the sample, its interval covered 0.843, against 0.933 for the untrimmed weights under the same fixed-weight interval. A loose bound on the weights left that coverage where it was and lowered the root mean squared error. With heavy-tailed residuals read through a normal density, or a residual spread the model left out, the trim covered 0.979 to 0.989, against 0.618 to 0.881 for the untrimmed weights and 0.644 to 0.906 for the bound, because it removed the units the misspecified density fit worst. Two cautions go with those numbers. The trim's intervals were computed on the trimmed sample, held the weights fixed, and were conservative, and its bias was measured against the effect in the full population, so part of that bias is the change of estimand. `"resid"` was not simulated; the results carry over to it only because a bound on the standardized residual is a floor on the conditional density.

A family that fits the residuals is the better repair where it applies. Under heavy tails, `dens_t()` with 4 degrees of freedom at its default scale had about a third of the root mean squared error of the bounded normal weights and an M-estimation interval that covered 0.926 to 0.939 with standard errors close to right, and it kept every unit. The trim's coverage was higher, but its interval was conservative. Check the family first; trim when the units at the low-density end are ones the analysis should not describe, and say so when reporting the estimand.

Object behavior:

Arithmetic operations on `ps_trim` objects return plain numeric vectors, since transformed propensity scores (e.g., `1/ps`) are no longer propensity scores. Trimmed values propagate as `NA` in calculations; use `na.rm = TRUE` where appropriate.

When combining `ps_trim` objects with `c()`, trimming parameters must match. Mismatched parameters trigger a warning and return a numeric vector.

Use `ps_trim_meta()` to inspect the trimming metadata, including the method, cutoffs, and which observations were retained or trimmed.

Missing values:

A propensity score that arrives missing is not one this function removed, so it takes no part in the trimming record: it joins neither the retained nor the trimmed positions, and `is_unit_trimmed()` reports `FALSE` for it. The value propagates as `NA` into the result. For a matrix of categorical propensity scores, a row with a missing cell comes back exactly as it arrived, since there is no complete probability vector to place against a threshold.

The methods that read a cutoff off the scores read it off the scores they have. `"adaptive"`, `"pctl"`, `"cr"`, and `"optimal"` all work their cutoffs out from the complete scores or rows, so the cutoff is the one the same call would produce with the missing observations dropped.

`"pref"` centers its preference scores on the proportion exposed across the whole sample, which is a fact about the exposure rather than about the scores, so a unit whose propensity score is missing still counts toward it and the cutoff is not the one the shorter call would produce. That unit's own preference score is missing, which leaves it outside both the retained and the trimmed positions like any other missing score.

A missing exposure is a different matter, and `"pref"` and `"cr"` refuse one with an error of class `propensity_missing_value_error`. Their cutoffs come from the exposure groups, and a unit that belongs to neither leaves them undefined. Remove or impute the missing exposure values first.

The trimming record:

A ps_trim records which units were trimmed as positions among the observations it was written for, along with how many observations that was. Operations that hand this package the subscript re-index those positions onto the result: subsetting with [, sort(), rep(), and na.omit() all return a record written for what they return, and a subscript naming a position more than once reports that unit at every place it now holds. unique() does the same when it can, as described below.

Operations that change how many observations there are without supplying a subscript cannot re-index the record, and it is dropped rather than worked out from the values, since reading membership back from the NA pattern would report a propensity score that arrived missing as one this package removed. vctrs::vec_slice(), which is how filtering, joining, and grouped verbs in dplyr reach a column, is the usual route, and combining two or more vectors with c() is another, because concatenation appends one set of observations to another. The record is dropped without comment on every route: most of these length changes build vectors the caller never holds, such as the pieces a grouped verb slices a column into or the rows a tibble slices off to print, so a warning would mostly describe something that is not the result. The values, the class, and the method and its cutoffs are untouched.

A combine drops the positions even when it is handed a single vector. c() of one ps_trim returns it unchanged, record included, but vctrs::vec_c(x), dplyr::bind_rows(df), vctrs::vec_rbind(df), and an ungrouped dplyr::reframe() rebuild the column, so a later ps_refit() or is_unit_trimmed() on the result refuses it.

unique() keeps one element for each distinct value, or one row for each distinct row of a matrix of scores, and that element or row stands for every unit holding the same scores. A matrix comes back as a matrix of the same class with its column names. The record is re-indexed onto the result when all of the merged units share one status, and dropped otherwise: a trimmed score and one that arrived missing are both NA, and a trimmed row is NA throughout, so a vector or matrix holding both returns a single NA element or row that neither status describes.

as.data.frame() and tibble::as_tibble() turn a matrix of scores into a data frame whose columns are ps_trim vectors carrying the matrix's record, one row per unit, so the weight functions read the data frame as they read the matrix and a subset of its rows re-indexes the record. A data frame whose columns are not all trimmed with the same record is refused by the weight functions with an error of class propensity_matrix_type_error. A matrix without column names gives columns named V1, V2, and so on, which the weight functions then refuse because they name no exposure level, so name the columns after the exposure levels before converting.

A record can also outlive the observations it describes, because it travels by routes vctrs does not see: growing a ps_trim by subassignment carries it across the length change. is_unit_trimmed() and ps_refit() therefore check that the record covers the object they are given and raise an error of class propensity_missing_meta_error when it does not, rather than name trimmed units at stale positions.

That check compares how many observations the record was written for against how many the object holds, which a reordering does not change, so a route that could reorder the observations without saying how drops the positions instead, at any length: vctrs::vec_slice(), dplyr::arrange(), dplyr::filter(), and vctrs::vec_assign() and the helpers built on it return a ps_trim whose record keeps its method and cutoffs and names no units, and is_unit_trimmed() and ps_refit() refuse it. Subsetting with [, sort(), unique(), and rep() know where the units went and re-index the record, and [<- and is.na<- move no unit and keep it, so reorder with [, or put the propensity scores in the order you want before trimming them.

Casting a numeric vector into a ps_trim with vctrs::vec_cast() is a type operation and not a trimming. The result is described by the method and cutoffs of the target and records that none

of the arriving values was trimmed, so it can hold scores outside those cutoffs, including 0 and 1. Call `ps_trim()` on the scores to trim them.

Value

A `ps_trim` object (a numeric vector with class `"ps_trim"`, or a matrix with class `"ps_trim_matrix"`). Trimmed observations are `NA`. Metadata is stored in the `"ps_trim_meta"` attribute and can be accessed with `ps_trim_meta()`. Key fields include:

- `method`: the trimming method used
- `keep_idx`: integer indices of retained observations
- `trimmed_idx`: integer indices of trimmed (NA) observations
- `n_obs`: the number of observations those indices describe
- Method-specific fields such as `cutoff` (adaptive), `q_lower`/`q_upper` (pctl), `cr_lower`/`cr_upper` (cr), `delta` (categorical ps), or `lambda` (optimal)
- `focal_inverted` (vector scores only): `TRUE` when the scores are one minus the probability a fitted model reports, because the model was trimmed with its first level named as focal, and `FALSE` otherwise, including for every vector of scores supplied directly

A trim of a dose model records `method`, `lower` (`"density"`, else `NULL`), `upper` (`"resid"`, else `NULL`), `threshold` (the realized floor on the conditional density), `sigma` (the spread it was read at), `sigma_kind` (`"pooled"`, `"mle"`, or `"supplied"`), `density` (the density specification), `keep_idx`, `trimmed_idx`, and `n_obs`.

References

Branson, Z., Kennedy, E. H., Balakrishnan, S., & Wasserman, L. (2024). Causal effect estimation after propensity score trimming with continuous treatments. *arXiv preprint* arXiv:2309.00706.

Crump, R. K., Hotz, V. J., Imbens, G. W., & Mitnik, O. A. (2009). Dealing with limited overlap in estimation of average treatment effects. *Biometrika*, 96(1), 187–199.

Hirano, K., & Imbens, G. W. (2004). The propensity score with continuous treatments. In *Applied Bayesian Modeling and Causal Inference from Incomplete-Data Perspectives* (pp. 73–84).

Walker, A. M., Patrick, A. R., Lauer, M. S., et al. (2013). A tool for assessing the feasibility of comparative effectiveness research. *Comparative Effectiveness Research*, 3, 11–20.

Yang, S., Imbens, G. W., Cui, Z., Faries, D. E., & Kadziola, Z. (2016). Propensity score matching and subclassification in observational studies with multi-level treatments. *Biometrics*, 72(4), 1055–1065.

See Also

`ps_trunc()` for bounding (winsorizing) instead of discarding, `ps_refit()` to re-estimate propensity scores after trimming, `ps_calibrate()` for calibration-based adjustment, `ps_trim_meta()` to inspect trimming metadata, `is_ps_trimmed()` and `is_unit_trimmed()` for logical queries.

Examples

```
set.seed(2)
n <- 300
x <- rnorm(n)
z <- rbinom(n, 1, plogis(1.3 * x))
fit <- glm(z ~ x, family = binomial)
ps <- predict(fit, type = "response")

# Fixed threshold trimming (default)
trimmed <- ps_trim(ps, method = "ps", lower = 0.1, upper = 0.9)
trimmed

# How many observations were trimmed?
sum(is_unit_trimmed(trimmed))

# Data-driven adaptive trimming
ps_trim(ps, method = "adaptive")

# Quantile-based trimming at 5th and 95th percentiles
ps_trim(ps, method = "pctl")

# Refit after trimming, then compute weights
trimmed <- ps_trim(ps, method = "adaptive")
refitted <- ps_refit(trimmed, fit)
wt_ate(refitted, .exposure = z)

# Trim the scores a fitted model reports, reading the exposure off the model
ps_trim(fit, method = "cr")

# Trim a dose model on the scale of its conditional density
dose <- 1 + 0.5 * x + rt(n, df = 3)
dose_fit <- lm(dose ~ x)
ps_trim(dose_fit, method = "density", lower = 0.05)
ps_trim(dose_fit, method = "resid", upper = 3, .density = dens_t(4))

if (rlang::is_installed("nnet")) {
  trt <- factor(sample(c("a", "b", "c"), n, replace = TRUE))
  multinomial_fit <- nnet::multinom(trt ~ x, trace = FALSE)
  ps_trim(multinomial_fit, method = "optimal")
}
```

`ps_trim_meta` *Extract trimming metadata from a* `ps_trim` *object*

Description

`ps_trim_meta()` returns the metadata list attached to a `ps_trim` object by `ps_trim()`.

Usage

```
ps_trim_meta(x)
```

Arguments

x	A ps_trim object.

Value

A named list with elements:

`method` Character string indicating the trimming method used.

`keep_idx` Integer vector of retained observation indices.

`trimmed_idx` Integer vector of trimmed observation indices.

`n_obs` The number of observations those indices describe.

`lower`, `upper` Numeric cutoffs, when applicable.

`refit` Logical, `TRUE` if the model was refit via `ps_refit()`.

`focal_inverted` For a vector of scores, logical, `TRUE` if the scores are one minus the probability the fitted model reports, which `ps_refit()` reads to predict the same level.

Additional method-specific elements (e.g. `cutoff`, `delta`, `lambda`) may also be present.

`keep_idx`, `trimmed_idx`, and `n_obs` are absent when an operation that changed the number of observations dropped the record; see `ps_trim()`.

See Also

`ps_trim()` for trimming propensity scores, `is_ps_trimmed()` and `is_unit_trimmed()` for predicate queries.

Examples

```
ps <- c(0.05, 0.3, 0.6, 0.95)
trimmed <- ps_trim(ps, method = "ps", lower = 0.1, upper = 0.9)

ps_trim_meta(trimmed)
```

---

ps_trunc *Truncate (Winsorize) Propensity Scores*

Description

`ps_trunc()` bounds extreme propensity scores to fixed limits, replacing out-of-range values with the boundary value (a form of *winsorizing*). The result is a vector or matrix of the same length and dimensions as `.propensity`, with no observations removed. This contrasts with `ps_trim()`, which sets extreme values to `NA` (effectively removing those observations from analysis).

Usage

```
ps_trunc(
  .propensity,
  method = c("ps", "adaptive", "pctl", "cr"),
  lower = NULL,
  upper = NULL,
  .exposure = NULL,
  .focal_level = NULL,
  .reference_level = NULL,
  ...,
  .treated = NULL,
  .untreated = NULL,
  ps = lifecycle::deprecated()
)
```

Arguments

`.propensity` A numeric vector of propensity scores in (0, 1) (binary exposures), or a matrix/data.frame where each column contains propensity scores for one level of a categorical exposure. A data frame truncated for a binary exposure is reduced to a single column: the second column of a two column data frame, which is the probability of the second level in the layout model predictions come in, and the first column otherwise. The column taken is announced; `options(propensity.quiet = TRUE)` silences the announcement. A vector is held to the open interval, so a score of exactly 0 or 1 is refused. A matrix is held to the closed interval instead: a cell of exactly 0 or 1 is bounded rather than refused, which is how a separated multinomial fit is repaired, and what comes back lies strictly inside (0, 1). Anything outside $[0, 1]$ is refused either way, and so is a truncation whose own computed bounds leave a score at an endpoint; see **Propensity scores at 0 and 1** in `wt_ate()`. A data frame holding a `.pred_class` column, which a fitted tidymodels classification model returns when no prediction type is named, carries predicted levels rather than probabilities and is refused with an error of class `propensity_df_class_column_error`.

`method` One of `"ps"`, `"adaptive"`, `"pctl"`, or `"cr"`:

- `"ps"` (default): Truncate directly on propensity score values. Values outside `[lower, upper]` are set to the nearest bound. For categorical exposures, applies symmetric truncation using `lower` as the threshold (delta) and renormalizes rows to sum to 1.
- `"adaptive"`: Truncate at $[1/c, 1 - 1/c]$ with $c = \sqrt{n}\log(n)/5$ (Gruber et al., 2022), where n is the number of propensity scores present (binary exposures only). See **The adaptive bound** below.
- `"pctl"`: Truncate at quantiles of the propensity score distribution. The `lower` and `upper` arguments specify quantile probabilities. For categorical exposures, quantiles are computed across all columns.
- `"cr"`: Truncate to the common range of propensity scores across exposure groups (binary exposures only). Bounds are `[min(.propensity[focal]), max(.propensity[refe`

Requires `.exposure`. When those bounds cross, so that the two distributions do not overlap at all, there is no common range to bound the scores to and the call errors with class `propensity_no_overlap_error`. That differs deliberately from `ps_trim()`, which trims every observed unit in the same situation, a truthful record of an empty overlap region.

For categorical exposures, only `"ps"` and `"pctl"` are supported.

`lower, upper` Bounds for truncation. Interpretation depends on `method`:

- `method = "adaptive"`: Not used. Supplying either is ignored with a warning.
- `method = "ps"`: Propensity score values. See **The `"ps"` bounds** below.
- `method = "pctl"`: Quantile probabilities (defaults: 0.05 and 0.95; categorical defaults: 0.01 and 0.99).
- `method = "cr"`: Ignored; bounds are determined by the data.

`.exposure` An exposure vector. Required for method `"cr"` (binary exposure vector) and for categorical exposures (factor or character vector) with any method.

`.focal_level` The value of `.exposure` representing the focal (treated) group, used by `"cr"`. Every binary coding honors it: 0/1 numeric, logical, two-level factor, and two-level character exposures are all coded with the named level as focal, and a level the exposure never takes is an error. With no level named, a binary exposure defaults to its higher level, which is `1` for a 0/1 exposure, `TRUE` for a logical one, and the second of the two levels a factor or character exposure takes. Levels a factor declares but never takes are not candidates. Naming any other level reverses the coding, so `.propensity` must then hold the probability of the named level.

`.reference_level`
The value of `.exposure` representing the reference (control) group. Naming it makes the exposure's other level focal, with the same consequence for `.propensity`, and a level the exposure never takes is an error. Automatically detected if not supplied.

`...` Additional arguments passed to methods.

`.treated` **[Deprecated]** Use `.focal_level` instead.

`.untreated` **[Deprecated]** Use `.reference_level` instead.

`ps` **[Deprecated]** Use `.propensity` instead. A call that names `ps` must name the arguments after it as well, since a positional argument binds to `.propensity`.

Details

Unlike `ps_trim()`, truncation preserves all observations. No `NA` values are introduced; out-of-range scores are replaced with the boundary value.

For **binary exposures**, each propensity score e_i is bounded:

- If $e_i < l$, set $e_i = l$ (the lower bound).
- If $e_i > u$, set $e_i = u$ (the upper bound).

For **categorical exposures**, values below the threshold are set to the threshold and each row is renormalized to sum to 1.

The "ps" bounds:

With `method = "ps"`, both paths default to a floor of 0.1. For a **binary exposure**, each bound supplied must be a single number strictly between 0 and 1, and a bound supplied alone implies its mirror: `lower` alone bounds the scores at `[lower, 1 - lower]`, and `upper` alone at `[1 - upper, upper]`. With neither supplied, the bounds are `[0.1, 0.9]`; with both supplied, both are used as written, so an asymmetric bound must be given in full. The mirror follows from the weights: an untreated unit's ATE weight is $1/(1-e)$, so a floor on e alone would bound the treated weights and leave the untreated ones unbounded. A lone `lower` of 0.5 or more, or a lone `upper` of 0.5 or less, meets or crosses its own mirror and is an error of class `propensity_range_error`. `lower = 1/c` gives the bounds `method = "adaptive"` computes for the same c.

For a **categorical exposure**, `lower` is the threshold δ (default 0.1) applied to every column, after which each row is renormalized, and `upper` is not read and is recorded as `NA`. With `k` exposure levels, a threshold of `1/k` or larger cannot be met by every column of a row that sums to one and is an error, so an exposure with ten or more levels needs an explicit `lower` below `1/k`.

The adaptive bound:

`method = "adaptive"` bounds a binary propensity score at $[1/c, 1-1/c]$, where $c = \sqrt{n}\log(n)/5$ is the weight bound of Gruber et al. (2022) and n counts the scores that are present, as in `wt_trunc()`. At $n = 1000$, c is 43.7 and the bounds are 0.023 and 0.977. The bound adapts to the sample size alone, loosening as n grows, and leaves the estimand alone. `ps_trim(method = "adaptive")` instead adapts to the estimated scores, tightening under poor overlap, and changes the estimand to the population it keeps.

For **unstabilized ATE weights** the bound is the weight bound written on the score scale. A treated unit's weight is $1/e$, so the floor $1/c$ caps the treated weights at c; an untreated unit's weight is $1/(1-e)$, so the ceiling $1-1/c$ caps the untreated weights at c. `wt_ate()` on the result therefore gives the weights `wt_trunc(wt_ate(ps, .exposure), method = "adaptive")` gives, with one difference, provided every unit's exposure is observed. (A unit with a present score and a missing exposure counts toward n here but has no weight for `wt_trunc()` to count, so the two bounds then differ.) The difference is this: a treated unit above $1-1/c$ or an untreated unit below $1/c$, whose weight is already close to 1, is moved to that bound too, and its weight rises to $c/(c-1)$. Every weight above $c/(c-1)$ is the same under either route. A lower bound alone would not do the same work, since no floor on e bounds an untreated unit's weight.

For **stabilized ATE weights** the equivalence does not carry over as it stands. Stabilizing multiplies each arm's weights by that arm's prevalence, so the treated weights are capped at $P(A=1)c$ and the untreated weights at $P(A=0)c$, which agree only when the two arms are equally common.

The bounds cross when $c < 2$, which holds for fewer than 15 scores, so `"adaptive"` needs at least 15 scores present and raises an error of class `propensity_range_error` otherwise. It is refused for categorical exposures with an error of class `propensity_method_error`: truncating a categorical score renormalizes each row, which moves scores the bound never reached, so the bound no longer caps the weights. A two-level `nnet::multinom()` fit is read as one binary score and accepted.

Arithmetic behavior: Arithmetic operations on `ps_trunc` objects return plain numeric vectors. Once propensity scores are transformed (e.g., into weights), the result is no longer a propensity score.

Combining behavior: Combining `ps_trunc` objects with `c()` requires matching truncation parameters. Mismatched parameters produce a warning and return a plain numeric vector.

Fitted models:

.propensity can be the fitted propensity score model instead of the scores it reports. A binomial stats::glm() and a two-level nnet::multinom() are read as one score per unit; a nnet::multinom() of three or more levels is read as one column per level. Those are the shapes predict(fit, type = "response") and fitted(fit) give, and truncating a fit bounds exactly what bounding those values would. A model of a continuous exposure's conditional mean, such as a stats::lm() fit or a gaussian stats::glm(), has no propensity score to bound and raises an error of class propensity_model_family_error. A dose has two routes instead: trim the dose model with ps_trim(method = "density"), which sets aside the units whose dose is implausible under the model, or build the weights with wt_ate() and bound them with wt_trunc(), which keeps every unit. There is no density method here: a floor on a dose's conditional density is a bound on its unstabilized weights, the ones wt_ate(stabilize = FALSE) builds, which wt_trunc() already applies.

The methods that read an exposure ("cr", and every method on the categorical route) take it from the model when .exposure is not supplied, announcing the variable they read; options(propensity.quiet = TRUE) silences the announcement. An .exposure you supply is used instead, and a categorical model's columns are matched to its levels by name, so an exposure whose levels are ordered differently is still bounded against the right column.

Missing values:

A propensity score that arrives missing is not one this function pinned to a bound, so it takes no part in the truncation record: it never joins truncated_idx, and is_unit_truncated() reports FALSE for it. The value stays missing in the result. For a matrix of categorical propensity scores, a row with a missing cell comes back missing throughout, because renormalizing a row divides by its sum and that sum is unknown.

The methods that read their bounds off the data read them off the data they have. "pctl" and "cr" work their bounds out from the complete scores, and the categorical "pctl" bounds come from the cells of the complete rows, so the bounds are the ones the same call would produce with the missing observations dropped.

A missing exposure is a different matter, and "cr" refuses one with an error of class propensity_missing_value_error. Its bounds come from the exposure groups, and a unit that belongs to neither leaves them undefined. Remove or impute the missing exposure values first.

The truncation record:

A ps_trunc records which units were winsorized as positions among the observations it was written for, along with how many observations that was. Operations that hand this package the subscript re-index those positions onto the result: subsetting with [, sort(), and rep() all return a record written for what they return, and a subscript naming a position more than once reports that unit at every place it now holds. unique() does the same when it can, as described below.

Operations that change how many observations there are without supplying a subscript cannot reindex the record, and it is dropped rather than worked out from the values, since a score that arrived equal to a bound is indistinguishable from one this function pinned there. vctrs::vec_slice(), which is how filtering, joining, and grouped verbs in dplyr reach a column, is the usual route, and combining two or more vectors with c() is another, because concatenation appends one set of observations to another. The record is dropped without comment on every route, since most of these length changes build vectors the caller never holds, such as the pieces a grouped verb slices a column into. The values, the class, and the method and its bounds are untouched.

A combine drops the positions even when it is handed a single vector. c() of one ps_trunc returns it unchanged, record included, but vctrs::vec_c(x), dplyr::bind_rows(df), vctrs::vec_rbind(df),

and an ungrouped `dplyr::reframe()` rebuild the column, so a later `is_unit_truncated()` on the result refuses it.

`unique()` keeps one element for each distinct value, or one row for each distinct row of a matrix of scores, and that element or row stands for every unit holding the same scores. A matrix comes back as a matrix of the same class with its column names. The record is re-indexed onto the result when all of the merged units share one status, and dropped otherwise: a score or row that arrived where truncation would put it and one truncation moved there merge into a single element or row that neither status describes.

`as.data.frame()` and `tibble::as_tibble()` turn a matrix of scores into a data frame whose columns are `ps_trunc` vectors carrying the matrix's record, one row per unit, so the weight functions read the data frame as they read the matrix and a subset of its rows re-indexes the record. A data frame whose columns are not all truncated with the same record is refused by the weight functions with an error of class `propensity_matrix_type_error`. A matrix without column names gives columns named `V1`, `V2`, and so on, which the weight functions then refuse because they name no exposure level, so name the columns after the exposure levels before converting.

A record can also outlive the observations it describes, because it travels by routes vctrs does not see: growing a `ps_trunc` by subassignment carries it across the length change. `is_unit_truncated()` therefore checks that the record covers the object it is given and raises an error of class `propensity_missing_meta_error` when it does not, rather than name truncated units at stale positions.

That check compares how many observations the record was written for against how many the object holds, which a reordering does not change, so a route that could reorder the observations without saying how drops the positions instead, at any length: `vctrs::vec_slice()`, `dplyr::arrange()`, `dplyr::filter()`, and `vctrs::vec_assign()` and the helpers built on it return a `ps_trunc` whose record keeps its method and bounds and names no units, and `is_unit_truncated()` refuses it. Subsetting with `[`, `sort()`, `unique()`, and `rep()` know where the units went and re-index the record, and `[<-` and `is.na<-` move no unit and keep it, so reorder with `[`, or put the propensity scores in the order you want before truncating them.

Casting a numeric vector into a `ps_trunc` with `vctrs::vec_cast()` is a type operation and not a truncation. The result is described by the method and bounds of the target and records that none of the arriving values was pinned to a bound, so it can hold scores outside those bounds, including 0 and 1. Call `ps_trunc()` on the scores to truncate them.

Value

A `ps_trunc` object (a numeric vector for binary exposures, or a matrix for categorical exposures). Use `ps_trunc_meta()` to inspect metadata including `method`, `lower_bound`, `upper_bound`, `truncated_idx` (positions of modified values), and `n_obs` (the number of observations those positions describe).

References

Crump, R. K., Hotz, V. J., Imbens, G. W., & Mitnik, O. A. (2009). Dealing with limited overlap in estimation of average treatment effects. *Biometrika*, 96(1), 187–199.

Gruber, S., Phillips, R. V., Lee, H., & van der Laan, M. J. (2022). Data-adaptive selection of the propensity score truncation level for inverse-probability-weighted and targeted maximum likelihood estimators of marginal point treatment effects. *American Journal of Epidemiology*, 191(9), 1640–1651.

Walker, A. M., Patrick, A. R., Lauer, M. S., et al. (2013). A tool for assessing the feasibility of comparative effectiveness research. *Comparative Effectiveness Research*, 3, 11–20.

See Also

`ps_trim()` for removing (rather than bounding) extreme values, `ps_refit()` for refitting the propensity model after trimming, `is_ps_truncated()`, `is_unit_truncated()`, `ps_trunc_meta()`

Examples

```
set.seed(2)
n <- 200
x <- rnorm(n)
z <- rbinom(n, 1, plogis(1.2 * x))
fit <- glm(z ~ x, family = binomial)
ps <- predict(fit, type = "response")

# Truncate to [0.1, 0.9]
ps_t <- ps_trunc(ps, method = "ps", lower = 0.1, upper = 0.9)
ps_t

# Bound at [1/c, 1 - 1/c], c = sqrt(n) log(n) / 5, which caps the
# unstabilized ATE weights at c
ps_trunc(ps, method = "adaptive")

# Truncate at the 1st and 99th percentiles
ps_trunc(ps, method = "pctl", lower = 0.01, upper = 0.99)

# Use truncated scores to calculate weights
wt_ate(ps_t, .exposure = z)

# Inspect which observations were truncated
is_unit_truncated(ps_t)

# Bound the scores a fitted model reports, reading the exposure off the model
ps_trunc(fit, method = "cr")

if (rlang::is_installed("nnet")) {
  trt <- factor(sample(c("a", "b", "c"), n, replace = TRUE))
  multinomial_fit <- nnet::multinom(trt ~ x, trace = FALSE)
  ps_trunc(multinomial_fit, method = "ps")
}
```

---

ps_trunc_meta *Extract truncation metadata from a* ps_trunc *object*

Description

Returns the metadata list attached to a `ps_trunc` object. The list includes fields such as `method`, `lower_bound`, `upper_bound`, and `truncated_idx`.

Usage

```
ps_trunc_meta(x)
```

Arguments

x	A ps_trunc object created by ps_trunc().

Value

A named list with truncation metadata, including:

- method – the truncation method used ("ps", "adaptive", "pctl", or "cr")
- lower_bound, upper_bound – the applied bounds
- truncated_idx – integer positions of values that were winsorized
- n_obs – the number of observations those positions describe

truncated_idx and n_obs are absent when an operation that changed the number of observations dropped the record; see ps_trunc().

See Also

ps_trunc(), is_ps_truncated(), is_unit_truncated()

Examples

```
ps <- c(0.02, 0.3, 0.5, 0.7, 0.98)
ps_t <- ps_trunc(ps, method = "ps", lower = 0.05, upper = 0.95)
ps_trunc_meta(ps_t)
```

---

psw *Propensity Score Weight Vectors*

Description

psw objects are numeric vectors that carry metadata about propensity score weights, including the target estimand and whether the underlying propensity scores were trimmed, truncated, or calibrated.

Most users will encounter psw objects as return values from wt_ate() and related weight functions. These constructor and helper functions are useful for inspecting weight objects or for package developers extending propensity.

Usage

```
new_psw(
  x = double(),
  estimand = NULL,
  stabilized = FALSE,
  trimmed = FALSE,
  truncated = FALSE,
  calibrated = FALSE,
  stabilization_score = NULL,
  wt_truncated = FALSE,
  ...
)

psw(
  x = double(),
  estimand = NULL,
  stabilized = FALSE,
  trimmed = FALSE,
  truncated = FALSE,
  calibrated = FALSE,
  stabilization_score = NULL,
  wt_truncated = FALSE
)

is_psw(x)

is_stabilized(wt)

stabilization_score(wt)

as_psw(x, estimand = NULL)
```

Arguments

x	For `psw()` and `new_psw()`: a numeric vector of weights (default: `double()`). For `is_psw()` and `as_psw()`: an object to test or coerce.
estimand	A character string identifying the target estimand (e.g., `"ate"`, `"att"`, `"ato"`). Defaults to `NULL`.
stabilized	Logical. Were the weights stabilized? Defaults to `FALSE`.
trimmed	Logical. Were the weights derived from trimmed propensity scores? Defaults to `FALSE`.
truncated	Logical. Were the weights derived from truncated propensity scores? Defaults to `FALSE`.
calibrated	Logical. Were the weights derived from calibrated propensity scores? Defaults to `FALSE`.

stabilization_score
: Optional numeric stabilization score to record on the object, either a single value or one value per observation. Every value must be positive and finite. Defaults to NULL, meaning no fixed score was supplied.

wt_truncated
: Logical. Were the weights themselves truncated? This is separate from truncated, which describes the propensity scores the weights were built from. Defaults to FALSE.

...
: Additional attributes stored on the object (developer use only).

wt
: A psw or causal_wts object.

Details

Constructors:

- psw() is the **user-facing** constructor. It coerces x to double and validates inputs before creating the object.
- new_psw() is the **low-level** constructor intended for developers. It assumes x is already a double vector and performs minimal validation.
- as_psw() coerces an existing numeric vector to a psw object.

Queries:

- is_psw() tests whether an object is a psw vector.
- is_stabilized() returns TRUE if the weights are stabilized.
- is_wt_truncated() returns TRUE if the weights themselves were truncated, which is recorded apart from whether the propensity scores they were built from were.
- stabilization_score() returns the user-supplied stabilization score, or NULL when none was recorded or when a per-observation score was dropped because an operation changed the length of the weights.
- numerator_model() returns the fitted model supplied to stabilize that estimated the numerator of the weights, or NULL when no model did.

The stabilization score:

A stabilization_score is the multiplier the weights were stabilized on. It holds either a single value, which scales every weight, or one value per observation, which scales each weight by its own. Both forms are checked where the score is recorded: a score must be numeric, positive, and finite, and hold a length the weights can use, and one that does not is refused with an error of class propensity_stabilization_score_error. A score is recorded as a plain double vector, with its storage type normalized and any names dropped, so a score written 1L and one written 1 are the same score and combine without conflict.

The exposure records:

Weights built by wt_ate() and its relatives also record the exposure they were built for: exposure_type names its type, and density_meta describes the ratio of densities that weights for a continuous exposure are. Both are read back with exposure_type() and density_meta(), and both describe the exposure rather than the units, so neither carries a length of its own.

A zero-length psw is a prototype: it records what a result built from it will carry rather than describing observations of its own, so a per-observation score on one is recorded as given and checked for length when the observations arrive. A cast whose data arrives at a length the score

does not match drops the score rather than refusing the cast, since the score describes the prototype's observations and says nothing about the data being cast. A cast to no observations keeps it, having brought no observations for it to contradict.

psw objects also inherit the broader causal_wts class. The accessors that class carries, causalgenerics::is_causal_wt(causalgenerics::estimand(), and estimand<-, are re-exported by propensity and documented at causalgenerics::estimand().

Arithmetic and combining:

Arithmetic operations on psw objects preserve the class and attributes, so operations like normalization (weights / sum(weights)) retain metadata.

An operation between two psw objects merges what each of them records. Two different estimands are pasted together, and an estimand only one operand names stands for the result; the result is stabilized only when both operands are, and it is marked as trimmed, truncated, calibrated, or weight-truncated when either operand is. The remaining attributes, the stabilization_score, the records left by a modified propensity score, the attributes describing a categorical exposure, and the exposure records, are carried by agreement: one only a single operand records carries, and one both record with the same value carries. One they record differently is dropped, since neither value describes the result, and a warning of class propensity_metadata_conflict_warning names it, once for each attribute dropped that way and whatever order the inputs were given in. The rule is applied per operation, so an attribute one operation drops for a disagreement can be carried again by a later operation whose operands agree. Two density records agree when they name the same family with the same parameters and the same numerator and residual spread, so weights built the same way in two calls agree even though each call builds its own copy of the function that evaluates the density. A numerator estimated by a fitted model is read the same way: two records agree when their models write the same formula and were fit to the same coefficients, and a model is not compared against the marginal density or against no numerator at all, both of which are disagreements.

A density_meta record describes a continuous exposure, so a result that drops exposure_type for a disagreement drops the density record with it, without a second warning: the record has nothing left to be about.

A stabilization_score is carried only when the result is stabilized, which takes both operands. A result that is not stabilized drops the score without comment.

Combining psw objects with c() preserves the class only when all metadata matches; mismatched metadata produces a warning and falls back to a plain numeric vector. The trimming, truncation, and weight truncation records are compared by what they say about the modification: weights built from scores trimmed or truncated differently, from a trim that was refit and one that was not, or truncated at different bounds describe different estimands and are combined only as numbers. Weights flagged as modified with no record agree with any record. Concatenation appends one set of observations to another, so the positions a record names would describe units from the other input; the result keeps each record without its positions, which is_refit(), the printed footer, and a later combine still read. The calibration record (ps_calib_meta) names the curve the scores were calibrated with rather than any position, the categorical attributes name exposure levels, and the exposure records describe the exposure rather than any unit, so all of them carry by the agreement rule above: two calibration records agree when they name the same method and smoothing.

c() of a single psw returns it unchanged, with every record. A combine through vctrs drops the positional records even when it is handed a single input: vctrs::vec_c(x), dplyr::bind_rows(df), vctrs::vec_rbind(df), and an ungrouped dplyr::reframe() all rebuild the column, so a later

is_unit_trimmed(), is_unit_truncated(), or is_unit_wt_truncated() on the result refuses it.

Subsetting with [preserves class and attributes for vector subscripts. Two kinds of attribute hold one value per observation. The records left by a trimmed or truncated propensity score (ps_trim_meta and ps_trunc_meta) or by truncating the weights themselves (psw_trunc_meta) name units by position, and [is handed the subscript, so it re-indexes each record onto the result: rev(), sort(), x[order(x)], and a shorter subset all return records naming the units at their new positions. Any other operation through vctrs is not handed a subscript, so it keeps the positions only where every unit stays at its place, as in elementwise arithmetic, and otherwise drops them, at any length: vctrs::vec_slice(), dplyr::arrange(), dplyr::filter(), unique(), rep_len(), and vctrs::vec_c() of a single input all return weights whose records name no units. A dropped record keeps its method, its bounds, and whether the model was refit, so is_refit(), the printed footer, and the bound comparison of a later combine still read it. Subassignment with [<- and is.na<- moves no unit and keeps the records whole; [<- refuses a value whose records describe a different modification, and accepts one with no record. A [<- of a value modified the same way that writes a missing weight leaves that unit listed as retained in the record, as is.na<- on a psw does; is.na<- on a ps_trim score instead removes the unit from its retained set. vctrs' own assignment, vctrs::vec_assign(), reaches the same restore a slice does and cannot be told apart from one, so it drops the positions, and so do the helpers built on it or on a combine, such as tidyr::replace_na(), dplyr::coalesce(), dplyr::if_else(), and dplyr::case_when().

A stabilization_score with more than one value, on the weights or on a component of a wt_joint() product, is in the order of the units too. [subsets it with the weights, elementwise arithmetic keeps it, and any other route through vctrs that brings back observations drops it with a warning of class propensity_stabilization_score_warning, since the score may no longer line up with the weights.

Dropping the stabilization_score warns, because the score was supplied by the user and the weights can be recomputed on the subset. Dropping a modification record is silent, because these records also travel by routes vctrs does not see, and a warning on the one route it controls would be neither complete nor about anything the user wrote. Subassignment that grows the vector carries a record across the length change, since [<- casts the replacement and then leaves base R to preserve the attributes; and model.frame() drops the NA-weighted rows from a weights column in C and re-attaches the original variable's attributes to the shortened result, so weights built on trimmed propensity scores come back out of every outcome model fit on them still carrying a record written for rows that are no longer there.

Honesty therefore lives at query time. is_unit_trimmed(), is_unit_truncated(), and is_unit_wt_truncated() answer by position, so each checks that the record covers the vector it is given and raises an error of class propensity_missing_meta_error when it does not, or when weights marked as modified carry no record at all, rather than name modified units at stale positions. is_refit() reads a single flag rather than a position, so it answers from any record present and refuses only when the record is absent entirely.

The result of any of these operations stays a psw and keeps every other attribute, including its stabilized, trimmed, truncated, and calibrated status, the calibration record, which names a calibration curve rather than any unit, the attributes describing a categorical exposure, which name the exposure levels rather than the units, and the exposure records, which describe the exposure rather than any unit, so all of them mean the same thing at any length.

Matrix or array subscripts intentionally drop the psw class and return a plain numeric vector via base R linear indexing; this is required so glm.fit()-style internal indexing works on psw-

weighted GLMs. Summary functions (sum(), mean(), etc.) return plain numeric values.

Value

- new_psw(), psw(), as_psw(): A psw vector.
- is_psw(), is_stabilized(): A single logical value.
- stabilization_score(): A numeric value or vector, or NULL if none was recorded or a per-observation score was dropped.

See Also

wt_ate(), wt_att(), wt_atu(), wt_atm(), wt_ato() for calculating propensity score weights (which return psw objects).

ps_trim(), ps_trunc(), and ps_calibrate() for modifying propensity scores before weight calculation.

Examples

```
# Create psw objects directly
w <- psw(c(1.2, 0.8, 1.5), estimand = "ate")
w

# Query metadata
is_psw(w)
estimand(w)
is_stabilized(w)

# Coerce a plain numeric vector
as_psw(c(1.0, 2.0), estimand = "att")

# Arithmetic preserves the psw class
w / sum(w)

# Combining: compatible metadata is preserved
x <- psw(c(1.2, 0.8), estimand = "ate")
y <- psw(c(1.1, 0.9), estimand = "ate")
c(x, y)

# Combining: incompatible metadata warns and returns numeric
x <- psw(c(1.2, 0.8), estimand = "ate")
y <- psw(c(1.1, 0.9), estimand = "att")
c(x, y)
```

`tidy.ipw` *Tidy an inverse probability weighted result*

Description

`tidy()` returns the estimates of an `ipw()` result as a tibble using the column names broom conventions use. For a binary or categorical exposure there is one row per exposure level, under the term `"mean"`, and then one row per effect measure per contrast of levels; for a continuous exposure there is one row per reported coefficient. The rows arrive in the order the result stores them and nothing is dropped.

Those columns are the ones the result's own coercion surface reports, so either reading is `as.data.frame()` read as a tibble rather than a second assembly of the same table. The values are the ones the result already holds, nothing being re-estimated: the confidence interval is the only thing rebuilt, and only when the requested `conf.level` differs from the level the result was fit at. The one thing the frame carries that the tibble does not is the covariance it attaches as an attribute, a tidied table being its columns.

`conf.int`, `conf.level`, and `exponentiate` are the arguments this method shares with that surface, so the surface validates them, in both readings and before either is assembled. A value it refuses is reported under the causalgenerics condition naming the argument at fault, which is what keeps one argument of one method from being well formed in one reading of a result and refused in the other.

A result reports its effects in one of two readings, and `tidy()` returns the one the result records unless `effects` names the other for the call. The marginal reading is the table of causal contrasts described above. The conditional reading is the outcome model's coefficient surface: one row per coefficient, with the standard errors of the block of the joint estimation that carries the uncertainty of having estimated the weights from the same data. The two readings return the same columns in the same order, with one exception: the `contrast` column belongs to the marginal reading. A binary and a categorical result both carry it there, naming the exposure level each mean row belongs to and the pair of levels each effect measure compares, and the conditional reading of either returns the same table one column narrower. A coefficient names neither a level nor a pair of them.

Usage

```
## S3 method for class 'ipw'
tidy(
  x,
  conf.int = FALSE,
  conf.level = 0.95,
  exponentiate = FALSE,
  ...,
  effects = NULL,
  parametric = NULL
)
```

Arguments

`x`	An `ipw` object, as returned by `ipw()`.

`conf.int` Logical. Should the confidence interval bounds be returned in the `conf.low` and `conf.high` columns? Defaults to `FALSE`.

`conf.level` The confidence level of the interval, a single number between 0 and 1. Defaults to `0.95`, which is this method's own default rather than the level `x` was fit at. When the two differ, the bounds are recomputed as the normal approximation `estimate +/- qnorm(1 - (1 - conf.level) / 2) * std.error`, which is the interval `ipw()` itself reports. Not used when `conf.int` is `FALSE`, though it must still be a valid level.

`exponentiate` Logical. Should the estimate and its bounds be exponentiated on the rows reported on the log scale? Defaults to `FALSE`. This behaves exactly as it does in `as.data.frame()`: the `log(rr)` and `log(or)` rows have their estimate and bounds exponentiated and their `term` relabeled to `rr` and `or`, while every other row is left alone. The standard error, the test statistic, and the p-value describe the log scale estimate and stay there.

The conditional reading has no rows labeled as ratios to pick out, so the link of the outcome model settles the question there: a `logit` link puts every coefficient on the log odds scale and a `log` link puts every coefficient on the log risk scale, and both are scales an exponential undoes. The estimate and, when they were asked for, the interval bounds are exponentiated, no term is relabeled, and the standard error, the test statistic, and the p-value describe the link scale and stay there. Every other link errors rather than exponentiating coefficients that are not on such a scale.

`...` These dots are for future extensions and must be empty.

`effects` The reading to report, either `"marginal"` or `"conditional"`. `NULL`, the default, reports the reading the result records; any other value overrides it for the one call and leaves the result as it is. The marginal reading reports the population-averaged causal contrasts; the conditional reading reports the outcome model's coefficient surface. `as_marginal()` and `as_conditional()` move a result between the two readings.

`"fixed"` is also accepted, and asks for the reading the result records, exactly as `NULL` does. It is the name mixed models give the effects that are not random, and `mice::pool()` asks every tidier it calls for it. A result reports one reading at a time and neither of them is random, so the name resolves to whichever one the result records. The accessors do not accept it: it is spelled here, where multiple imputation reaches the result, rather than in the surface underneath.

The covariance the conditional reading reports is the outcome block of the jointly estimated sandwich, which every route that stacks estimating equations attaches to the outcome model it stores: `se_method = "mestimation"` for a binary exposure, the categorical and joint routes, which run on M-estimation alone, and the continuous route under that method. A linearization or robust fit stacks no system and records no covariance of either kind, so its conditional reading errors rather than reporting the covariance the outcome model computed for itself, which treats the estimated weights as fixed.

`parametric` Accepted and ignored. `mice::pool()` passes `parametric = TRUE` to every tidier it calls, to ask the models it was written against for their parametric coefficient table rather than for a smooth term. An `ipw` result reports one table, and the inference it reports is already parametric, so there is nothing for the argument to

select and nothing is read from it. It is here so that a result can be pooled across multiply imputed datasets.

Value

A tibble with one row per estimate and the columns:

`term` The effect measure, such as `"mean"` for a counterfactual mean, or `"rd"`, `"log(rr)"`, `"log(or)"`, `"diff"`, `"slope"`, or `"coef"` for the measures built from them.

`contrast` What the row is about: the exposure level a `"mean"` row belongs to, such as `"0"` or `"b"`; the pair of levels an effect measure compares, such as `"1 vs 0"` or `"b vs a"`; or the coefficient name on a surface whose rows are named after their coefficients. Present only where a row reports one.

`estimate` The estimated effect.

`std.error` The standard error of the estimate.

`statistic` The z statistic, the estimate over its standard error.

`p.value` The two-sided p-value of that statistic.

`conf.low`, `conf.high` The interval bounds. Present only when `conf.int` is `TRUE`.

The conditional reading returns those columns in that order, holding one row per coefficient of the outcome model: `term` is the coefficient's name, `estimate` is the coefficient, `std.error` is the square root of the diagonal of the corrected covariance, `statistic` is the estimate over that standard error, and `p.value` is the two-sided normal p-value of the statistic. There is no `contrast` column, whatever the exposure, because a coefficient names neither an exposure level nor a pair of them, and the bounds are built at the level `conf.level` asks for, the stored ones belonging to the effects the marginal reading reports.

See Also

`ipw()` for the estimator, `glance()` for a one-row summary of the fit, `augment()` for its per-observation columns, `as.data.frame()` for the result's own columns, and `as_marginal()` and `as_conditional()` for the reading a result records. For results fitted to multiply imputed data, see the Multiple imputation section of `ipw()`, `pool_ipw()` for the pooling, and `tidy()` for what it returns.

Examples

```
set.seed(123)
n <- 200
x1 <- rnorm(n)
z <- rbinom(n, 1, plogis(0.5 * x1))
y <- rbinom(n, 1, plogis(-0.5 + 0.8 * z + 0.3 * x1))
dat <- data.frame(x1, z, y)

ps_mod <- glm(z ~ x1, data = dat, family = binomial())
wts <- wt_ate(ps_mod)
outcome_mod <- glm(y ~ z, data = dat, family = quasibinomial(), weights = wts)
result <- ipw(ps_mod, outcome_mod)

tidy(result)
```

```
# A 90% interval, with the ratio measures on their natural scale
tidy(result, conf.int = TRUE, conf.level = 0.9, exponentiate = TRUE)

# The outcome model's coefficients, with the standard errors the joint
# estimation of the weights and the outcome implies
tidy(result, conf.int = TRUE, effects = "conditional")
```

`tidy.ipw_pooled` *Tidy a pooled inverse probability weighted result*

Description

`tidy()` returns the estimates of a `pool_ipw()` result as a tibble using the column names broom conventions use. It holds the rows the results themselves report: for a binary or categorical exposure one row per exposure level, under the term `"mean"`, and then one row per effect measure per contrast of levels; for a continuous exposure one row per reported coefficient. The rows arrive in the order the pooled result stores them and nothing is dropped.

Those columns are the ones the pooled result's own coercion surface reports, so this method is that surface read as a tibble rather than a second assembly of the same table. The covariance of the effects the frame attaches is the one thing that does not travel: it is an attribute rather than a column, and a tidied table is its columns.

The pooling has already happened by the time this method sees the result. Nothing here re-pools and no argument changes an estimate: they say how the table already in the object is reported, which is which of the two readings it is, whether an interval comes with it, the level that interval is reported at, and the scale the ratio rows are put on.

A pooled result reports its effects in one of two readings, and `tidy()` returns the one the result records unless `effects` names the other for the call. `pool_ipw()` pools both readings of the results it is given, so naming one here reports a table the pooling already built rather than pooling again. The marginal reading is the table of pooled causal contrasts described above. The conditional reading is the outcome models' coefficient surface, pooled the same way: one row per coefficient, from the block of the joint estimation that carries the uncertainty of having estimated the weights from the same data. The two readings return the same columns in the same order, with one exception: the `contrast` column belongs to the marginal reading. A binary and a categorical pool both carry it there, naming the exposure level each mean row belongs to and the pair of levels each effect measure compares, and the conditional reading of either returns the same table one column narrower.

Usage

```
## S3 method for class 'ipw_pooled'
tidy(
  x,
  conf.int = FALSE,
  conf.level = NULL,
  exponentiate = FALSE,
  ...,
```

```
  effects = NULL
)
```

Arguments

x	An ipw_pooled object, as returned by pool_ipw().
conf.int	Logical. Should the confidence interval bounds be returned in the conf.low and conf.high columns? Defaults to FALSE.
conf.level	The confidence level of the interval. NULL, the default, reports the bounds at the level the pooled result stored, which is the level its own intervals were built at. A number between 0 and 1 rebuilds them at that level instead, from the estimate and its standard error referred to t on the pooled degrees of freedom. Not used when conf.int is FALSE, though it must still be a valid level.
exponentiate	Logical. Should the estimate and its bounds be exponentiated on the rows reported on the log scale? Defaults to FALSE. This behaves exactly as it does on the pooled result's own coercion surface: in the marginal reading the log(rr) and log(or) rows have their estimate and bounds exponentiated and their term relabeled to rr and or, and in the conditional reading the outcome model's link settles the scale for every row at once and no term is relabeled. In both readings the standard error, the test statistic, and the p-value describe the log scale and stay there, which is where the inference was done.
...	These dots are for future extensions and must be empty.
effects	The reading to report, either "marginal" or "conditional". NULL, the default, reports the reading the pooled result records; naming a reading reports that one for the one call and leaves the result as it is. The marginal reading reports the pooled causal contrasts; the conditional reading reports the pooled coefficients of the outcome models, which is the reading exponentiate above puts on the natural scale every row at once. as_marginal() and as_conditional() move a pooled result between the two readings for good rather than for a call. A reading the pooling could not compute is refused rather than reported under the other one's name. The conditional reading needs the covariance the joint estimation of the weights and the outcome implies, and a linearization fit stacks no such system and records no such block, so a set of those fits pools the marginal reading alone. A result pooled before both readings were kept holds only the one it was pooled in, and asking it for the other is refused the same way. "fixed" is not among the values this method takes, though tidy() on a single result accepts it. mice::pool() asks for that reading from the tidier of each imputation's own fit, which is where the name arrives and where the tolerance for it belongs; nothing tidies a pooled result on its behalf.

Value

A tibble with one row per pooled estimate and the columns:

term The effect measure, such as "mean" for a counterfactual mean, or "rd", "log(rr)", "log(or)", "diff", "slope", or "coef" for the measures built from them, or the coefficient name in the conditional reading.

`contrast` What the row is about: the exposure level a `"mean"` row belongs to, such as `"0"` or `"b"`; the pair of levels an effect measure compares, such as `"1 vs 0"` or `"b vs a"`; or the coefficient name on a surface whose rows are named after their coefficients. Present only in the marginal reading, and there only where a row reports one: the conditional reading names each coefficient in `term` and returns no `contrast` column.

`estimate` The pooled point estimate.

`std.error` The pooled standard error.

`statistic` The test statistic, the estimate over its standard error.

`df` The pooled degrees of freedom the statistic is referred to. These carry the Barnard-Rubin small-sample adjustment, so the statistic is referred to t on them rather than to the normal.

`p.value` The two-sided p-value of that statistic.

`conf.low`, `conf.high` The interval bounds. Present only when `conf.int` is `TRUE`.

See Also

`pool_ipw()` for the pooling, `glance()` for a one-row summary of the pooled fit, `as_marginal()` and `as_conditional()` for the reading a pooled result records, and the Multiple imputation section of `ipw()` for the workflow they belong to.

Examples

```
set.seed(2024)
n <- 150
x1 <- rnorm(n)
z <- rbinom(n, 1, plogis(0.3 * x1))
y <- rbinom(n, 1, plogis(-0.4 + 0.9 * z + 0.5 * x1))
dat <- data.frame(x1, z, y)
dat$x1[rbinom(n, 1, 0.2) == 1] <- NA

imp <- mice::mice(dat, m = 2, print = FALSE, seed = 1)
fits <- with(imp, {
  ps <- glm(z ~ x1, family = binomial())
  w <- wt_ate(ps)
  om <- glm(y ~ z, family = quasibinomial(), weights = w)
  ipw(ps, om)
})

tidy(pool_ipw(fits))

tidy(pool_ipw(fits), conf.int = TRUE, exponentiate = TRUE)

# The outcome models' coefficients, pooled the same way
tidy(pool_ipw(fits), effects = "conditional")
```

wt_ate *Calculate propensity score weights*

Description

Compute inverse probability weights for causal inference under different estimands. Each function targets a different population:

- `wt_ate()`: **Average Treatment Effect** – the full population.
- `wt_att()`: **Average Treatment Effect on the Treated** – the treated (focal) group.
- `wt_atu()`: **Average Treatment Effect on the Untreated** – the untreated (reference) group. `wt_atc()` is an alias.
- `wt_atm()`: **Average Treatment Effect for the Evenly Matchable** – units with the most overlap.
- `wt_ato()`: **Average Treatment Effect for the Overlap Population** – weights proportional to overlap.
- `wt_entropy()`: **Entropy-weighted Average Treatment Effect** – the entropy-tilted population.
- `wt_cens()`: **Inverse probability of censoring weights** – uses the same formula as `wt_ate()` but labels the estimand `"uncensored"`. Use these to adjust for censoring in survival analysis, not for treatment weighting. Remaining uncensored is a two-level event, so `wt_cens()` supports binary and continuous exposures only.

`.propensity` accepts a numeric vector of predicted probabilities, a `data.frame` or matrix of per-level probabilities, a fitted propensity score model, or a modified propensity score created by `ps_trim()`, `ps_trunc()`, `ps_refit()`, or `ps_calibrate()`.

All functions return a `psw` object – a numeric vector that tracks the estimand, stabilization status, and any trimming or truncation applied.

Usage

```
wt_ate(
  .propensity,
  .exposure,
  .sigma = NULL,
  exposure_type = c("auto", "binary", "categorical", "continuous"),
  .focal_level = NULL,
  .reference_level = NULL,
  stabilize = NULL,
  stabilization_score = NULL,
  ...,
  .density = "normal",
  numerator = c("marginal", "integrated"),
  .treated = NULL,
  .untreated = NULL
```

```
)

## S3 method for class 'data.frame'
wt_ate(
  .propensity,
  .exposure,
  .sigma = NULL,
  exposure_type = c("auto", "binary", "categorical", "continuous"),
  .focal_level = NULL,
  .reference_level = NULL,
  stabilize = NULL,
  stabilization_score = NULL,
  ...,
  .density = "normal",
  numerator = c("marginal", "integrated"),
  .propensity_col = NULL,
  .treated = NULL,
  .untreated = NULL
)

wt_att(
  .propensity,
  .exposure,
  exposure_type = c("auto", "binary", "categorical"),
  .focal_level = NULL,
  .reference_level = NULL,
  ...,
  .treated = NULL,
  .untreated = NULL
)

## S3 method for class 'data.frame'
wt_att(
  .propensity,
  .exposure,
  exposure_type = c("auto", "binary", "categorical"),
  .focal_level = NULL,
  .reference_level = NULL,
  ...,
  .propensity_col = NULL,
  .treated = NULL,
  .untreated = NULL
)

wt_atu(
  .propensity,
  .exposure,
  exposure_type = c("auto", "binary", "categorical"),
```

```
  .focal_level = NULL,
  .reference_level = NULL,
  ...,
  .treated = NULL,
  .untreated = NULL
)

## S3 method for class 'data.frame'
wt_atu(
  .propensity,
  .exposure,
  exposure_type = c("auto", "binary", "categorical"),
  .focal_level = NULL,
  .reference_level = NULL,
  ...,
  .propensity_col = NULL,
  .treated = NULL,
  .untreated = NULL
)

wt_atm(
  .propensity,
  .exposure,
  exposure_type = c("auto", "binary", "categorical"),
  .focal_level = NULL,
  .reference_level = NULL,
  ...,
  .treated = NULL,
  .untreated = NULL
)

## S3 method for class 'data.frame'
wt_atm(
  .propensity,
  .exposure,
  exposure_type = c("auto", "binary", "categorical"),
  .focal_level = NULL,
  .reference_level = NULL,
  ...,
  .propensity_col = NULL,
  .treated = NULL,
  .untreated = NULL
)

wt_ato(
  .propensity,
  .exposure,
  exposure_type = c("auto", "binary", "categorical"),
```

```
  .focal_level = NULL,
  .reference_level = NULL,
  ...,
  .treated = NULL,
  .untreated = NULL
)

## S3 method for class 'data.frame'
wt_ato(
  .propensity,
  .exposure,
  exposure_type = c("auto", "binary", "categorical"),
  .focal_level = NULL,
  .reference_level = NULL,
  ...,
  .propensity_col = NULL,
  .treated = NULL,
  .untreated = NULL
)

wt_entropy(
  .propensity,
  .exposure,
  exposure_type = c("auto", "binary", "categorical"),
  .focal_level = NULL,
  .reference_level = NULL,
  ...,
  .treated = NULL,
  .untreated = NULL
)

## S3 method for class 'data.frame'
wt_entropy(
  .propensity,
  .exposure,
  exposure_type = c("auto", "binary", "categorical"),
  .focal_level = NULL,
  .reference_level = NULL,
  ...,
  .propensity_col = NULL,
  .treated = NULL,
  .untreated = NULL
)

wt_atc(
  .propensity,
  .exposure,
  exposure_type = c("auto", "binary", "categorical"),
```

```
  .focal_level = NULL,
  .reference_level = NULL,
  ...,
  .treated = NULL,
  .untreated = NULL
)

wt_cens(
  .propensity,
  .exposure,
  .sigma = NULL,
  exposure_type = c("auto", "binary", "continuous"),
  .focal_level = NULL,
  .reference_level = NULL,
  stabilize = NULL,
  stabilization_score = NULL,
  ...,
  .density = "normal",
  numerator = c("marginal", "integrated"),
  .treated = NULL,
  .untreated = NULL
)

## S3 method for class 'data.frame'
wt_cens(
  .propensity,
  .exposure,
  .sigma = NULL,
  exposure_type = c("auto", "binary", "continuous"),
  .focal_level = NULL,
  .reference_level = NULL,
  stabilize = NULL,
  stabilization_score = NULL,
  ...,
  .density = "normal",
  numerator = c("marginal", "integrated"),
  .propensity_col = NULL,
  .treated = NULL,
  .untreated = NULL
)
```

Arguments

`.propensity` Propensity scores in one of several forms:

- A **numeric vector** of predicted probabilities (binary/continuous).
- A **data frame** or **matrix** with one column per exposure level. Both shapes are read the same way, for categorical exposures and for binary ones alike, including the choice of column described under `.propensity_col`. That

argument is itself a formal of the data frame methods only, so selecting a column by name means passing a data frame. A frame of probabilities predicted from a fitted tidymodels classification model has its columns named `.pred_<level>`, and the level a column holds is read from the part of the name after that prefix. A frame holding a `.pred_class` column, which the same models return when no prediction type is named, carries predicted levels rather than probabilities and is refused with an error of class `propensity_df_class_column_error`.

- A fitted **propensity score model** – fitted values are extracted automatically. For a binary exposure that is a `glm()` fit with `binomial()` or `quasibinomial()`. For a continuous exposure it is a model of the conditional mean: `lm()`, a `glm()` fit with `gaussian()` under any of its links, an `mgcv::gam()` fit with `gaussian()`, or a `MASS::rlm()`. For a categorical exposure it is an `nnet::multinom()`, whose fitted values hold the probability of every level; a multinomial fit of two levels reports a single probability and is read as a model of a binary exposure. Handing one exposure type a model of another, or a family whose spread changes with its fitted values, is an error of class `propensity_model_family_error`. The one exception is the multinomial fit, which offers no continuous type at all, so naming one for it is an error of class `causalgenerics_unsupported_exposure_type`. See **Exposure types** and **Continuous exposures** in Details.
- A modified propensity score created by `ps_trim()`, `ps_trunc()`, `ps_refit()`, or `ps_calibrate()`.

For a binary exposure, `.propensity` is the probability of the focal level, so a numeric vector must be supplied on the scale of whichever level `.focal_level` or `.reference_level` resolves to. The model methods derive it from the fitted values, which give the probability of the response's second level, and subtract them from one when the resolved focal level is the first level instead. A data frame or matrix reduces to a single column, read on the same scale; see `.propensity_col` for how that column is chosen. A matrix reduces by those same rules, but `.propensity_col` belongs to the data frame methods alone, so a matrix whose column you want to name has to be converted with `as.data.frame()` first.

For a continuous exposure, `.propensity` is the conditional mean of the exposure. A data frame of a single column is read as that mean; a data frame of several columns is refused with an error of class `propensity_df_ambiguous_column_error` unless `.propensity_col` names the column to read, since a continuous exposure has no levels the columns could be matched against.

`.exposure` The exposure (treatment) variable. For binary exposures, a numeric 0/1 vector, logical, or two-level factor. For categorical exposures, a factor or character vector. For continuous exposures, a numeric vector. Optional when `.propensity` is a fitted model, in which case it is the model's response. Missing values are not counted as a level of their own, and are carried through to the weights as missing.

`.sigma` The residual spread of the conditional density for continuous exposures: a single standard deviation applied to every unit, or one for each observation (e.g., `influence(model)$sigma`). Optional: with none supplied, including when

.propensity is a fitted model, the conditional density uses the spread its family estimates from the residuals of .exposure around .propensity, the pooled root mean square unless the family estimates a scale of its own.

The two shapes differ downstream. A single spread is a constant the weights can be rebuilt from, so it is recorded in density_meta() as sigma_value and ipw() reads it as a spread held fixed, propagating none of its uncertainty. A spread supplied for each observation is a function of the data that nothing rebuilds, so the record holds where it came from and nothing more, and ipw() refuses such weights; see **Continuous exposures** in Details.

A dose model trimmed with ps_trim() holds the spread its trim was read at, and weights built from it are read at that spread, which density_meta() records as sigma_value as well. A .sigma supplied with such a trim would be a second spread for the same density and is refused with an error of class propensity_sigma_error.

Must be numeric, and applies only to continuous exposures. .sigma sits in the third position, which is where a value meant for exposure_type arrives when it is supplied without a name, so anything else is refused with an error of class propensity_sigma_error. A spread you supply and a density built with sigma_method = "mle", which estimates one from the residuals, are two instructions about the same quantity and are refused together with an error of class propensity_density_error.

exposure_type — Type of exposure: "auto" (default), "binary", "categorical", or "continuous". "auto" detects the type from .exposure. Not every weight function answers every type; see **Exposure types** in Details for which does which. Naming a type a function does not support, or supplying an exposure detection reads as one, is an error of class causalgenerics_unsupported_exposure_type.

.focal_level — The value of .exposure representing the focal (treated) group. Every binary coding honors it: 0/1 numeric, logical, two-level factor, and two-level character exposures are all coded with the named level as focal, and a level the exposure never takes is an error. With no level named, a binary exposure defaults to its higher level, which is 1 for a 0/1 exposure, TRUE for a logical one, and the second of the two levels a factor or character exposure takes. Levels a factor declares but never takes are not candidates. Naming any other level reverses the coding, so .propensity must then hold the probability of the named level. Required for wt_att() and wt_atu() with categorical exposures.

.reference_level

The value of .exposure representing the reference (control) group. For a binary exposure, naming it makes the exposure's other level focal, with the same consequence for .propensity, and a level the exposure never takes is an error. Automatically detected if not supplied.

stabilize — Whether to multiply the weights by an estimate of the marginal probability of the exposure a unit took (binary or categorical) or of its density (continuous), and what that estimate is. It takes one of three forms:

- A logical. TRUE and FALSE ask for stabilization or its absence outright, and an unstabilized continuous exposure reports that its weights are not the recommended ones. NULL, the default, reads the answer from the exposure

type: a continuous exposure is stabilized, and a binary or categorical exposure is not.

- A fitted model of the exposure, which stabilizes the weights on what that model estimates rather than on the marginal probability or density of the exposure.
  For a binary exposure the model is a `binomial()` fit and the numerator is its fitted probability of the level each unit took, so the weights are $P(A = a_i \mid V_i)/f(a_i \mid X_i)$ for the variables V the numerator model reads. For a continuous exposure the numerator is the family `.density` names, read at the model's fitted mean and the spread its own residuals leave under that family, so the weights are $f(A \mid V)/f(A \mid X)$. For a categorical exposure the model is an `nnet::multinom()` fit and the numerator is its fitted probability of the level each unit took, read from the column named for that level, so the weights are $P(Z = z_i \mid V_i)/f(z_i \mid X_i)$. The multinomial fit has to report a probability for every level the exposure has, matched by name rather than by position.
  The model is recorded on the result, where `numerator_model()` reads it back, and `ipw()` estimates it alongside everything else so that the standard errors account for it having been fitted, whichever type the exposure is: a categorical exposure's multinomial score is stacked there the way a binary exposure's binomial one is. Conditioning the numerator on V changes what is estimated unless the model the estimates are read from also reads V; see **Stabilization** in Details.
  Any `lm()`, or anything built on one, is read this way, as is an `nnet::multinom()`. The model is held to the exposure: a fit whose spread changes with its fitted values, or a model of a conditional mean where a probability is needed, is refused with `propensity_model_family_error`, which is also what a model of a dose handed to a binary exposure, a `binomial()` model handed to a dose, a multinomial fit handed either of them, and a fit reporting some other set of levels than the exposure's are refused with. A model supplied together with `stabilization_score` or `numerator = "integrated"` is `propensity_numerator_error`. So is a model fit with case `weights`, which estimates the numerator in a reweighted sample rather than in the one the weights are being built for. A model with a fitted value for some other set of observations is `propensity_length_error`. That check counts the fitted values rather than reading which rows they came from, so a model fit to another dataset of the same length passes it: fit the numerator model on the data the weights are being built for.

Anything else is refused with an error of class `propensity_stabilize_error`. Stabilization is only supported by `wt_ate()` and `wt_cens()`, since the other estimands' weights already carry a tilting function where the numerator would go and stabilizing them would move the population they target rather than their variance. See **Stabilization** in Details.

`stabilization_score`

Optional stabilization multiplier to use instead of the default described under **Stabilization**: the marginal mean of `.exposure`, or its marginal normal density for a continuous exposure. Either a single value applied to every weight or a numeric vector holding one value per observation, which is multiplied into the

weights observation by observation. Every value must be positive and finite, and any other length or value is refused with an error of class `propensity_stabilization_score_error`. A per-observation score is recorded on the result and is dropped, with a warning, by any operation that changes the length of the weight vector, since it cannot be re-indexed for the new length. Ignored when stabilization is off, whether because `stabilize` is `FALSE` or because the exposure type resolves the default that way.

`...` These dots are for future extensions and must be empty.

`.density` The family of the conditional density for continuous exposures, described under **Continuous exposures** in Details. One of the strings `"normal"` (the default), `"laplace"`, or `"kernel"`; a specification built by `dens_normal()`, `dens_laplace()`, `dens_t()`, `dens_kernel()`, or `dens_fn()`; or a bare function of the standardized residual, which is wrapped with `dens_fn()`.

A function is called on the whole vector of standardized residuals at once rather than on one residual at a time, and is called a second time on the standardized exposure when the marginal numerator stabilizes the weights. Each call must return one finite, non-negative value for each value it was given, not every one of them zero. Anything else is refused with an error of class `propensity_density_error`, which is also what a family other than the default gets for a binary or categorical exposure, whose weights are not a ratio of densities. The default is accepted for every exposure type and ignored outside a continuous one.

`.density` sits after `...` and so can only be supplied by name. The family the weights were built from is recorded on the result and read back with `density_meta()`.

Weights built from a dose model trimmed with `ps_trim()` are read under the family the trim recorded, whatever the default says. A `.density` that names another family, other parameters, or another `sigma_method` is refused with an error of class `propensity_density_error`.

`numerator` How the marginal density that stabilizes a continuous exposure's weights is obtained, described under **Stabilization** in Details. Either `"marginal"`, the default, which reads the family `.density` names at the mean of `.exposure` and the spread the same estimator gives for it, both over the rows the propensity model kept, or `"integrated"`, which averages the conditional density over the units on a grid spanning `.exposure` and interpolates the result back to each observed exposure.

`"integrated"` describes a continuous exposure alone, and needs a conditional density to marginalize: it is refused with an error of class `propensity_numerator_error` for a binary or categorical exposure, with `stabilize = FALSE`, with a `stabilization_score`, and with any `.sigma`. The default is accepted for every exposure type and ignored outside a continuous one.

`numerator` sits after `...` and so can only be supplied by name. The numerator the weights were built from is recorded on the result and read back with `density_meta()`.

`.treated` **[Deprecated]** Use `.focal_level` instead.

`.untreated` **[Deprecated]** Use `.reference_level` instead.

`.propensity_col`

Column to use when `.propensity` is a data frame with a binary or continuous exposure. Accepts a column name (quoted or unquoted) or numeric index. For

a binary exposure the selected column is read as the probability of the resolved focal level; for a continuous one it is read as the conditional mean of the exposure.

With no column named and a binary exposure, the exposure's levels drive the choice. When the data frame has a column named for every level of `.exposure`, the column named for the resolved focal level is used, wherever it sits in the frame. Columns named `.pred_<level>`, the shape a fitted tidymodels classification model predicts probabilities in, are matched by the level after the prefix. Otherwise the second column is used, or the only column when the frame has just one. Falling back to position after `.focal_level` or `.reference_level` was supplied warns and reports the column used, since the named level could not be matched to a column; the fallback is silent when no level was named and when the frame has a single column.

With no column named and a continuous exposure, a frame of a single column is read as the conditional mean, and a frame of several columns is refused with an error of class `propensity_df_ambiguous_column_error`. There are no levels to read the names against, so nothing distinguishes one column of conditional means from another and there is no position worth trusting.

Ignored for categorical exposures, where all columns are used.

Details

Exposure types:

All weight functions support binary exposures. `wt_ate()` and `wt_cens()` also support continuous exposures. All except `wt_cens()` support categorical exposures; naming a categorical exposure to `wt_cens()`, or handing it one detection reads as categorical, is an error of class `causalgenerics_unsupported_exposure_type`.

- **Binary**: `.exposure` is a two-level vector (e.g., 0/1, logical, or a two-level factor). `.propensity` is a numeric vector of P(treatment | X), or a matrix or data frame holding it in one of its columns.
- **Categorical**: `.exposure` is a factor or character vector with 3+ levels. `.propensity` must be a matrix or data frame with one column per level, where rows sum to 1. Every weight function but `wt_cens()` also takes a fitted `nnet::multinom()` and reads those columns off its fitted values, matching them to the levels of `.exposure` by name. A multinomial fit of only two levels reports a single probability and is read as a model of a binary exposure. Those per-level probabilities are the generalized propensity score of a multi-valued exposure (Imbens, 2000), and weighting by them is the multi-arm inverse probability estimator (Feng et al., 2012).
- **Continuous**: `.exposure` is a numeric vector. `.propensity` is a vector of conditional means (fitted values). Weights are a ratio of densities, whose family is chosen by `.density`, and are stabilized unless `stabilize = FALSE` asks otherwise.
- **Auto** (default): Detects the exposure type from `.exposure`.

Stabilization:

Stabilization multiplies the base weight by an estimate of P(A) (binary) or f_A(A) (continuous), or by an estimate of either conditional on variables a model of the exposure reads, reducing variance. When no `stabilization_score` is supplied, that estimate is the marginal mean of `.exposure`

for a binary or categorical exposure, and for a continuous exposure the marginal density of the family `.density` names, evaluated at the population mean of `.exposure` and at the spread the same estimator gives for it. Stabilization is supported for ATE and censoring weights (`wt_ate()` and `wt_cens()`) alone.

Whether it is applied is read from the exposure type unless `stabilize` says so outright. A continuous exposure is stabilized, since the unstabilized ratio carries the exposure's own units and is not a form of the weights the package recommends; a binary or categorical exposure is left unstabilized, and `stabilize = TRUE` asks for it there.

For a continuous exposure, `numerator` chooses how that marginal density is arrived at. `"marginal"`, the default, reads the family `.density` names at the mean of `.exposure` and the spread the same estimator gives for it, both over the rows the propensity model kept, which are the rows the weights describe: a model that dropped rows leaves `.propensity` missing at them, and those rows carry no weight. Those two moments are parameters of the weights, and `ipw()` estimates them alongside the rest of its parameter vector, so the standard errors account for the numerator having been estimated. `"integrated"` marginalizes the conditional density numerically instead: it averages $f_{A|X}(t \mid X_i)$ over the units at each of 50 points spanning `.exposure`, then interpolates that average back to each observed exposure with a cubic spline. It estimates no parameters of its own, and it is what WeightIt has done since version 2.0.0.

The two agree whenever the family is normal and the fitted conditional means are themselves normal, since an average of normal densities centered on normally distributed means is again a normal density. They separate as those means grow skewed or bimodal, or as the family's tails grow heavier than the normal's.

The marginal numerator is the default because it is the published convention (Robins et al., 2000; Naimi et al., 2014) and because it held up: in a simulation study run for this package, over linear, skewed, heteroskedastic, heavy-tailed, and bimodal designs, marginalizing never improved bias, root mean squared error, or weighted covariate balance by more than Monte Carlo noise, including in the scenarios drawn to favor it, and in the heavy-tailed cells it was worse: it inflated the root mean squared error, and in some replicates the interpolated density came back negative. The integrated numerator is here for agreement with WeightIt, and for a conditional density whose marginal has no closed form in the same family. Being read off an interpolation rather than a formula, it can dip below zero where the density on the grid comes close to it, which is refused with an error of class `propensity_density_error` rather than turned into a negative weight.

The default numerator conditions on nothing, and `stabilization_score` replaces it with one of your own, which leaves `numerator` nothing to choose: the two cannot be supplied together. The case that pays is a numerator conditioning on a variable the model the estimates are read from also reads, such as an effect modifier. Fit that numerator and evaluate it at the exposure each unit actually took, which is the shape the default already has:

```
num <- glm(A ~ V, data = dat, family = binomial())
p <- fitted(num)
wt_ate(ps_mod, stabilize = TRUE,
       stabilization_score = ifelse(dat$A == 1, p, 1 - p))
```

Passing `fitted(num)` on its own is P(A = 1 | V) for every unit, which is not the probability of the exposure an untreated unit took, so the untreated weights would carry the wrong numerator. Whether a numerator may condition on `V` at all is a question about the reported model rather than about the weights; see **Effect modification** in `ipw()`.

There is a third way of writing the same thing: pass the fitted numerator model to `stabilize` itself, and what it estimates becomes the numerator.

```
num <- glm(A ~ V, data = dat, family = binomial())
wt_ate(ps_mod, stabilize = num)

num <- lm(A ~ V, data = dat)
wt_ate(dose_mod, stabilize = num)

num <- nnet::multinom(Z ~ V, data = dat, trace = FALSE)
wt_ate(ps_mat, Z, exposure_type = "categorical", stabilize = num)
```

For a binary exposure the weights are then $P(A = a_i \mid V_i)/f(a_i \mid X_i)$: the model's fitted probability read at the level each unit actually took, which is p_i for a unit with $A_i = 1$ and $1 - p_i$ for a unit with $A_i = 0$, over the same denominator the unstabilized weights divide by. Writing it as the model rather than as the numbers is what keeps the untreated units from carrying $P(A = 1 \mid V)$, which is the mistake the `stabilization_score` form has to be written around by hand.

For a continuous exposure they are $f(A \mid V)/f(A \mid X)$: the family `.density` names, read at the numerator model's fitted mean and the spread its own residuals leave under that family, over the same family read at the propensity score model's.

For a categorical exposure they are $P(Z = z_i \mid V_i)/f(z_i \mid X_i)$: the multinomial model's fitted probability of the level each unit took, gathered from the column named for that level, over the same denominator again. The fit has to report a probability for every level the exposure takes, and its columns are read by name rather than by position, so a fit that declares those levels in another order builds the same weights here. `ipw()` asks for the propensity score model's own level order in addition, since the block it stacks reads the coefficients positionally. Fitting a multinomial model of the exposure on the stabilization terms is what the ipw package (van der Wal & Geskus, 2011) does for a multi-valued exposure as well.

The same contract governs all three, and it is a statement about the model the estimates will be read from rather than about the weights. A numerator conditioning on `V` builds a pseudo-population in which `V` still predicts the exposure, so confounding by `V` remains there and what is estimated is the effect conditional on `V` rather than the marginal one. The outcome model, or the marginal structural model, must therefore include `V` (Robins et al., 2000; Cole & Hernán, 2008; Hernán & Robins, 2020, Chapter 12). The limiting case says it from the other end: with `V` the whole set of confounders the numerator is the denominator, the weights collapse toward 1, and the weighting has nothing left to do that the model is not already doing. The case that pays is a numerator conditioning on an effect modifier the reported model reads anyway.

Handing the model over rather than the numbers it evaluates to is what lets `ipw()` estimate it: the model's own estimating equations join the stacked system, so the standard errors account for the numerator having been fitted, where a `stabilization_score` is carried as a known constant. That accounting has something to say only where the reported model is not saturated in the variables the numerator reads. A numerator of a binary or categorical exposure on an effect modifier is constant within each cell of the modifier and the exposure, so a model saturated in those cells fits the same coefficients with the numerator and without it, and the standard errors do not move either.

What a numerator model is held to is its shape rather than its provenance: its class and family, the levels it declares for a categorical exposure, and one fitted value per observation. A fit to a different dataset of the same length passes every one of those and multiplies in a numerator belonging to other rows, without a word, so fit the numerator model on the data the weights are being built for.

Only `wt_ate()` and `wt_cens()` stabilize at all, which follows from what the other estimands are rather than from anything missing here. An `att`, `atu`, `atm`, `ato`, or entropy weight already carries a tilting function where a numerator would go, so an estimate of $P(A)$ or $P(A \mid V)$ put there would move the population the weights target rather than how variable they are. Asking for it is an error.

`numerator_model()` reads the model back off the weights.

Handling extreme weights:

Extreme weights signal positivity violations, poor model fit, or limited overlap. You can address them by:

- Choosing an overlap-focused estimand (`wt_ato()`, `wt_atm()`, `wt_entropy()`), which down-weight units in regions of poor overlap.
- Trimming (`ps_trim()`) or truncating (`ps_trunc()`) propensity scores before computing weights.
- Calibrating weights with `ps_calibrate()`.
- Stabilizing ATE and censoring weights: a continuous exposure is stabilized by default, and a binary or categorical one is with `stabilize = TRUE`.
- Choosing a heavier-tailed `.density` for a continuous exposure, which holds down the weight of a unit whose exposure the propensity score model fits poorly.

See the halfmoon package for weight diagnostics and visualization.

Propensity scores at 0 and 1:

Propensity scores must lie strictly inside (0, 1), since a score at either endpoint leaves the weight undefined. A score of exactly 0 or 1 is refused with an error of class `propensity_range_error`. For a categorical exposure that rule reads every cell of the matrix rather than only the column holding each unit's observed level, which is the same open interval the binary path enforces on the single score and its complement.

`ipw()` applies a narrower rule to the propensity scores it rebuilds from its own propensity score model. For a categorical exposure it refuses only a probability of exactly zero at the level a unit was actually assigned, since that alone leaves the unit's own weight undefined, and a column for a level the unit was not assigned may reach zero without the fit being refused. The difference is deliberate: the weight functions can refuse a matrix that holds nothing `ipw()` would have objected to. The weights still have to be built before `ipw()` can be reached, so the refusal here is the one to resolve.

`nnet::multinom()` reaches the endpoints readily. Under separation the softmax puts the probability at a unit's assigned level at exactly 1 in double precision, and the columns for the other levels can underflow to exactly 0. `ps_trunc()` is the repair for such a matrix: its categorical matrix method reads the closed interval $[0, 1]$, so it accepts a cell of exactly 0 or 1, pins every score below its threshold up to it, and renormalizes each row to sum to 1. What it returns lies strictly inside (0, 1) and weights like any other truncated matrix. That relaxation belongs to the categorical matrix method alone: `ps_trim()` sets extreme scores to missing and gains nothing from an endpoint, so it holds the open interval, and so does the binary vector path of `ps_trunc()`. If the bounds a truncation computes for itself leave a score at an endpoint anyway, which `method = "pctl"` can do when the lower quantile of the scores is itself zero, the call is refused and names the bound it computed; supply `lower` and `upper` explicitly instead. Refitting the propensity score model so that it does not separate remains the alternative.

Weight formulas:

Binary exposures:
For binary treatments ($A \in \{0, 1\}$), with propensity score $e(X) = P(A = 1 \mid X)$:

- **ATE**: $w = \frac{A}{e(X)} + \frac{1-A}{1-e(X)}$
- **ATT**: $w = A + \frac{(1-A)\cdot e(X)}{1-e(X)}$
- **ATU**: $w = \frac{A\cdot(1-e(X))}{e(X)} + (1 - A)$
- **ATM**: $w = \frac{\min(e(X),1-e(X))}{A\cdot e(X)+(1-A)\cdot(1-e(X))}$
- **ATO**: $w = A \cdot (1 - e(X)) + (1 - A) \cdot e(X)$
- **Entropy**: $w = \frac{h(e(X))}{A\cdot e(X)+(1-A)\cdot(1-e(X))}$, where $h(e) = -[e \log(e) + (1 - e) \log(1 - e)]$

The entropy weight tilts the propensity score by $h(e)$, the entropy of the score itself, in the sense of Zhou, Matsouaka, and Thomas (2020). It is not entropy balancing, which solves for weights that satisfy exact covariate moment constraints rather than tilting a fitted propensity score.

Continuous exposures:
Weights use the density ratio $w = f_A(A)/f_{A|X}(A \mid X)$, where f_A is the marginal density of the exposure and $f_{A|X}$ is the conditional density the propensity model estimates. Only `wt_ate()` and `wt_cens()` support continuous exposures.
Both densities are read on a standardized residual. The conditional density is evaluated at $z_i = (A_i - \hat{A}_i)/\sigma$, where $\hat{A}_i$ is the fitted conditional mean in `.propensity` and σ is the residual spread; the marginal density is evaluated at $z_i^A = (A_i - \bar{A})/s_A$, where $\bar{A}$ is the mean of `.exposure` over the rows the propensity model kept and s_A is the spread the same estimator gives for it, the standard deviation unless the family estimates a scale of its own. Both moments are read over those rows whether the spread was pooled from the model's own residuals or supplied through `.sigma`, so the two halves of the ratio describe one set of units. Each density is then divided by the spread that standardized it, the Jacobian of that change of variable, which returns both to the exposure's own units so that each integrates to one:

$$w_i = \frac{g(z_i^A)/s_A}{g(z_i)/\sigma}$$

`.density` chooses the family g:

`.density`	$g(z)$
`"normal"`, `dens_normal()`	the standard normal density, the default
`"laplace"`, `dens_laplace()`	$\exp(-\lvert z\rvert)/2$
`dens_t()`	Student's t with `df` degrees of freedom
`"kernel"`, `dens_kernel()`	a kernel density estimate of the standardized residuals
`dens_fn()`, or a bare function	a density you write yourself

The numerator and the denominator take the same family, so the choice describes the shape of the residuals rather than the estimand. A heavier tail than the normal's, such as `dens_t()` or `"laplace"`, holds down the weight of a unit whose exposure the model fits poorly, and `"kernel"` assumes no shape at all, at the cost of a density that is not a smooth function of the model's parameters. A `stabilization_score` replaces the numerator outright and `stabilize = FALSE` leaves it out, so under either the family describes the denominator alone. Under `numerator = "integrated"` it describes both again, the numerator there being the same conditional density averaged over the units rather than the same family read at the exposure's own moments; see **Stabilization**.

A unit whose exposure or fitted conditional mean is missing has no standardized residual, and so no weight: the density is asked only about the units that have one, and a kernel is fit on those. A propensity score `ps_trim()` set aside is the ordinary way this arises.

The conditional means in `.propensity` can be read from the model that fit them. `wt_ate()` and `wt_cens()` take a fitted `lm()`, a `glm()` fit with `gaussian()` under any of its links, an `mgcv::gam()` fit with `gaussian()`, and a `MASS::rlm()`, and read the exposure from the model's response when `.exposure` is not supplied. Each of these reports its conditional mean on the scale of the exposure, so a log, inverse, or square root link never has to be undone. Any other subclass of `lm()` is read as an `lm`, because the fitted conditional means are all the weights ask of a model. `ipw()` names the classes it supports instead of accepting a subclass on those terms: it stacks each model's own estimating equations, which it can only write for a score it knows.

A family whose spread changes with its fitted values, such as `poisson()` or `quasi(variance = "mu")`, describes a different density for every unit, which a single spread cannot stand in for, and is refused with an error of class `propensity_model_family_error`. `quasi(variance = "constant")` is the gaussian variance under another name and is accepted. A model of a binary exposure is held to the same rule from the other side: only `binomial()` and `quasibinomial()` fit the probability those weights divide by, so a model of a conditional mean is refused there whatever its fitted values happen to fall on, a linear probability model included.

σ is the spread the family `.density` names asks for, read from the residuals of the propensity score model. For a family with no scale estimator of its own that is the pooled root mean square $\sqrt{\text{mean}((A - \hat{A})^2)}$; for `dens_t()` and `dens_laplace()` it is the maximum likelihood scale of the family itself, which both take by default. `sigma_method = "rms"` asks either of them for the root mean square instead; see **The spread of the conditional density** in `dens_t()`'s documentation. `.sigma` supplies a spread of your own, a single standard deviation or one for each unit, which the model methods do not do on their own, and is refused alongside a family that estimates its own scale, being a second instruction about the same quantity. A family that estimates its own scale spreads both densities of the ratio, so the marginal density is read at that scale too and the two halves are densities of one width rather than two.

Every model class is spread that same way, `MASS::rlm()` included. `rlm` reports a robust scale estimate of its own in `fit$s`, which resists the extreme residuals rather than pooling all of them; pass `.sigma = fit$s` to be spread by it instead. That is rarely what you want on the data a robust fit is used for: the scale resists the very residuals the conditional density is meant to spread over, so the ratio can concentrate on a handful of units. `ipw()` stacks a robust fit's own score with that scale held fixed as a known constant, and reads `.sigma = fit$s` as a fixed spread in the same way, propagating the uncertainty of neither.

`ipw()` models the conditional density with a single conditional variance, estimated jointly with the rest of the parameter vector, by the moment the pooled spread is the root of or, for a density built with `sigma_method = "mle"`, by the score of the family it names. A single `.sigma` is taken instead as a known constant: the weights are rebuilt at the number that was supplied, and the stacked system carries none of that number's uncertainty. An observation-level `.sigma` is a different function of the data with no counterpart in that estimating equation, and is refused before anything is solved, with an error of class `propensity_ipw_sigma_error`. Build weights with the pooled default, or with one number, when the outcome model is headed for `ipw()`.

A conditional density of exactly zero at a unit's own exposure is the continuous counterpart of a propensity score of exactly 0 or 1: the weight it leaves is infinite, and every estimate built from it is undefined. It is refused with an error of class `propensity_density_error` naming the observations it happened at. A light-tailed family at an outlying residual and a kernel read past the range it was fit on both reach it, so the remedy is a family with heavier tails, such as

dens_t() or dens_kernel(). A zero in the numerator is not refused: a marginal density of zero is a weight of zero, which is a weight like any other.
Stabilized weights from the normal family have a finite second moment only while the marginal variance of the exposure stays below twice the variance of the conditional density, $\mathrm{Var}(A) < 2\sigma^2$. Past that boundary the weights have no finite variance, and estimates built from them are erratic however large the sample is. Reaching it is reported with a message of class propensity_density_variance_messa the weights are still returned, and the report is informational because a well-fitting model reaches the boundary in ordinary use. Explaining more of the exposure lowers σ^2 while the marginal variance is fixed by the data, so a better fit tightens the boundary rather than escaping it; a heavier-tailed family sits at a different one, and an unstabilized weight has no boundary at all. The boundary is a property of the normal family read against the exposure's own marginal density, so nothing is reported for an unstabilized weight, a supplied stabilization_score, numerator = "integrated", another family, or an observation-level .sigma.

Categorical exposures:
For K-level treatments, weights take the tilting-function form $w_i = h(\mathbf{e}_i)/e_{i,Z_i}$, where e_{i,Z_i} is the propensity for unit i's observed level and $h(\cdot)$ depends on the estimand:

- **ATE**: $h(\mathbf{e}) = 1$
- **ATT**: $h(\mathbf{e}) = e_{\text{focal}}$
- **ATU**: $h(\mathbf{e}) = 1 - e_{\text{focal}}$
- **ATM**: $h(\mathbf{e}) = \min(e_1, \ldots, e_K)$
- **ATO**: $h(\mathbf{e}) = \left(\sum_k 1/e_k\right)^{-1}$
- **Entropy**: $h(\mathbf{e}) = -\sum_k e_k \log(e_k)$

Value

A psw vector (a double vector with class psw) carrying these attributes:

- estimand: character, e.g. "ate", "att", "uncensored".
- stabilized: logical, whether stabilization was applied.
- stabilization_score: double, the user-supplied stabilization score (one value, or one per observation), or NULL if none was supplied.
- trimmed: logical, whether the propensity scores were trimmed.
- truncated: logical, whether the propensity scores were truncated.
- calibrated: logical, whether the propensity scores were calibrated.
- exposure_type: character, the type of exposure the weights were built for, read back with exposure_type().
- density_meta: for a continuous exposure, whose weights are a ratio of densities, the record of that ratio: the density family, the numerator that stabilized the weights, and where the residual spread came from. Read back with density_meta(), and NULL for a binary or categorical exposure.

References

Barrett, M., D'Agostino McGowan, L., & Gerke, T. *Causal Inference in R*. https://www.r-causal.org/

Rosenbaum, P. R., & Rubin, D. B. (1983). The central role of the propensity score in observational studies for causal effects. *Biometrika*, 70(1), 41–55.

Li, L., & Greene, T. (2013). A weighting analogue to pair matching in propensity score analysis. *The International Journal of Biostatistics*, 9(2), 215–234. (ATM weights)

Li, F., Morgan, K. L., & Zaslavsky, A. M. (2018). Balancing covariates via propensity score weighting. *Journal of the American Statistical Association*, 113(521), 390–400. (ATO weights)

Zhou, Y., Matsouaka, R. A., & Thomas, L. (2020). Propensity score weighting under limited overlap and model misspecification. *Statistical Methods in Medical Research*, 29(12), 3721–3756. (Entropy weights)

Hirano, K., & Imbens, G. W. (2004). The propensity score with continuous treatments. In *Applied Bayesian Modeling and Causal Inference from Incomplete-Data Perspectives* (pp. 73–84).

Robins, J. M., Hernán, M. A., & Brumback, B. (2000). Marginal structural models and causal inference in epidemiology. *Epidemiology*, 11(5), 550–560. (Stabilized weights and the conditional numerator)

Cole, S. R., & Hernán, M. A. (2008). Constructing inverse probability weights for marginal structural models. *American Journal of Epidemiology*, 168(6), 656–664. (Numerator covariates in the weighted model)

Hernán, M. A., & Robins, J. M. (2020). *Causal Inference: What If*. Chapman & Hall/CRC. (Chapter 12, stabilized weights and the conditional numerator)

Imbens, G. W. (2000). The role of the propensity score in estimating dose-response functions. *Biometrika*, 87(3), 706–710. (Generalized propensity score of a multi-valued exposure)

Feng, P., Zhou, X.-H., Zou, Q.-M., Fan, M.-Y., & Li, X.-S. (2012). Generalized propensity score for estimating the average treatment effect of multiple treatments. *Statistics in Medicine*, 31(7), 681–697.

van der Wal, W. M., & Geskus, R. B. (2011). ipw: An R package for inverse probability weighting. *Journal of Statistical Software*, 43(13), 1–23. (Multinomial numerator model for a multi-valued exposure)

Naimi, A. I., Moodie, E. E. M., Auger, N., & Kaufman, J. S. (2014). Constructing inverse probability weights for continuous exposures: a comparison of methods. *Epidemiology*, 25(2), 292–299.

Austin, P. C., & Stuart, E. A. (2015). Moving towards best practice when using inverse probability of treatment weighting (IPTW). *Statistics in Medicine*, 34(28), 3661–3679.

Greifer, N. *WeightIt: Weighting for Covariate Balance in Observational Studies*. R package. https://CRAN.R-project.org/package=WeightIt (the integrated numerator, which WeightIt has computed for continuous treatments since version 2.0.0)

See Also

- `psw()` for the returned weight vector class.
- `ps_trim()`, `ps_trunc()`, `ps_refit()`, and `ps_calibrate()` for modifying propensity scores before weighting.
- `ipw()` for inverse-probability-weighted estimation of causal effects.
- `ps_tilt()` for the tilting function each weight divides by the propensity score of the received exposure level, which also standardizes a g-computation estimate to the same target population.

Examples

```
# -- Binary exposure, numeric propensity scores ----------------------
set.seed(123)
ps <- runif(100, 0.1, 0.9)
trt <- rbinom(100, 1, ps)

wt_ate(ps, trt)
wt_att(ps, trt)
wt_atu(ps, trt)
wt_atm(ps, trt)
wt_ato(ps, trt)
wt_entropy(ps, trt)

# Stabilized ATE weights (reduces variance)
wt_ate(ps, trt, stabilize = TRUE)

# Inspect the result
w <- wt_ate(ps, trt)
estimand(w)
summary(w)

# -- Overlap-focused estimands handle extreme PS better --------------
ps_extreme <- c(0.01, 0.02, 0.98, 0.99, rep(0.5, 4))
trt_extreme <- c(0, 0, 1, 1, 0, 1, 0, 1)

max(wt_ate(ps_extreme, trt_extreme))
max(wt_ato(ps_extreme, trt_extreme))

# -- From a fitted GLM -----------------------------------------------
x1 <- rnorm(100)
x2 <- rnorm(100)
trt2 <- rbinom(100, 1, plogis(0.5 * x1 + 0.3 * x2))
ps_model <- glm(trt2 ~ x1 + x2, family = binomial)

# Exposure is extracted from the model automatically
wt_ate(ps_model)

# -- Continuous exposure from a fitted model -------------------------
dose <- 1 + 0.5 * x1 - 0.3 * x2 + rnorm(100)
dose_model <- lm(dose ~ x1 + x2)

# The exposure is read from the model, and a continuous exposure is
# stabilized without being asked
w_dose <- wt_ate(dose_model)

# A tail heavier than the normal's holds down the weight of a unit whose
# dose the model fits poorly
wt_ate(dose_model, .density = dens_t(df = 4))

# The stabilizing numerator can marginalize the conditional density over the
# units instead of reading the same family at the exposure's own moments
w_int <- wt_ate(
```

```
  dose_model,
  .density = dens_t(df = 4),
  numerator = "integrated"
)

# It can also be the conditional density a second model estimates, which is
# worth having when the model the estimates are read from also reads the
# variables that model conditions on
num_model <- lm(dose ~ x1)
w_cond <- wt_ate(dose_model, stabilize = num_model)

# What each set of weights records about the ratio it is
density_meta(w_dose)
density_meta(w_int)

# -- Data frame input ------------------------------------------------
ps_df <- data.frame(
  control = c(0.9, 0.7, 0.3, 0.1),
  treated = c(0.1, 0.3, 0.7, 0.9)
)
exposure <- c(0, 0, 1, 1)
wt_ate(ps_df, exposure)
wt_ate(ps_df, exposure, .propensity_col = "treated")

# -- Censoring weights -----------------------------------------------
cens_ps <- runif(50, 0.6, 0.95)
cens_ind <- rbinom(50, 1, cens_ps)
wt_cens(cens_ps, cens_ind)
estimand(wt_cens(cens_ps, cens_ind))  # "uncensored"


# -- Categorical exposure from a multinomial fit ---------------------
set.seed(5)
x3 <- rnorm(100)
dose_level <- factor(sample(c("low", "mid", "high"), 100, replace = TRUE))
cat_model <- nnet::multinom(dose_level ~ x3, trace = FALSE)

# The fit reports one probability per level, and the exposure and its level
# order are read from the model
wt_ate(cat_model)

# The same weights from the fitted matrix, which needs the exposure and its
# type supplied
ps_mat <- predict(cat_model, type = "probs")
wt_ate(ps_mat, dose_level, exposure_type = "categorical")

# A multinomial model of the exposure on a modifier stabilizes the weights on
# the probability of the level each unit took given that modifier
v <- rbinom(100, 1, 0.5)
cat_num_model <- nnet::multinom(dose_level ~ v, trace = FALSE)
w_cat <- wt_ate(cat_model, stabilize = cat_num_model)
numerator_model(w_cat)
```

wt_joint *Product weights for a joint intervention on two treatments*

Description

`wt_joint()` builds the weight for a joint intervention on two treatments by multiplying the two component weight vectors. A joint exposure needs a weight for the pair, and that weight factorizes sequentially: the density of the first treatment given the covariates, times the density of the second given the first treatment and the covariates. `wt_joint()` is the product, plus the checks that make the product a joint weight rather than an arbitrary one.

`is_joint_wt()` reports whether a set of weights was built this way, and `joint_wt_meta()` reads back what each component was.

Usage

```
wt_joint(w_a, w_e, exposure_type = NULL)

is_joint_wt(x)

joint_wt_meta(x)
```

Arguments

`w_a`, `w_e`	The two component weight vectors, as `psw()` objects. Both must target the `ate`, both must be the same length, and a component weighting a continuous exposure must be stabilized. The order is the factorization's order: `w_a` weights the first treatment and `w_e` weights the second, whose model conditions on the first.
`exposure_type`	A character vector of length two naming each component's exposure type, one of `"binary"`, `"categorical"`, or `"continuous"`, in the order the components were given. Defaults to the types the components record, which is what a weight function writes on the weights it builds. A value given here is used instead of what they record, and a component recording no type, such as one assembled by hand, is refused unless both types are named.
`x`	An object to test, or the weights to read the record from.

Value

`wt_joint()` returns a `psw()` vector of the elementwise product, carrying a `joint_wt_meta` record.

`is_joint_wt()` returns a single logical.

`joint_wt_meta()` returns the record as a list of five elements, with a sixth on a record that has dropped a stabilization score, each one per component and in the order the components were given, or `NULL` for weights that are not a product:

`exposure_type` The two components' exposure types, in order.

`stabilized` Whether each component was stabilized, in order.

`density` Each component's density record, as `density_meta()` returns it, in order. A component weighting a discrete exposure is `NULL`, since its weights are no ratio of densities.

`numerator_model` Each component's numerator model, as `numerator_model()` returns it, in order, for the components that have nowhere else to keep one. A component weighting a continuous exposure keeps its model inside its own `density` record, so this element is `NULL` for it, and so it is for a component stabilized on anything other than a fitted model. A record written before this element existed holds none at all, which says the same thing as a record whose components each record no model.

`stabilization_score` Each component's stabilization score, as `stabilization_score()` returns it, in order. A component stabilized on anything other than a score the caller supplied is `NULL`. A record written before this element existed holds none at all, which says the same thing as a record whose components each record no score.

`score_dropped` Whether each component's stabilization score was dropped by an operation that changed the length of the weights, in order. A drop empties the component's `stabilization_score` slot, which is the slot a component stabilized on anything else has, so a component marked `TRUE` here was stabilized on a score that is gone rather than on a numerator an estimator can rebuild. A record holding no such element, which is what a product nothing has shortened carries, says no component's score was dropped.

The record names the components rather than the observations, so it survives subsetting and every other operation that keeps the vector a `psw()`. A stabilization score is the exception: it holds one value per observation, so a score that no longer describes the observations the result holds is dropped the way the score on the weights themselves is, and a score of one value is carried at any length. Such a drop is recorded in `score_dropped`, so a component stabilized on a score that is gone stays distinguishable from one stabilized on the marginal numerator, and `ipw()` names it rather than standing a numerator in for it silently. Arithmetic that leaves the result unstabilized reduces the record's numerator side, as described under **What the product records**.

The factorization

The weight for a joint intervention on `A` and `E` is $1/[f(A|L)f(E|A,L)]$. The second factor conditions on the first treatment. The product of two *marginal* weights, $1/[f(A|L)f(E|L)]$, is a different quantity, and it is not the joint weight for any data in which `E` depends on `A`.

Nothing downstream can tell the two apart. Both are ordinary vectors of positive numbers, both fit an outcome model without complaint, and both return estimates with standard errors. That is why `joint_wt_models()` refuses a second model that does not condition on the first treatment, and why the dependence should be modeled flexibly rather than as a single additive term: an additive term satisfies the factorization but may model it badly, and a badly modeled dependence biases the joint weight in a way that is equally invisible.

Stabilization

A continuous component must be stabilized. The unstabilized density ratio $1/f(A|L)$ has a heavy right tail on its own, and multiplying it by a second weight inherits that tail, leaving the product with no usable variance. A continuous component is stabilized unless it was built with `stabilize = FALSE`, so the requirement is met by default. A binary or categorical component needs no stabilization and is accepted either way.

Any numerator that stabilizes the ratio satisfies the requirement, and any density the ratio is read in is accepted: a component built with `numerator = "integrated"`, with a heavier-tailed `.density`, or with a `stabilization_score` of its own is stabilized as much as the default marginal one is. What each component was built with is recorded per component rather than merged, so the product carries both answers.

What the product records

The result is a `psw()` with estimand `"ate"`, since the product of two `ate` weights targets the joint `ate`. It is marked stabilized when *either* component is, because a stabilizing numerator on either factor is a stabilizing numerator on the product; `joint_wt_meta()` keeps the per-component truth, so the coarse flag hides nothing.

The product records nothing about trimming, truncation, or calibration of the propensity scores the components were built from. Those records name the observations of one component, and the product is not that component.

Arithmetic that leaves a result unstabilized, such as multiplying the product by unstabilized weights, reduces the numerator side of the record it carries: the components read as unstabilized, their numerator models and stabilization scores are dropped, and each density record's numerator fields read as the record of no numerator. Such a result was divided by no numerator, so a record naming one would describe a different vector. What each component's weights divide by stays, and so does the component structure.

See Also

`joint_wt_models()` for the container recording the two treatment models, and `wt_ate()` for building the components. These are used by `ipw()` methods for joint treatment models.

Examples

```
set.seed(4)
n <- 200
x1 <- rnorm(n)
a <- rbinom(n, 1, plogis(0.3 * x1))
e <- rbinom(n, 1, plogis(-0.2 + 0.5 * x1 - 0.8 * a))
dat <- data.frame(x1, a, e)

# The second model conditions on the first treatment, and does so flexibly
mod_a <- glm(a ~ x1, data = dat, family = binomial())
mod_e <- glm(e ~ a * x1, data = dat, family = binomial())

w <- wt_joint(wt_ate(mod_a), wt_ate(mod_e))
head(w)
estimand(w)
is_joint_wt(w)
joint_wt_meta(w)

# A continuous component must be stabilized, which it is by default
d <- 0.5 + 0.6 * x1 - 0.7 * a + rnorm(n)
mod_d <- lm(d ~ a * x1, data = dat)
w_d <- wt_ate(mod_d)
```

```
# Each component records the exposure type it weights, so the product knows
# which of them is the dose without being told
joint_wt_meta(wt_joint(wt_ate(mod_a), w_d))
```

`wt_trunc` *Truncate (Winsorize) Propensity Score Weights*

Description

`wt_trunc()` bounds extreme weights at a limit read on the scale of the weights themselves, replacing each weight beyond the limit with the limit. No unit is removed. Where `ps_trunc()` bounds the propensity scores the weights are built from, `wt_trunc()` bounds the weights after they are built, so it applies to weights for any exposure type, including the density-ratio weights of a continuous exposure, which have no propensity score to bound.

Usage

```
wt_trunc(
  .weights,
  method = c("adaptive", "wt", "pctl", "count"),
  lower = NULL,
  upper = NULL,
  ...
)
```

Arguments

`.weights`	A psw vector, such as one returned by `wt_ate()`, or a numeric vector of weights. A numeric vector is returned as a psw vector with no estimand. Propensity scores, including those returned by `ps_trim()`, `ps_trunc()`, and `ps_calibrate()`, are refused with an error of class `propensity_type_error`: build weights from them first.
`method`	The rule that sets the bound: • `"adaptive"` (default): an upper bound of $\sqrt{n}\log(n)/5$ (Gruber et al., 2022), where n is the number of weights present. • `"wt"`: bounds given as weight values. • `"pctl"`: bounds read at sample quantiles of the weights. • `"count"`: bounds that move a given number of the most extreme weights.
`lower`, `upper`	The bounds, read according to `method`: • `"adaptive"`: not used. Supplying either is ignored with a warning. • `"wt"`: `upper` is required and must be a positive finite number. `lower` is optional and must be a finite number of at least 0 and below `upper`. • `"pctl"`: `upper` is required and must lie in (0.5, 1). `lower` is optional and must lie in (0, 0.5).

- `"count"`: `upper` is required and must be a whole number of at least 1; the `upper` largest weights are bounded at the next largest. `lower` is optional and bounds the `lower` smallest weights at the next smallest in the same way.

`...` Not used. Any argument supplied here is an error, so that a misspelled bound is not silently ignored.

Details

Methods:

`"adaptive"` is the default. Its bound depends on the sample size alone and loosens as the sample grows: at $n = 1000$ it is 43.7. Gruber et al. (2022) derived it for the propensity score of a binary treatment, choosing a bound that loosens with the sample so that its bias shrinks. Whether the rule carries over to the density-ratio weights of a continuous exposure has not been established in the literature. Here n counts the weights that are present, so for weights built from a trimmed propensity score, whose trimmed units are `NA`, it is the number of units kept by the trim. The bound is below 1 for $n \leq 6$ (0.72 at $n = 5$), so for a sample that small `"adaptive"` pulls every weight above that value down to a constant below 1.

`"wt"` winsorizes at the values given, and `"count"` at the order statistic next to the weights it bounds. Under `"count"`, a weight tied with that order statistic is not moved, so fewer weights than `upper` may change. The two counts must leave at least two weights between them, and `upper` alone must leave at least one weight below it.

`"pctl"` reads its bounds with `stats::quantile()` at its default type, as `ps_trunc(method = "pctl")` does. Because such a rule moves a fixed share of the units however the weights are distributed, it is best used to check how sensitive an analysis is to its bound rather than as the bound itself. An `upper` below 0.5 is refused rather than read as the other tail: write `upper = 0.9` to bound the upper tail at the 0.9 quantile, or `lower = 0.1` to bound the lower tail at the 0.1 quantile.

`"adaptive"` and `"pctl"` need at least two weights present, and every method leaves a missing weight missing.

One-sided by default:

Each method bounds only the upper tail unless `lower` is supplied, and `"adaptive"`, which takes no `lower`, always bounds only the upper tail. The largest weights are the ones that dominate an estimate; small weights contribute little to it, so bounding them changes little. `lower` bounds the lower tail when that is wanted.

What is recorded:

The result keeps every attribute of `.weights`: the stabilization status and any numerator model, the density record of a continuous exposure, the attributes of a categorical exposure, and the records left by `ps_trim()`, `ps_trunc()`, or `ps_calibrate()`. It gains a `wt_truncated` flag, read by `is_wt_truncated()`, and a record of the method, the bounds as given and as applied, and the positions of the weights moved, read by `is_unit_wt_truncated()`. The record is positional and is dropped by any operation that changes the length of the weights; see psw.

The estimand gains the label `"; weights truncated"`, as in `"ate; weights truncated"`. A truncated weight targets a different quantity from the weight before truncation: any fixed bound introduces a bias (Ma and Wang, 2020; Gruber et al., 2022), and an interval computed with the truncated weights held fixed is an interval for that quantity. A weight truncation does not describe the same operation as truncating the propensity score, which is labeled `"; truncated"` and reported by `is_ps_truncated()`.

Weights that have already been truncated are returned unchanged with a warning of class `propensity_already_modified_`. One record describes one truncation, so to truncate at a different bound, call `wt_trunc()` on the original weights.

Weights from a trimmed, truncated, or calibrated score:

Weights built from a score modified by `ps_trim()`, `ps_trunc()`, or `ps_calibrate()` can be truncated as well. The result carries both records, and the labels stack, as in `"ate; trimmed; weights truncated"`. The package does not prevent combining these tools.

Effect estimation:

`ipw()` does not accept weights whose values were truncated, because its standard errors do not account for the bound. An analysis with truncated weights needs an interval built another way: one computed with the weights held fixed, which conditions on the bound applied, or a bootstrap that repeats the truncation in every resample, which a percentile bound in particular needs, since that bound is itself estimated from the weights.

To see how much a truncation changed the effective sample size, add the weights before and after it to a data frame as separate columns and pass both to `halfmoon::check_ess()`.

Truncating weights or trimming the density:

For a continuous exposure there are two ways to hold down extreme weights. `wt_trunc()` bounds the weights once they are built and keeps every unit. `ps_trim()` with `method = "density"` or `method = "resid"` sets aside the units whose observed dose is implausible under the dose model, and `ps_refit()` refits the model on the units that remain; see **Trimming a dose model** in `ps_trim()`. Both change the estimand. A fixed bound introduces a bias that does not vanish however large the sample, while a bound that loosens with the sample, as `"adaptive"` does, leaves the estimand alone in the limit. A trim describes only the units it keeps. `wt_trunc()` applies to weights from any density family, and so does `"density"`; `"resid"` accepts the normal, t, and Laplace families only. `ipw()` refuses the weights from either tool.

A simulation study run for this package (samples of 1000, 1000 replications, a correctly specified mean for the dose) found that which tool helps depends on whether the density family fits the residuals. The coverage figures below are for a sandwich interval that holds the weights fixed, unless stated. With normal residuals, a normal density, and an effective sample size of 64% of the sample, the `"adaptive"` bound moved almost no weight and covered 0.933, as the untruncated weights did under the same interval. The M-estimation interval on the untruncated weights covered 0.863, so the gain in coverage there comes from the interval, not the bound. What the bound bought was precision: a root mean squared error of 0.084 against 0.096, for a bias of 0.015 (3% of the effect) against 0.012. At an effective sample size of 53% it covered 0.847 against 0.839 untruncated, and at 81% it changed almost nothing (it applied in 0.2% of replications, leaving coverage and RMSE unchanged). A density trim at `lower = 0.01` covered 0.843 at 64%, below the untruncated weights.

With t(4) or Laplace residuals, or a residual spread that varied with a covariate, the normal density fit some units badly. The bound then helped little (coverage 0.822, 0.906, and 0.644, against 0.788, 0.881, and 0.618 untruncated), while the 1% density trim covered **0.989**, **0.980**, and **0.979**, because it removed the units that density fit worst. The trim's intervals were computed on the trimmed sample and were conservative, and its bias was measured against the effect in the full population, so part of that bias is the change of estimand. The `"resid"` method was not simulated; these results carry over to it only because a bound on the standardized residual is a floor on the conditional density.

Neither tool is the first step. Under the same heavy-tailed residuals, `dens_t(4)` at its default scale covered 0.926 to 0.939 with the M-estimation interval, whose standard errors were close to right, with about a third of the root mean squared error of the bounded normal weights, and kept every unit. So first check the density family against the residuals, for instance by comparing the weights to those of a `dens_kernel()` density or a heavier tail. If the weights are still heavy under a family that fits, bound them with `wt_trunc()`. Trim the dose model when the units at the low-density end are ones the analysis should not describe, and state the population the trimmed analysis describes.

Value

A psw vector of the same length as `.weights`, with the bounded values, the estimand labeled as truncated, and a record of the truncation.

References

Gruber, S., Phillips, R. V., Lee, H., & van der Laan, M. J. (2022). Data-adaptive selection of the propensity score truncation level for inverse-probability-weighted and targeted maximum likelihood estimators of marginal point treatment effects. *American Journal of Epidemiology*, 191(9), 1640–1651.

Ma, X., & Wang, J. (2020). Robust inference using inverse probability weighting. *Journal of the American Statistical Association*, 115(532), 1851–1860.

See Also

`is_wt_truncated()` and `is_unit_wt_truncated()` to read the record, `ps_trunc()` to bound propensity scores instead, and `ps_trim()` to remove units.

Examples

```
set.seed(1)
n <- 50
x <- rnorm(n)
z <- rbinom(n, 1, plogis(2 * x))
fit <- glm(z ~ x, family = binomial)
w <- wt_ate(fit, .exposure = z)

# The adaptive bound, on the upper tail
w_adaptive <- wt_trunc(w)
w_adaptive
is_wt_truncated(w_adaptive)
which(is_unit_wt_truncated(w_adaptive))

# A bound given as a weight, on one tail or both
wt_trunc(w, method = "wt", upper = 5)
wt_trunc(w, method = "wt", lower = 1.05, upper = 5)

# The 99th percentile, as a sensitivity check
wt_trunc(w, method = "pctl", upper = 0.99)

# The three largest weights, bounded at the fourth largest
wt_trunc(w, method = "count", upper = 3)
```

```
# Weights for a continuous exposure
a <- 1 + 0.5 * x + rnorm(n)
dose <- lm(a ~ x)
w_dose <- wt_ate(dose, .exposure = a)
wt_trunc(w_dose)

# Weights from a trimmed propensity score carry both records
trimmed <- ps_refit(ps_trim(fit, lower = 0.05, upper = 0.95), fit)
wt_trunc(wt_ate(trimmed, .exposure = z), method = "wt", upper = 5)
```

`wt_trunc_sensitivity` *Tabulate Truncated Weights Over a Grid of Bounds*

Description

`wt_trunc_sensitivity()` truncates a set of weights with `wt_trunc()` at each bound in a grid and tabulates what each truncation does to them: the bound as given and as applied, the number of weights moved, and the range and mean of the weights that result. The first row describes the weights before any truncation, so every other row can be read against it. This is the weight side of the sensitivity table Cole and Hernán (2008) recommend for choosing a bound.

Usage

```
wt_trunc_sensitivity(
  .weights,
  method = c("pctl", "wt", "count"),
  lower = NULL,
  upper = NULL
)
```

Arguments

`.weights`	A psw vector, such as one returned by `wt_ate()`, or a numeric vector of weights. Propensity scores, including those returned by `ps_trim()`, `ps_trunc()`, and `ps_calibrate()`, are refused with an error of class `propensity_type_error`, and weights that have already been truncated with an error of class `propensity_already_modified_err`
`method`	The rule that reads each bound, one of `"pctl"` (the default), `"wt"`, or `"count"`, as in `wt_trunc()`. The `"adaptive"` method sets a single bound from the number of weights, so it has no grid to vary and is not accepted.
`lower`	An optional grid of lower bounds, read according to `method` as `wt_trunc()` reads `lower`. A single value is used with every upper bound; otherwise `lower` must have the same length as `upper`, and the two are paired element by element.
`upper`	The grid of upper bounds, read according to `method` as `wt_trunc()` reads `upper`. For `"pctl"` it defaults to `c(0.99, 0.975, 0.95, 0.90)`; for `"wt"` and `"count"` it is required.

Details

Each row after the first is computed by calling `wt_trunc()` with that row's bounds, so it describes exactly the weights that call returns. Every bound in the grid must be one `wt_trunc()` accepts: a single invalid bound refuses the whole grid, with the error `wt_trunc()` would raise for it, rather than leaving a gap in the table. The rows follow the grid in the order given.

The summaries are taken over the weights that are present, so missing weights, such as those of units set aside by `ps_trim()`, take no part in them and are never counted as truncated. A grid over weights of which none are present is refused with an error of class `propensity_range_error`. Weights can be zero, and a zero weight makes `range_ratio` infinite.

Weights that have already been truncated are refused, because the first row would then describe weights that were already bounded rather than the weights before truncation. Pass the weights as they were before `wt_trunc()` instead.

What is left to the caller:

The table does not include effect estimates or effective sample sizes. propensity fits no outcome models, so to add estimates to the table, map over the grid, truncate the weights at each bound with `wt_trunc()`, and fit the outcome model with each set of weights. For the effective sample size, add the weights before and after truncation to a data frame as separate columns and pass them to `halfmoon::check_ess()`.

Value

A tibble with one row for the untruncated weights followed by one row per bound, and the columns:

- `lower`, `upper`: the bounds as given, `NA` in the first row and `lower` `NA` when no lower bound was given.
- `lower_value`, `upper_value`: the bounds as applied, on the scale of the weights, recorded by `wt_trunc()`. Both are `NA` in the first row, and `lower_value` is `NA` when no lower bound was given.
- `n_truncated`: the number of weights moved, 0 in the first row.
- `min`, `max`, `mean`: the smallest, largest, and mean weight present.
- `range_ratio`: `max` / `min`.

References

Cole, S. R., & Hernán, M. A. (2008). Constructing inverse probability weights for marginal structural models. *American Journal of Epidemiology*, 168(6), 656–664.

See Also

`wt_trunc()` to truncate the weights at a chosen bound.

Examples

```
set.seed(1)
n <- 200
x <- rnorm(n)
z <- rbinom(n, 1, plogis(2 * x))
```

```
fit <- glm(z ~ x, family = binomial)
w <- wt_ate(fit, .exposure = z)

# The default percentile grid. The first row is the untruncated weights.
wt_trunc_sensitivity(w)

# A grid of absolute bounds, and a lower tail bounded at every one of them
wt_trunc_sensitivity(w, method = "wt", upper = c(20, 10, 5))
wt_trunc_sensitivity(w, method = "wt", lower = 1.05, upper = c(20, 10, 5))

# The largest one, three, and ten weights
wt_trunc_sensitivity(w, method = "count", upper = c(1, 3, 10))
```

Index

as.data.frame(), *22*, *67*, *75*, *83–85*
as.data.frame.ipw_diagnostic_se *(*print.ipw_diagnostic_se*)*, 51
as_conditional(), *36*, *84, 85*, *87, 88*
as_marginal(), *36*, *84, 85*, *87, 88*
as_psw *(*psw*)*, 77
as_psw(), *3*
augment(), *11*, *85*
augment.ipw, 4

binomial(), *50*, *94*, *96*, *103*

c(), *45, 46*, *66, 67*, *74*, *80*
causalgenerics::as_conditional(), *19*
causalgenerics::as_marginal(), *15*, *19*
causalgenerics::estimand(), *80*
causalgenerics::is_causal_wt(), *80*
causalgenerics::joint_exposure(), *13*, *24*
causalgenerics::new_ipw(), *19*

dens_fn *(*dens_normal*)*, 6
dens_fn(), *97*, *102*
dens_kernel *(*dens_normal*)*, 6
dens_kernel(), *97*, *102*, *104*, *114*
dens_laplace *(*dens_normal*)*, 6
dens_laplace(), *9*, *64*, *97*, *102, 103*
dens_normal, 6
dens_normal(), *9*, *64*, *97*, *102*
dens_t *(*dens_normal*)*, 6
dens_t(), *9*, *64*, *66*, *97*, *102–104*
density_meta *(*exposure_type*)*, 8
density_meta(), *8*, *50*, *79*, *97*, *104*, *109*

estimand(), *3*
exposure_type, 8
exposure_type(), *79*, *104*

format.propensity_density_meta *(*exposure_type*)*, 8

gaussian(), *94*, *103*
generics::tidy(), *15*, *20*
glance(), *5*, *36*, *85*, *88*
glance.ipw, 10
glance.ipw_pooled, 12
glm, *56*
glm(), *94*, *103*

ipw(), *3–6*, *8–12*, *14*, *39*, *48–52*, *65*, *83–85*, *88*, *95, 96*, *99–101*, *103*, *105*, *109, 110*, *113*
ipw-methods, 13
ipw.glm *(*ipw-methods*)*, 13
ipw.joint_wt_models *(*ipw-methods*)*, 13
ipw.lm *(*ipw-methods*)*, 13
ipw.multinom *(*ipw-methods*)*, 13
is_joint_wt *(*wt_joint*)*, 108
is_joint_wt_models *(*joint_wt_models*)*, 48
is_ps_calibrated, 41
is_ps_calibrated(), *54*, *57*
is_ps_trimmed, 42
is_ps_trimmed(), *45*, *68*, *70*
is_ps_truncated, 43
is_ps_truncated(), *46, 47*, *76, 77*, *112*
is_psw *(*psw*)*, 77
is_psw(), *3*
is_refit, 44
is_refit(), *57*, *80, 81*
is_stabilized *(*psw*)*, 77
is_stabilized(), *3*
is_unit_trimmed, 45
is_unit_trimmed(), *42–44*, *66–68*, *70*, *81*
is_unit_truncated, 46
is_unit_truncated(), *43*, *47*, *74–77*, *81*
is_unit_wt_truncated *(*is_wt_truncated*)*, 47
is_unit_wt_truncated(), *81*, *112*, *114*
is_wt_truncated, 47
is_wt_truncated(), *79*, *112*, *114*

joint_wt_meta *(wt_joint)*, 108
joint_wt_models, 48
joint_wt_models(), *13, 14*, *25*, *28*, *109, 110*

lm(), *94*, *96*, *103*

MASS::psi.bisquare(), *32*
MASS::psi.hampel(), *32*
MASS::psi.huber(), *32*
MASS::rlm(), *13–15*, *19*, *29, 30*, *32*, *56*, *94*, *103*
mean(), *82*
mgcv::bam(), *33*
mgcv::gam(), *14, 15*, *19*, *29, 30*, *32–34*, *53*, *56*, *94*, *103*
mgcv::gamm(), *33*
mice::pool(), *84*, *87*
model.frame(), *56*
multinom, *56*

na.omit(), *67*
new_psw *(psw)*, 77
nnet::multinom(), *13, 14*, *24, 25*, *29–31*, *39*, *48*, *50*, *94*, *96*, *98*, *101*
numerator_model, 49
numerator_model(), *79*, *96*, *101*, *109*

poisson(), *103*
pool_ipw(), *12*, *35, 36*, *85–88*
print(), *51*
print.ipw_diagnostic_se, 51
print.propensity_density_meta *(exposure_type)*, 8
propensity *(propensity-package)*, 2
propensity-package, 2
ps_calibrate, 52
ps_calibrate(), *3*, *37*, *42*, *57*, *60*, *68*, *82*, *89*, *94*, *101*, *105*, *111–113*, *115*
ps_refit, 55
ps_refit(), *3*, *44*, *61*, *65*, *67, 68*, *70*, *76*, *89*, *94*, *105*, *113*
ps_tilt, 58
ps_tilt(), *21*, *105*
ps_trim, *44*, 61
ps_trim(), *3*, *9*, *37*, *39, 40*, *42–45*, *53–57*, *60*, *69, 70*, *72*, *76*, *82*, *89*, *94, 95*, *97*, *101*, *103*, *105*, *111–116*
ps_trim_meta, 69
ps_trim_meta(), *43*, *45*, *66*, *68*
ps_trunc, 70, *76*
ps_trunc(), *3*, *37*, *39, 40*, *42, 43*, *46*, *53, 54*, *56*, *60*, *62–64*, *68*, *77*, *82*, *89*, *94*, *101*, *105*, *111–115*
ps_trunc_meta, 76
ps_trunc_meta(), *43*, *46*, *75, 76*
psw, *44–47*, 77, *89*, *104*, *111, 112*, *114, 115*
psw(), *3–5*, *9, 10*, *20*, *33*, *105*, *108–110*

rep(), *67*, *74*

sort(), *67*, *74*
stabilization_score *(psw)*, 77
stabilization_score(), *50*, *109*
stats::coef(), *15*, *20*, *36*
stats::confint(), *15*, *19, 20*, *36*
stats::density(), *6, 7*
stats::df.residual(), *11*, *20*
stats::glm(), *13–18*, *21*, *24, 25*, *29–34*, *48*, *53, 54*, *56*, *60*, *65*, *74*
stats::lm(), *13–17*, *21*, *29, 30*, *32*, *34*, *48*, *56*, *65*, *74*
stats::model.frame(), *5*, *17*
stats::nobs(), *11, 12*, *20*
stats::quantile(), *112*
stats::vcov(), *15*, *19, 20*, *24*, *36*
stats::weights(), *20*
sum(), *82*

tibble, *5*, *11, 12*, *85*, *87*, *116*
tidy(), *5*, *11, 12*, *19*, *35, 36*, *85*, *87*
tidy.ipw, 83
tidy.ipw_diagnostic_se *(print.ipw_diagnostic_se)*, 51
tidy.ipw_pooled, 86

unique(), *67*, *74, 75*
update(), *56*

vctrs::vec_assign(), *67*, *75*, *81*
vctrs::vec_cast(), *67*, *75*
vctrs::vec_slice(), *45, 46*, *67*, *74*

wt_atc *(wt_ate)*, 89
wt_ate, 89
wt_ate(), *3*, *6*, *10*, *13*, *17, 18*, *20*, *23*, *25*, *28–31*, *33, 34*, *38, 39*, *50*, *54, 55*, *57*, *60*, *62*, *64, 65*, *71*, *74*, *77*, *79*, *82*, *110, 111*, *115*
wt_atm *(wt_ate)*, 89

wt_atm(), *3*, *39*, *82*
wt_ato *(*wt_ate*)*, 89
wt_ato(), *3*, *39*, *82*
wt_att *(*wt_ate*)*, 89
wt_att(), *3*, *21*, *39*, *59*, *82*
wt_atu *(*wt_ate*)*, 89
wt_atu(), *3*, *21*, *39*, *82*
wt_cens *(*wt_ate*)*, 89
wt_cens(), *3*, *6*
wt_entropy *(*wt_ate*)*, 89
wt_entropy(), *3*, *39*
wt_joint, 108
wt_joint(), *13*, *25, 26*, *29, 30*, *49*, *81*
wt_trunc, 111
wt_trunc(), *37*, *39*, *64, 65*, *73, 74*, *115, 116*
wt_trunc_sensitivity, 115