

Package: hvtiRpropensity (via r-universe)

October 20, 2026

Type Package

Title Propensity Score Methods for HVTI CORR Research at the Cleveland Clinic

Version 0.2.1

Date 2026-10-14

License GPL-3

Encoding UTF-8

URL <https://github.com/ehrlinger/hvtiRpropensity>,
<https://ehrlinger.github.io/hvtiRpropensity/>

BugReports <https://github.com/ehrlinger/hvtiRpropensity/issues>

Description Propensity score estimation, matching, and inverse-probability treatment weighting (IPTW) for clinical investigations by the Cardiovascular Outcomes, Registries and Research (CORR) group within The Heart & Vascular Institute (HVTI) at the Cleveland Clinic. Provides a consistent two-step workflow: construct a scored or weighted dataset, then inspect the results or pass them downstream. Balance diagnostics (standardized mean difference (SMD) tables, effective sample size) are compatible with the plotting functions in 'hvtiPlotR'. The propensity score follows Rosenbaum and Rubin (1983) <[doi:10.1093/biomet/70.1.41](https://doi.org/10.1093/biomet/70.1.41)>.

Depends R (>= 4.4.0)

Imports rlang (>= 1.0.0), stats, withr (>= 2.5.0)

Suggests hvtiPlotR, knitr, MASS, nnet, randomForestSRC, survival, pkgdown (>= 2.0.0), testthat (>= 3.0.0), devtools, quarto, pkgload

VignetteBuilder quarto

SystemRequirements Quarto command line tool
(<<https://github.com/quarto-dev/quarto-cli>>)

Roxygen list(markdown = TRUE)

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

Repository <https://ehrlinger.r-universe.dev>

Date/Publication 2026-10-14 22:57:47 UTC

RemoteUrl <https://github.com/ehrlinger/hvtiRpropensity>

RemoteRef HEAD

RemoteSha 05f43ccf20f12535da0c6e6c262c9413918a8ccf

Contents

bs_continuous	3
bs_count	5
fit_linear	9
fit_logistic	10
is_ps_data	13
print.ps_data	14
ps_forest	15
ps_logistic	16
ps_match	20
ps_mw_var	23
ps_nominal	25
ps_ordinal	27
ps_rmst	30
ps_stddiff	32
ps_stddiff_perm	34
ps_support	36
ps_weight	37
sa_evaluate	39
sa_overlap	41
sa_rosenbaum	43
sa_trim_sweep	45
sample_ps_data	47
sample_ps_data_count	48
sample_ps_data_nominal	49
sample_ps_data_ordinal	50
summary.ps_data	51
validate_logistic	51

Index	53
--------------	-----------

bs_continuous

*Estimate a balancing score via linear regression***Description**

Fits `stats::lm()` on a (possibly multiply-imputed) dataset, averages the per-patient **fitted values** across imputations, and assigns rank-based quintile, decile, and cluster strata. Mirrors the workflow in `tp.rm.continuous.balncing_score.sas` (PROC REG + PROC SUMMARY + the %grp macro for binary stratum indicators).

Usage

```
bs_continuous(
  formula,
  data,
  outcome_col = NULL,
  id_col = "id",
  imputation_col = NULL,
  score_col = "bs",
  n_strata = 10L,
  strata_col = "cluster",
  create_binary_strata = TRUE,
  covariates = NULL
)
```

Arguments

formula	A formula with the continuous outcome / exposure on the LHS, e.g. <code>hct_nadr ~ female + agee + in_bmi + hct_pr + iv_cpb</code> . A <code>.</code> on the right-hand side expands to the data's other columns, excluding the response, <code>id_col</code> and <code>imputation_col</code> .
data	A data frame, or a stacked MI data frame when <code>imputation_col</code> is set.
outcome_col	Name of the continuous outcome column. NULL (default) extracts from formula, whose response must read exactly one column; a supplied value must name that column.
id_col	Patient identifier column. Required, and checked, only when <code>imputation_col</code> is set. Default <code>"id"</code> .
imputation_col	Imputation-index column for stacked MI data. NULL (default) assumes a single complete dataset.
score_col	Name of the output balancing score column (averaged fitted value). Default <code>"bs"</code> .
n_strata	Number of strata (clusters). Also controls how many binary indicator columns are created when <code>create_binary_strata = TRUE</code> . Default <code>10L</code> .
strata_col	Name of the cluster/stratum column appended to <code>\$data</code> . Default <code>"cluster"</code> .

`create_binary_strata` Logical. If TRUE (default), binary columns `stra_1` through `stra_<n_strata>` are appended, mirroring the `%grp` macro in the SAS template.

`covariates` Covariate columns for diagnostics (not used in computation; reserved for future balance-checking helpers).

Details

The balancing score is the expected value of the continuous exposure (e.g. nadir haematocrit) given the full covariate set. Patients in the same stratum have similar expected exposure values, making within-stratum comparisons less confounded.

A formula with an aliased term, such as a covariate that duplicates another, is an error naming the imputation, rather than a score from a rank-deficient fit.

Value

An object of class `c("bs_continuous", "ps_data")` with:

`$data` Base data frame with `score_col`, `"quintile"`, `"decile"`, `strata_col`, and (if `create_binary_strata`) `stra_1` through `stra_<n_strata>` appended.

`$meta` Named list: `formula`, `outcome_col`, `id_col`, `imputation_col`, `score_col`, `n_strata`, `strata_col`, `bundle_version`, `model_family`, `package_versions`, `method`, `n_imputations`, `n_total`.

`$tables` Named list: `strata_counts` (stratum, n), `estimates` (Rubin-pooled coefficients when imputed, with Barnard and Rubin's degrees of freedom), `covariance`, `by_imputation` and `fit_status`, laid out as in `fit_linear()`.

`$models` The fitted `stats::lm()` models, one per imputation, named by imputation index.

Pooled degrees of freedom

With `imputation_col` set, each pooled coefficient is tested against a t distribution, and the `df` column of `$tables$estimates` gives its degrees of freedom. We use Barnard and Rubin's (1999) small-sample degrees of freedom, computed the way `mice::pool()` computes them, so a pooled table here and one from `mice` agree.

Rubin's (1987) original degrees of freedom are $\nu_{old} = (m - 1)/\lambda^2$, where m is the number of imputations and $\lambda = (1 + 1/m)B/T$ is the proportion of a coefficient's total variance T due to missing data, through the between-imputation variance B . (This is `lambda` in `mice`, not its finite-sample fraction of missing information, `fmi`.) When little is missing, λ is small and ν_{old} can run into the millions, larger than the sample could ever support, and the intervals come out too narrow. Barnard and Rubin cap it with the complete-data degrees of freedom ν_{com} , the value the analysis would have had with no missing data:

$$\nu_{obs} = \frac{\nu_{com} + 1}{\nu_{com} + 3} \nu_{com} (1 - \lambda), \quad \nu = \left(\frac{1}{\nu_{old}} + \frac{1}{\nu_{obs}} \right)^{-1}.$$

So ν never exceeds ν_{com} , and it approaches ν_{old} as ν_{com} grows.

ν_{com} is $n - k$, the residual degrees of freedom of the first imputation's model: n analysed patients minus k estimated parameters. That holds for every family, linear, logistic, ordinal, nominal and

count, as it does in mice. `nnet::multinom()` reports no residual degrees of freedom, so for a nominal model k is its effective degrees of freedom, again as mice does. Pooling stops with an error when ν_{com} is zero or negative, as it is for a saturated model, because no small-sample degrees of freedom exist then.

Two consequences are worth knowing. First, a single logistic, ordinal, nominal or count fit is tested against the normal distribution (df is Inf), but the same model pooled over imputations reports a finite df and a t test. A single linear fit uses its residual degrees of freedom. Second, SAS PROC MIANALYZE reports ν_{old} unless you give it the complete-data degrees of freedom with EDF=. To reproduce these df, and the p-values and intervals that depend on them, run PROC MIANALYZE with EDF= set to $n - k$.

Barnard J, Rubin DB (1999). Small-sample degrees of freedom with multiple imputation. *Biometrika*, 86(4), 948–955. doi:10.1093/biomet/86.4.948

Rubin DB (1987). *Multiple Imputation for Nonresponse in Surveys*. Wiley. doi:10.1002/9780470316696

See Also

`bs_count()`, `fit_linear()`, `ps_logistic()`, `sample_ps_data()`

Examples

```
# Use ef (ejection fraction) as a stand-in for a continuous exposure
dta <- sample_ps_data(n = 300, seed = 42)
obj <- bs_continuous(
  ef ~ age + female + diabetes + hypertension,
  data = dta,
  id_col = "id"
)
print(obj)
table(obj$data$cluster)
```

bs_count	<i>Estimate a balancing score via negative-binomial or Poisson regression</i>
----------	---

Description

Fits a GLM with a count distribution (`MASS::glm.nb()` for negative binomial, or `stats::glm()` with Poisson family) on a (possibly multiply-imputed) dataset, averages the per-patient **linear predictors** (log-scale, i.e. $x\beta$) across imputations, and assigns rank-based strata. Mirrors `tp.pm.count.balncing_score.sas` (PROC GENMOD dist=nb link=log

- PROC SUMMARY + the %grp macro).

Usage

```
bs_count(
  formula,
  data,
  outcome_col = NULL,
  id_col = "id",
  imputation_col = NULL,
  dist = c("negbin", "poisson"),
  score_col = "bs",
  n_strata = 10L,
  strata_col = "cluster",
  create_binary_strata = TRUE,
  covariates = NULL
)
```

Arguments

formula	A formula with the count outcome, untransformed, on the LHS, e.g. <code>rbc_tot ~ female + agee + in_hct + hx_csurg + iv_cpb</code> . A <code>.</code> on the right-hand side expands to the data's other columns, excluding the outcome, <code>id_col</code> and <code>imputation_col</code> .
data	A data frame, or a stacked MI data frame when <code>imputation_col</code> is set.
outcome_col	Name of the count outcome column. <code>NULL</code> (default) extracts from formula, whose response must read exactly one column; a supplied value must name that column.
id_col	Patient identifier column. Required, and checked, only when <code>imputation_col</code> is set. Default <code>"id"</code> .
imputation_col	Imputation-index column. <code>NULL</code> (default) means a single complete dataset.
dist	Distribution for the GLM. <code>"negbin"</code> (default) calls <code>MASS::glm.nb()</code> (negative binomial, log link); <code>"poisson"</code> calls <code>stats::glm()</code> with <code>family = poisson(link = "log")</code> .
score_col	Name of the output balancing score column (averaged linear predictor on the log scale). Default <code>"bs"</code> .
n_strata	Number of strata. Default 10L.
strata_col	Name of the cluster/stratum column. Default <code>"cluster"</code> .
create_binary_strata	Logical. If <code>TRUE</code> (default), binary indicator columns <code>stra_1</code> through <code>stra_<n_strata></code> are appended.
covariates	Covariate columns for diagnostics.

Details

The linear predictor (xbeta, log-scale) rather than the fitted count is used as the balancing score because it is on an unbounded continuous scale more suitable for rank-based stratification.

Value

An object of class `c("bs_count", "ps_data")` with:

`$data` data (the lowest-numbered imputation when `imputation_col` is set, without that column) with `score_col`, "quintile", "decile", `strata_col`, and (if `create_binary_strata`) `stra_1` through `stra_<n_strata>` appended.

`$meta` Named list: `formula`, `outcome_col`, `id_col`, `imputation_col`, `score_col`, `dist`, `theta` (the negative-binomial dispersion estimate, one per imputation, named by imputation index; NULL for `dist = "poisson"`), `n_strata`, `strata_col`, `bundle_version`, `model_family` ("count"), `package_versions`, `method`, `n_imputations`, `n_total`.

`$tables` Named list: `strata_counts` (stratum, n), `estimates`, `covariance`, `by_imputation` and `fit_status`, laid out as in `fit_logistic()` except that `estimates` has `rate_ratio`, the exponentiated coefficient, in place of `odds_ratio`. With `imputation_col`, `estimates` and `covariance` are pooled with Rubin's rules, with Barnard and Rubin's degrees of freedom.

`$models` The fitted models, one per imputation, named by imputation index: `MASS::glm.nb()` or `stats::glm()` fits.

Package dependency

`dist = "negbin"` (the default) requires the **MASS** package (a recommended R package, usually pre-installed). Install with `install.packages("MASS")` if missing, or use `dist = "poisson"`.

Pooled degrees of freedom

With `imputation_col` set, each pooled coefficient is tested against a t distribution, and the `df` column of `$tables$estimates` gives its degrees of freedom. We use Barnard and Rubin's (1999) small-sample degrees of freedom, computed the way `mice::pool()` computes them, so a pooled table here and one from `mice` agree.

Rubin's (1987) original degrees of freedom are $\nu_{old} = (m - 1)/\lambda^2$, where m is the number of imputations and $\lambda = (1 + 1/m)B/T$ is the proportion of a coefficient's total variance T due to missing data, through the between-imputation variance B . (This is `lambda` in `mice`, not its finite-sample fraction of missing information, `fmi`.) When little is missing, λ is small and ν_{old} can run into the millions, larger than the sample could ever support, and the intervals come out too narrow. Barnard and Rubin cap it with the complete-data degrees of freedom ν_{com} , the value the analysis would have had with no missing data:

$$\nu_{obs} = \frac{\nu_{com} + 1}{\nu_{com} + 3} \nu_{com} (1 - \lambda), \quad \nu = \left(\frac{1}{\nu_{old}} + \frac{1}{\nu_{obs}} \right)^{-1}.$$

So ν never exceeds ν_{com} , and it approaches ν_{old} as ν_{com} grows.

ν_{com} is $n - k$, the residual degrees of freedom of the first imputation's model: n analysed patients minus k estimated parameters. That holds for every family, linear, logistic, ordinal, nominal and count, as it does in `mice`. `nnet::multinom()` reports no residual degrees of freedom, so for a nominal model k is its effective degrees of freedom, again as `mice` does. Pooling stops with an error when ν_{com} is zero or negative, as it is for a saturated model, because no small-sample degrees of freedom exist then.

Two consequences are worth knowing. First, a single logistic, ordinal, nominal or count fit is tested against the normal distribution (`df` is `Inf`), but the same model pooled over imputations reports

a finite df and a t test. A single linear fit uses its residual degrees of freedom. Second, SAS PROC MIANALYZE reports ν_{old} unless you give it the complete-data degrees of freedom with EDF=. To reproduce these df, and the p-values and intervals that depend on them, run PROC MIANALYZE with EDF= set to $n - k$.

Barnard J, Rubin DB (1999). Small-sample degrees of freedom with multiple imputation. *Biometrika*, 86(4), 948–955. doi:10.1093/biomet/86.4.948

Rubin DB (1987). *Multiple Imputation for Nonresponse in Surveys*. Wiley. doi:10.1002/9780470316696

See Also

[bs_continuous\(\)](#), [ps_logistic\(\)](#)

Examples

```
# Mirrors the SAS template workflow:
# PROC GENMOD dist=nb link=log (negative binomial for overdispersed counts)
# xbeta (log-scale linear predictor) used as the balancing score
# (not the fitted count -- log scale is more suitable for stratification).
# The %grp macro binary indicators stra_1 ... stra_k are created when
# create_binary_strata = TRUE (the default).

dta <- sample_ps_data_count(n = 300, seed = 42)

# Poisson fallback (no MASS dependency)
obj <- bs_count(
  rbc_tot ~ female + age + diabetes + hypertension,
  data = dta,
  dist = "poisson"
)
print(obj)
table(obj$data$cluster) # 10 equal-ish strata by default

# Binary stratum indicators mirror the SAS %grp macro output
head(obj$data[, c("id", "bs", "cluster", "stra_1", "stra_2", "stra_10")])

# Negative binomial (accounts for overdispersion; requires MASS)
if (requireNamespace("MASS", quietly = TRUE)) {
  obj_nb <- bs_count(
    rbc_tot ~ female + age + diabetes + hypertension,
    data = dta,
    dist = "negbin"
  )
  print(obj_nb)
}
```

fit_linear

*Fit an ordinary least-squares model bundle***Description**

Fits `stats::lm()` to one dataset, or once per imputation of stacked long-form data, and combines imputations with Rubin's rules. A single fit is inferred on the t distribution with its residual degrees of freedom, as PROC REG reports it. Pooled coefficients use Barnard and Rubin's degrees of freedom, described below.

Usage

```
fit_linear(
  formula,
  data,
  outcome_col = NULL,
  id_col = "id",
  imputation_col = NULL,
  prediction_col = "fitted"
)
```

Arguments

<code>formula</code>	Model formula. The response may be transformed, e.g. $\log(\text{los}) \sim \text{age}$. A <code>.</code> on the right-hand side expands to the data's other columns, excluding the response, <code>id_col</code> and <code>imputation_col</code> .
<code>data</code>	A data frame, or stacked imputations when <code>imputation_col</code> is set.
<code>outcome_col</code>	Outcome column. By default the one variable in the formula's response; if supplied, it must name that variable. The response must read exactly one column.
<code>id_col</code>	Patient identifier column. Required only when <code>imputation_col</code> is set.
<code>imputation_col</code>	Imputation-index column, or NULL for one dataset.
<code>prediction_col</code>	Name of the fitted-value column added to <code>\$data</code> , averaged across imputations.

Value

An object of class `c("rm_fit", "ps_data")`. `$tables$fit_quality` gives each fit's residual degrees of freedom, residual standard error and R-squared; R-squared is NA for imputed data, where Rubin's rules define none.

Pooled degrees of freedom

With `imputation_col` set, each pooled coefficient is tested against a t distribution, and the `df` column of `$tables$estimates` gives its degrees of freedom. We use Barnard and Rubin's (1999) small-sample degrees of freedom, computed the way `mice::pool()` computes them, so a pooled table here and one from `mice` agree.

Rubin's (1987) original degrees of freedom are $\nu_{old} = (m - 1)/\lambda^2$, where m is the number of imputations and $\lambda = (1 + 1/m)B/T$ is the proportion of a coefficient's total variance T due to missing data, through the between-imputation variance B . (This is lambda in mice, not its finite-sample fraction of missing information, fmi.) When little is missing, λ is small and ν_{old} can run into the millions, larger than the sample could ever support, and the intervals come out too narrow. Barnard and Rubin cap it with the complete-data degrees of freedom ν_{com} , the value the analysis would have had with no missing data:

$$\nu_{obs} = \frac{\nu_{com} + 1}{\nu_{com} + 3} \nu_{com} (1 - \lambda), \quad \nu = \left(\frac{1}{\nu_{old}} + \frac{1}{\nu_{obs}} \right)^{-1}.$$

So ν never exceeds ν_{com} , and it approaches ν_{old} as ν_{com} grows.

ν_{com} is $n - k$, the residual degrees of freedom of the first imputation's model: n analysed patients minus k estimated parameters. That holds for every family, linear, logistic, ordinal, nominal and count, as it does in mice. `nnet::multinom()` reports no residual degrees of freedom, so for a nominal model k is its effective degrees of freedom, again as mice does. Pooling stops with an error when ν_{com} is zero or negative, as it is for a saturated model, because no small-sample degrees of freedom exist then.

Two consequences are worth knowing. First, a single logistic, ordinal, nominal or count fit is tested against the normal distribution (df is Inf), but the same model pooled over imputations reports a finite df and a t test. A single linear fit uses its residual degrees of freedom. Second, SAS PROC MIANALYZE reports ν_{old} unless you give it the complete-data degrees of freedom with EDF=. To reproduce these df, and the p-values and intervals that depend on them, run PROC MIANALYZE with EDF= set to $n - k$.

Barnard J, Rubin DB (1999). Small-sample degrees of freedom with multiple imputation. *Biometrika*, 86(4), 948–955. doi:10.1093/biomet/86.4.948

Rubin DB (1987). *Multiple Imputation for Nonresponse in Surveys*. Wiley. doi:10.1002/9780470316696

See Also

`fit_logistic()`, `bs_continuous()`

Examples

```
dta <- sample_ps_data(n = 150, seed = 42)
fit <- fit_linear(ef ~ age + female, data = dta)
fit$tables$estimates[, c("term", "estimate", "std.error", "p.value")]
```

fit_logistic

Fit a general logistic model bundle

Description

Fits binary, proportional-odds ordinal, or generalized-logit nominal models under an explicit outcome-level contract. Stacked long-form imputations are fitted separately, one model per imputation. The coefficients are combined with Rubin's rules; the predicted probabilities are averaged per patient.

Usage

```
fit_logistic(
  formula,
  data,
  family = c("binary", "ordinal", "nominal"),
  outcome_col = NULL,
  id_col = "id",
  imputation_col = NULL,
  outcome_levels,
  event_level = NULL,
  reference_level = NULL,
  prediction_prefix = "prob",
  trace = FALSE
)
```

Arguments

formula	Model formula with the outcome column, untransformed, on the left. A . on the right-hand side expands to the data's other columns, excluding the outcome, id_col and imputation_col.
data	A data frame containing the outcome and predictors.
family	Model family: "binary" (default, <code>stats::glm()</code> with a logit link), "ordinal" (proportional-odds cumulative logit, <code>MASS::polr()</code> ; a two-level outcome is fitted with <code>stats::glm()</code>) or "nominal" (generalized logit, <code>nnet::multinom()</code>).
outcome_col	Outcome column. By default it is read from formula.
id_col	Patient identifier column. Default "id". Required, and checked, only when imputation_col is set, where it aligns each patient's rows across imputations.
imputation_col	Imputation-index column, or NULL for one dataset.
outcome_levels	Complete outcome levels in their declared order.
event_level	Event level for a binary model.
reference_level	Reference level for a nominal model.
prediction_prefix	Name for a binary probability column, or prefix for ordinal and nominal level-specific probability columns.
trace	Whether model engines should print fitting progress.

Details

Ordinal standard errors. An ordinal model is fitted with `MASS::polr()` at a tight `optim()` tolerance (`reltol = 1e-14`), and its covariance in `$tables` is the inverse of the expected (Fisher) information, the matrix SAS PROC LOGISTIC reports. `polr()` itself, and `vcov()` on the model in `$models`, invert the observed information instead. For the cumulative logit the two differ at the estimate, here by a fraction of a percent in the standard errors, so the bundle's ordinal standard errors and p-values match SAS rather than `summary()` of the stored model. Binary and nominal models use canonical links, where the two matrices are the same.

Multiple imputation. With `imputation_col` set, one model is fitted to each imputed dataset. The two outputs are then combined differently. `$tables$estimates` and `$tables$covariance` pool the coefficients and their variances with Rubin's rules, with Barnard and Rubin's degrees of freedom (see "Pooled degrees of freedom" below). The probability columns in `$data` are the per-patient mean, on the probability scale, of the predictions from each imputation's model. Rubin's rules play no part there: averaging the predictions is what Leyrat et al. (2019) call the "across" approach. For a propensity score they favour the "within" approach instead; see `ps_logistic()` for the recipe.

`$data` is built from the lowest-numbered imputation: its rows, ordered as they appear there, its covariate values, and no `imputation_col`.

Value

An object of class `c("lm_fit", "ps_data")` with:

- `$data` data (the lowest-numbered imputation when `imputation_col` is set, without that column) with the predicted probabilities appended: one column named `prediction_prefix` for a binary model (prob by default), the probability of `event_level`; or one `<prediction_prefix>_<level>` column per outcome level for an ordinal or nominal model, whose outcome column is returned as a factor (ordered for an ordinal model). A row missing a predictor has missing probabilities.
- `$meta` Named list: `bundle_version` (1), `formula`, `outcome_col`, `model_family`, `outcome_levels`, `event_level` (binary only, else NULL), `reference_level` (nominal only, else NULL), `cumulative_direction` (ordinal only, else NULL), `predictors`, `id_col`, `imputation_col`, `n_imputations`, `n_input`, `n_analyzed`, `n_excluded` (summed over imputations), `prediction_prefix` and `package_versions` (R, this package and the model engine).
- `$tables` Named list: `estimates` (one row per term: term, estimate, std.error, statistic, df, p.value, conf.low, conf.high, odds_ratio (NA for an ordinal threshold), and pooled, TRUE when Rubin's rules combined imputations; a pooled df is Barnard and Rubin's, see "Pooled degrees of freedom"), `covariance` (the coefficient covariance matrix, pooled when imputed), `by_imputation` (imputation, term, estimate, std.error) and `fit_status` (one row per imputation: imputation, converged, n_input, n_analyzed, n_excluded).
- `$models` The fitted models, one per imputation, named by imputation index ("1" for a single dataset). For binary bundles, `validate_logistic()` scores new data with them; it rejects ordinal and nominal bundles.

Pooled degrees of freedom

With `imputation_col` set, each pooled coefficient is tested against a t distribution, and the `df` column of `$tables$estimates` gives its degrees of freedom. We use Barnard and Rubin's (1999) small-sample degrees of freedom, computed the way `mice::pool()` computes them, so a pooled table here and one from `mice` agree.

Rubin's (1987) original degrees of freedom are $\nu_{old} = (m - 1)/\lambda^2$, where m is the number of imputations and $\lambda = (1 + 1/m)B/T$ is the proportion of a coefficient's total variance T due to missing data, through the between-imputation variance B . (This is `lambda` in `mice`, not its finite-sample fraction of missing information, `fmi`.) When little is missing, λ is small and ν_{old} can run into the millions, larger than the sample could ever support, and the intervals come out too narrow. Barnard and Rubin cap it with the complete-data degrees of freedom ν_{com} , the value the analysis

would have had with no missing data:

$$\nu_{obs} = \frac{\nu_{com} + 1}{\nu_{com} + 3} \nu_{com} (1 - \lambda), \quad \nu = \left(\frac{1}{\nu_{old}} + \frac{1}{\nu_{obs}} \right)^{-1}.$$

So ν never exceeds ν_{com} , and it approaches ν_{old} as ν_{com} grows.

ν_{com} is $n - k$, the residual degrees of freedom of the first imputation's model: n analysed patients minus k estimated parameters. That holds for every family, linear, logistic, ordinal, nominal and count, as it does in mice. `nnet::multinom()` reports no residual degrees of freedom, so for a nominal model k is its effective degrees of freedom, again as mice does. Pooling stops with an error when ν_{com} is zero or negative, as it is for a saturated model, because no small-sample degrees of freedom exist then.

Two consequences are worth knowing. First, a single logistic, ordinal, nominal or count fit is tested against the normal distribution (df is Inf), but the same model pooled over imputations reports a finite df and a t test. A single linear fit uses its residual degrees of freedom. Second, SAS PROC MIANALYZE reports ν_{old} unless you give it the complete-data degrees of freedom with EDF=. To reproduce these df, and the p-values and intervals that depend on them, run PROC MIANALYZE with EDF= set to $n - k$.

Barnard J, Rubin DB (1999). Small-sample degrees of freedom with multiple imputation. *Biometrika*, 86(4), 948–955. doi:10.1093/biomet/86.4.948

Rubin DB (1987). *Multiple Imputation for Nonresponse in Surveys*. Wiley. doi:10.1002/9780470316696

References

Leyrat C, Seaman SR, White IR, Douglas I, Smeeth L, Kim J, Resche-Rigon M, Carpenter JR, Williamson EJ (2019). Propensity score analysis with partially observed covariates: how should multiple imputation be used? *Statistical Methods in Medical Research*, 28(1), 3–19. doi:10.1177/0962280217713032

Examples

```
dta <- sample_ps_data(n = 150, seed = 42)
fit <- fit_logistic(
  tavr ~ age + female + ef,
  data      = dta,
  family    = "binary",
  outcome_levels = c(0, 1),
  event_level   = 1
)
fit$tables$estimates[, c("term", "estimate", "odds_ratio")]
head(fit$data[, c("id", "tavr", "prob")])
```

is_ps_data

Test whether an object is a ps_data instance

Description

Test whether an object is a ps_data instance

Usage

```
is_ps_data(x)
```

Arguments

x Any R object.

Value

TRUE if *x* inherits from "ps_data", FALSE otherwise.

Examples

```
dta <- sample_ps_data()
obj <- ps_match(dta)
is_ps_data(obj)    # TRUE
is_ps_data(list()) # FALSE
```

<code>print.ps_data</code>	<i>Print a ps_data object</i>
----------------------------	-------------------------------

Description

Displays a compact summary of the object class, group counts, and the available diagnostic tables.

Usage

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

Arguments

x A ps_data object.

... Ignored.

Value

x, invisibly.

ps_forest

Random-forest propensity score (out-of-bag)

Description

Estimates $P(\text{treated})$ with a random-forest classifier (`randomForestSRC::rfsrc()`) and returns the **out-of-bag** class probability as the score, so no patient is scored by trees that saw them. The result appends the same score columns as `ps_logistic()`, so it feeds `ps_weight()`, `ps_match()` and `sa_overlap()` unchanged, and its `$data` feeds `hvtiPlotR::hv_mirror_hist()`.

Usage

```
ps_forest(
  formula,
  data,
  treatment_col = NULL,
  id_col = "id",
  score_col = "prob_t",
  logit_col = "logit_t",
  weight_col = "mt_wt",
  treated_level = NULL,
  ntree = 500L,
  seed = NULL,
  clip = 0.001,
  ...
)
```

Arguments

<code>formula</code>	Two-sided formula, $\text{treatment} \sim x_1 + x_2 + \dots$. A <code>.</code> on the right-hand side expands to the data's other columns, excluding the treatment and <code>id_col</code> ; terms the formula subtracts, as in <code>. - x3</code> , are left out of the forest. The forest splits on the raw variables, so a transformation such as <code>log(age)</code> or an interaction such as <code>age:ef</code> contributes age and ef as they are.
<code>data</code>	Data frame with no missing values in the formula variables.
<code>treatment_col</code>	Treatment column name; defaults to the formula response. Binary, 0/1 or logical.
<code>id_col</code>	Optional patient identifier column, carried through.
<code>score_col</code> , <code>logit_col</code> , <code>weight_col</code>	Names of the appended score, logit and matching-weight columns.
<code>treated_level</code>	Value of <code>treatment_col</code> that is "treated" (the score is $P(\text{treated})$). Defaults to 1 / TRUE.
<code>ntree</code>	Number of trees.

seed	NULL (default) or one whole number. With a seed, R's random number stream is set to <code>abs(seed)</code> immediately before the fit and <code>randomForestSRC::rfsrc()</code> is given <code>seed = -abs(seed)</code> , the sign it requires; the caller's stream is restored afterwards and the seed is kept in <code>\$meta\$seed</code> . With NULL, <code>rfsrc()</code> takes its seed from the caller's stream, so the scores depend on whatever drew random numbers earlier in the session.
clip	Scores are clipped to <code>[clip, 1 - clip]</code> before the logit and weight are computed. Forest probabilities can be exactly 0 or 1.
...	Further arguments for <code>randomForestSRC::rfsrc()</code> .

Value

A `ps_forest/ps_data` object. `$data` is data plus `score_col`, `logit_col`, `weight_col`, `quintile` and `decile`. `weight_col` holds the Li-Greene matching weight computed as in `ps_logistic()`, $\min(p, 1 - p) / p$ for treated patients and $\min(p, 1 - p) / (1 - p)$ for controls, which targets the ATM estimand (average treatment effect among the matched; Li & Greene, 2013, [doi:10.1515/ijb-20120030](https://doi.org/10.1515/ijb-20120030)); `$meta` holds formula, `treatment_col`, `id_col`, `score_col`, `logit_col`, `weight_col`, `treated_level`, `treatment_levels`, `method` ("forest-oob"), `ntree`, `seed` (the seed used, NA when none was given) and `n_total`; `$tables` holds `smd` and `group_counts` (group, the value in `treatment_col`; a logical `treated`; `n`); `$models$forest` is the fitted forest.

Threads and reproducibility

A seed makes the forest reproducible: seeded this way, fits have matched to the last digit on a 192-thread OpenMP Linux server and a single-threaded Mac. `randomForestSRC` uses every core unless `options(rf.cores = )` says otherwise, and some `randomForestSRC` procedures have not been stable from run to run under OpenMP. When two runs with the same seed disagree, set `options(rf.cores = 1L)` before calling `ps_forest()`; it costs speed and buys a single-threaded, repeatable fit. This function leaves the option alone.

Examples

```
if (requireNamespace("randomForestSRC", quietly = TRUE)) {
  op <- options(rf.cores = 2L)
  dta <- sample_ps_data(n = 150, seed = 1)[, c("id", "tavr", "age", "ef")]
  obj <- ps_forest(tavr ~ age + ef, dta, ntree = 100, seed = 1)
  summary(obj$data$prob_t)
  options(op)
}
```

ps_logistic

Estimate propensity scores via binary logistic regression

Description

Fits `stats::glm(..., family = binomial())` and appends the estimated propensity score, its logit, and a matching weight to the dataset. Supports multiply-imputed data in a stacked "long" format.

Usage

```
ps_logistic(
  formula,
  data,
  treatment_col = NULL,
  id_col = "id",
  imputation_col = NULL,
  score_col = "prob_t",
  logit_col = "logit_t",
  weight_col = "mt_wt",
  covariates = NULL,
  treatment_levels = NULL,
  treated_level = NULL
)
```

Arguments

formula	A formula with the binary treatment on the left-hand side, e.g. <code>tavr ~ age + female + ef + diabetes</code> .
data	A data frame. If <code>imputation_col</code> is set, data must be a stacked "long" MI data frame as produced by <code>mice::complete(mids, action = "long")</code> .
treatment_col	Name of the binary treatment column: 0/1 or logical, or any two values when <code>treated_level</code> or <code>treatment_levels</code> names them. If NULL (default), extracted from the LHS of formula.
id_col	Patient identifier column. Required when <code>imputation_col</code> is set. Default "id".
imputation_col	Name of the imputation-index column in a stacked MI data frame. NULL (default) means a single complete dataset is supplied.
score_col	Output column name for the propensity score (probability of treatment). Default "prob_t".
logit_col	Output column name for the logit of the propensity score, $\log(p / (1 - p))$. Default "logit_t".
weight_col	Output column name for the Li-Greene matching weight (ATM). Default "mt_wt".
covariates	Character vector of covariate column names for SMD balance diagnostics, used as given. If NULL (default), every numeric column that varies is used, except <code>treatment_col</code> , <code>score_col</code> , <code>logit_col</code> , <code>weight_col</code> , <code>id_col</code> and the fixed columns this package appends (<code>quintile</code> , <code>decile</code> , <code>iptw</code> , <code>match</code> , <code>pair_id</code> and the like). A predictor named in formula is always kept.
treatment_levels	Complete binary treatment levels. NULL takes them from the data: the two values present when <code>treated_level</code> is given, otherwise 0/1 or FALSE/TRUE.
treated_level	Level whose probability is the propensity score. When <code>treatment_levels</code> is supplied and this is NULL, its last value is used. Given alone, the treatment may hold any two values (a factor such as "TAVR"/"SAVR", for example) and this names the treated one.

Details

Multiple imputation. When `imputation_col` is supplied, the model is fit separately on each imputed dataset, and each patient's predicted probabilities are averaged across the imputations on the probability scale, not the logit scale. `logit_col` is the logit of that average. Leyrat et al. (2019) call this the "across" approach. It is not Rubin's rules, which pool coefficients and their variances; `$tables$estimates` holds the coefficients pooled that way. The averaging mirrors the SAS template workflow of `PROC LOGISTIC ... BY _IMPUTATION_` followed by `PROC SUMMARY mean=`. `$data` is built from the lowest-numbered imputation, and its covariate values are the ones the `smd` balance table measures. That table reports balance in one imputed dataset, not across all of them.

In Leyrat et al.'s simulations of inverse probability of treatment weighting, only the "within" approach was unbiased in every scenario they studied, and they recommend it with the outcome included in the imputation model. To follow it, call `ps_logistic()` once per imputed dataset, without `imputation_col`; match or weight within each dataset; estimate the treatment effect in each; then pool those estimates and their variances with Rubin's rules.

The matching weight appended as `weight_col` is $\min(p, 1-p) / (p * trt + (1-p) * (1-trt))$, which equals the `mt_wt` column produced in the SAS templates (Li & Greene, 2013). It targets the ATM estimand (average treatment effect among the matched). Neither `ps_match()` nor `ps_weight()` reads it; both work from `score_col`. Pass it as `weight_col` to `ps_stddiff()` or `ps_mw_var()`, or use it as the weight in an outcome model.

A patient with a missing treatment is left out of the model fit, as `stats::glm()` drops the row, and out of the strata, the balance tables and the group counts. The model needs no treatment to predict, so that patient still gets `score_col` and `logit_col`; `weight_col`, quintile and decile are missing. One warning states how many such patients there are, and `$meta$n_missing_treatment` keeps the count. A patient missing a predictor gets no score; `$tables$fit_status` counts those rows.

Value

An object of class `c("ps_logistic", "ps_data")` with:

`$data` data (the lowest-numbered imputation when `imputation_col` is set, without that column) with `score_col`, `logit_col`, `weight_col`, quintile, and decile columns appended. `score_col` is the probability of treated_level. quintile and decile rank `score_col` among patients with a treatment. `weight_col` holds the Li-Greene matching weight, $\min(p, 1-p) / p$ for treated patients and $\min(p, 1-p) / (1-p)$ for controls, which targets the ATM estimand (average treatment effect among the matched; Li & Greene, 2013, doi:10.1515/ijb-20120030).

`$meta` Named list: `formula`, `treatment_col`, `id_col`, `imputation_col`, `score_col`, `logit_col`, `weight_col`, `treatment_levels`, `treated_level`, `bundle_version`, `model_family` ("binary"), `package_versions`, `method` ("logistic" or "logistic-MI"), `n_imputations`, `n_total`, `n_missing_treatment`.

`$tables` Named list: `smd` (variable, `smd`), `group_counts`, and the model tables of `fit_logistic()`: `estimates`, `covariance`, `by_imputation` and `fit_status`. `group_counts` has one row per treatment level, named as declared, with a logical treated flag and the count `n`.

`$models` The fitted `stats::glm()` models, one per imputation, named by imputation index.

References

- Leyrat C, Seaman SR, White IR, Douglas I, Smeeth L, Kim J, Resche-Rigon M, Carpenter JR, Williamson EJ (2019). Propensity score analysis with partially observed covariates: how should multiple imputation be used? *Statistical Methods in Medical Research*, 28(1), 3–19. doi:10.1177/0962280217713032
- Li L, Greene T (2013). A weighting analogue to pair matching in propensity score analysis. *The International Journal of Biostatistics*, 9(2), 215–234. doi:10.1515/ijb20120030

See Also

[ps_match\(\)](#), [ps_weight\(\)](#), [ps_ordinal\(\)](#), [ps_nominal\(\)](#), [fit_logistic\(\)](#), [sample_ps_data\(\)](#)

Examples

```
dta <- sample_ps_data(n = 200, seed = 42)
dta$prob_t <- NULL

# --- Single complete dataset (mirrors tp.lm.logistic_propensity_score.nomi.sas)
# Equivalent to: PROC LOGISTIC data=built descending; model tavr = ...;
obj <- ps_logistic(
  tavr ~ age + female + ef + diabetes + hypertension,
  data = dta
)
print(obj)
summary(obj)

# The function appends:
# prob_t -- propensity score (p-hat; SAS _p_ / _propen_)
# logit_t -- log(p/(1-p)) (SAS _logit_)
# mt_wt -- matching weight (SAS mt_wt = min(p,1-p)/(p*trt+(1-p)*(1-trt)))
# quintile -- rank-based quintile, ceiling(rank * 5 / n)
# decile -- rank-based decile
head(obj$data[, c("id", "tavr", "prob_t", "logit_t", "mt_wt",
  "quintile", "decile")])

# Pass the scored data to ps_match() for downstream matching
matched <- ps_match(obj$data, score_col = obj$meta$score_col, seed = 42)
nrow(matched$data[matched$data$match == 1L, ])

# --- Multiply-imputed data (mirrors tp.lm.logistic_propensity_score.sas)
# Equivalent to:
# PROC LOGISTIC data=built descending; BY _IMPUTATION_; model tavr = ...;
# PROC SUMMARY data=decile; class ccfid; var _p_; output out=... mean=_propen;

# Simulate a stacked MI dataset (2 imputations, column "_Imputation_")
dta_mi <- rbind(
  cbind(dta, `_Imputation_` = 1L),
  cbind(dta, `_Imputation_` = 2L)
)
names(dta_mi)[names(dta_mi) == "_Imputation_"] <- "imp"
```

```
obj_mi <- ps_logistic(
  tavr ~ age + female + ef + diabetes + hypertension,
  data      = dta_mi,
  imputation_col = "imp",
  id_col     = "id"
)
print(obj_mi)
# Per-patient PS is the average across the two imputed-dataset predictions,
# matching the PROC SUMMARY mean= step in the SAS template.
head(obj_mi$data[, c("id", "tavr", "prob_t")])
```

ps_match

*Propensity Score Nearest-Neighbour Matching***Description**

Performs 1:1 nearest-neighbour matching without replacement on a pre-computed propensity score column. Returns a `ps_match` object whose `$data` slot is the **full** original dataset with a match column set to 1 for matched pairs and 0 for unmatched patients.

Usage

```
ps_match(
  data,
  treatment_col = "tavr",
  score_col = "prob_t",
  id_col = "id",
  caliper = NULL,
  covariates = NULL,
  match_col = "match",
  ratio = 1L,
  seed = NULL,
  treated_level = NULL,
  scale = c("probability", "logit"),
  caliper_sd = NULL
)
```

Arguments

<code>data</code>	A data frame. Must contain <code>treatment_col</code> , <code>score_col</code> , and <code>id_col</code> .
<code>treatment_col</code>	Name of the binary treatment column (0/1 or logical). Default "tavr".
<code>score_col</code>	Name of the numeric propensity score column, on the scale named by <code>scale</code> : probabilities in [0, 1] for <code>scale = "probability"</code> , or logits (any finite value, such as the <code>logit_t</code> column <code>ps_logistic()</code> appends) for <code>scale = "logit"</code> . Default "prob_t".

id_col	Name of the patient identifier column. Default "id".
caliper	Maximum allowable difference in score_col, on its own scale, for a match to be accepted. NULL (default) imposes no caliper unless caliper_sd is given. Give at most one of caliper and caliper_sd.
covariates	Character vector of covariate columns for SMD diagnostics, used as given. If NULL (default), every numeric column that varies is used, except treatment_col, score_col, id_col, match_col and the fixed columns this package appends (prob_t, logit_t, mt_wt, iptw, pair_id, quintile, decile, bs, cluster).
match_col	Name of the output match-indicator column appended to \$data. Default "match". If the column already exists it is overwritten.
ratio	Matching ratio (controls : treated). Currently only the number 1 (1:1) is supported. Default 1.
seed	Seed for the random shuffling of treated-patient order that reduces order-dependence. NULL (default) or one whole number. With a seed, the random number stream is set to abs(seed) immediately before the draws, the result is reproducible whatever ran earlier in the session, the caller's stream is restored afterwards, and the seed is kept in \$meta\$seed. With NULL the draws come from the caller's stream and advance it, and \$meta\$seed is NA.
treated_level	The value of treatment_col that is treated. NULL (default) requires a 0/1 or logical column and takes 1 or TRUE. Give a value, for example "transcatheter", to use a column holding any two values.
scale	The scale score_col is on, and so the scale distances and the caliper are measured on: "probability" (default) or "logit". The recommended caliper, 0.2 standard deviations of the logit of the propensity score (Austin 2011), needs scale = "logit".
caliper_sd	Caliper in pooled standard-deviation units of score_col: the caliper applied is caliper_sd * sqrt((var_t + var_c) / 2), where var_t and var_c are the variances of score_col among treated and control patients (missing values dropped). NULL (default) uses caliper instead. scale = "logit", caliper_sd = 0.2 is the Austin (2011) recommendation.

Details

Estimand. Every treated patient looks for a control, so the matched sample targets the average treatment effect on the treated (ATT). A treated patient who finds no control within the caliper, or none at all once the controls run out, is dropped, and the estimand shifts to the treated patients who could be matched, in practice those inside common support. \$meta\$n_unmatched counts them; when it is not zero, say which population the estimate describes.

The \$tables slot contains standardised mean difference (SMD) tables before and after matching, and group counts. hvtiPlotR::hv_mirror_hist() reads \$data directly. hvtiPlotR::hv_balance() needs the two SMD tables stacked into long form with variable, group and std_diff (in percent); the vignette shows the reshape.

A patient with a missing treatment or a missing score cannot be matched. Those rows stay in \$data with match_col set to 0, are left out of the treated and control pools, the SMD tables and the group counts, and one warning states how many there are. \$meta\$n_excluded keeps the count.

Value

An object of class `c("ps_match", "ps_data")` with:

`$data` The full input data frame with `match_col` and `pair_id` appended. `match_col` is 1 for matched patients and 0 otherwise. `pair_id` is an integer identifying which matched pair each patient belongs to (NA for unmatched patients). Filter to `match == 1` to obtain the matched subset.

`$meta` Named list: `treatment_col`, `treated_level` (the value of `treatment_col` taken as treated), `score_col`, `id_col`, `match_col`, `pair_id_col`, `scale`, `caliper` (the absolute caliper applied, on scale), `caliper_sd`, `method`, `seed`, `n_total`, `n_excluded` (rows with a missing treatment or score), `n_matched` (the number of matched pairs, so the number of matched treated patients) and `n_unmatched` (treated patients left without a match; unmatched controls are not counted).

`$tables` Named list: `smd_before`, `smd_after` (variable, `smd`), `group_counts_before`, `group_counts_after` (group, treated, n). Group tables name each group by its value in `treatment_col` (0/1, FALSE/TRUE, or the two values when `treated_level` is given) and carry a logical treated column.

References

Austin PC (2011). Optimal caliper widths for propensity-score matching when estimating differences in means and differences in proportions in observational studies. *Pharmaceutical Statistics*, 10(2), 150–161. doi:10.1002/pst.433

See Also

`ps_weight()`, `sample_ps_data()`, `hvtiPlotR::hv_mirror_hist()`, `hvtiPlotR::hv_balance()`

Examples

```
dta <- sample_ps_data(n = 200, seed = 42)

# 1. Build the matched object
obj <- ps_match(dta)

# 2. Inspect diagnostics
print(obj)
summary(obj)

# 3. Extract the full data (with match indicator)
head(obj$data)
table(obj$data$match)

# 4. Matched subset only
matched <- obj$data[obj$data$match == 1L, ]
nrow(matched)

# 5. Match on the logit with a 0.2 SD caliper (Austin 2011)
dta$logit_t <- stats::qlogis(dta$prob_t)
obj_logit <- ps_match(dta, score_col = "logit_t", scale = "logit", caliper_sd = 0.2)
```

```
obj_logit$meta$caliper
```

ps_mw_var

Bootstrap variance of a matching-weight treatment effect

Description

Estimates the weighted difference in means, group 1 minus group 0, for each outcome, with a bootstrap standard deviation, percentiles, z and p. A port of the CCF %mw_var SAS macro (Rajeswaran 2014), for matching weights as described by Li and Greene (2013).

Usage

```
ps_mw_var(
  data,
  treatment_col = "tavr",
  outcomes,
  weight_col,
  n_rep = 1000L,
  seed = NULL,
  treated_level = NULL
)
```

Arguments

data	A data frame.
treatment_col	Name of the group column, coded 0/1 or logical, with both groups present. Default "tavr".
outcomes	Character vector of numeric outcome columns.
weight_col	Name of the column of matching weights, which must be positive and finite.
n_rep	Number of bootstrap replicates. Default 1000, the macro's RESAMPL=.
seed	NULL (default) or one whole number. With a seed, the random number stream is set to abs(seed) immediately before the draws, the result is reproducible whatever ran earlier in the session, the caller's stream is restored afterwards, and the seed is kept in \$meta\$seed. With NULL the draws come from the caller's stream and advance it, and \$meta\$seed is NA.
treated_level	The value of treatment_col that is treated. NULL (default) requires a 0/1 or logical column and takes 1 or TRUE. Give a value, for example "transcatheter", to use a column holding any two values.

Details

With the Li-Greene matching weight (`mt_wt` from `ps_logistic()` or `ps_forest()`) the estimate targets the ATM, the average treatment effect among the matched: the population that 1:1 matching on the propensity score would keep. With other weights it targets whatever estimand those weights define.

Estimate. For each group, PROC MEANS weighted summaries: the mean $\sum(w \cdot y) / \sum(w)$, the SD with an $n - 1$ denominator, and the sum of weights. The macro labels the sum of weights `N`; it is reported here as `sumwgt_0` and `sumwgt_1` because it is not a count. For a 0/1 outcome the means are weighted proportions.

Bootstrap. Each replicate resamples rows with replacement within each group, to that group's size, and recomputes the weighted means. The treatment group is the only stratum, as in the macro. `sd` is the standard deviation of the replicate differences, and the percentiles use `quantile(type = 4)`, which is SAS `PCTLDEF=1` as the macro sets it. Replicates where an outcome has no non-missing value in a group are left out for that outcome, and `n_rep` counts those that remain.

The weights are held fixed. Each resampled row keeps its original weight; the propensity model and the weights are not re-estimated. This reproduces the macro, and it ignores the uncertainty in estimating the weights, so `sd` can be too small.

Missing values. Rows with a missing group are dropped. A missing outcome is dropped for that outcome only.

Random numbers. With `seed`, the result is reproducible and the caller's random number stream is restored afterwards.

Value

An object of class `c("ps_mw_var", "ps_data")` with:

`$data` The input data frame, unchanged.

`$meta` Named list: `treatment_col`, `weight_col`, `outcomes`, `method`, `n_rep`, `seed`, `n_total`, `n_dropped`.

`$tables` Named list with `mw_var`, one row per outcome: `outcome`, `label`, `sumwgt_0`, `mean_0`, `sd_0`, `sumwgt_1`, `mean_1`, `sd_1`, `estimate`, `n_rep`, `sd`, `p2_5`, `p16`, `p50`, `p84`, `p97_5`, `z` and `p`.

References

Li L, Greene T. A weighting analogue to pair matching in propensity score analysis. The International Journal of Biostatistics. 2013;9(2):215-234. doi:[10.1515/ijb20120030](https://doi.org/10.1515/ijb20120030)

See Also

[ps_weight\(\)](#), [ps_stddiff\(\)](#)

Examples

```
dta <- sample_ps_data(n = 200, seed = 42)
dta$mt_wt <- pmin(dta$prob_t, 1 - dta$prob_t) /
  ifelse(dta$tavr == 1, dta$prob_t, 1 - dta$prob_t)
obj <- ps_mw_var(dta, treatment_col = "tavr", outcomes = c("age", "ef"),
```

```
weight_col = "mt_wt", n_rep = 200, seed = 1)
obj$tables$mw_var
```

ps_nominal	<i>Estimate propensity scores via nominal (generalised logit) logistic regression</i>
------------	---

Description

Fits a multinomial logistic regression model via `nnet::multinom()` for a nominal (unordered) treatment variable, equivalent to SAS PROC LOGISTIC ... / link=glogit. One probability column per treatment level is appended to the dataset.

Usage

```
ps_nominal(
  formula,
  data,
  treatment_col = NULL,
  id_col = "id",
  imputation_col = NULL,
  ref_level = NULL,
  score_col_prefix = "prob",
  trace = FALSE,
  covariates = NULL,
  treatment_levels = NULL,
  smd_pairs = "reference"
)
```

Arguments

formula	A formula with the nominal treatment on the LHS, e.g. <code>rtyp ~ age + female + ef + diabetes</code> .
data	A data frame (single or stacked MI).
treatment_col	Name of the nominal treatment column. NULL (default) extracts from formula.
id_col	Patient identifier column. Required when <code>imputation_col</code> is set. Default "id".
imputation_col	Imputation-index column. NULL (default) means a single complete dataset.
ref_level	Reference level for the multinomial model. NULL (default) uses the first factor level, matching <code>REF=first</code> in SAS.
score_col_prefix	Prefix for per-level output columns (<code>prob_<level></code> by default). Default "prob".
trace	Logical. If FALSE (default), suppresses <code>nnet::multinom()</code> iteration messages.
covariates	Covariate columns for the balance table. NULL (default) uses every numeric column other than the treatment, score, identifier and strata columns.

treatment_levels	Complete nominal treatment levels. NULL preserves the levels inferred by the historical interface.
smd_pairs	Which pairs of treatment levels the balance table compares. "reference" (default) compares each level with ref_level; "all" compares every pair. The levels have no order, so there is no "adjacent" choice.

Details

When `imputation_col` is supplied, models are fit on each imputed dataset and per-level probabilities are averaged per patient across imputations, on the probability scale. This is the "across" approach of Leyrat et al. (2019), not Rubin's rules; `ps_logistic()` describes it and the "within" alternative. `$data` and the `smd` table use the lowest-numbered imputation's covariates.

Value

An object of class `c("ps_nominal", "ps_data")` with:

`$data` `data` (the lowest-numbered imputation when `imputation_col` is set, without that column) with one `<score_col_prefix>_<level>` column per treatment level appended, `ref_level` first. The treatment column is returned as a factor with `ref_level` as its first level.

`$meta` Named list: `formula`, `treatment_col`, `id_col`, `imputation_col`, `score_cols`, `score_col_prefix`, `levels` (in model order, `ref_level` first), `smd_pairs`, `ref_level`, `treatment_levels` (as declared), `bundle_version`, `model_family` ("nominal"), `package_versions`, `method` ("nominal-logistic" or "nominal-logistic-MI"), `n_imputations`, `n_total`.

`$tables` Named list: `smd`, `group_counts` (`group`, `n`), and the model tables of `fit_logistic()`: `estimates`, `covariance`, `by_imputation` and `fit_status`. `smd` has one row per covariate and pair of levels: `variable`, `level`, `versus`, `smd`.

`$models` The fitted `nnet::multinom()` models, one per imputation, named by imputation index.

Package dependency

Requires **nnet** (a recommended R package, usually pre-installed). Install with `install.packages("nnet")` if missing.

References

Leyrat C, Seaman SR, White IR, Douglas I, Smeeth L, Kim J, Resche-Rigon M, Carpenter JR, Williamson EJ (2019). Propensity score analysis with partially observed covariates: how should multiple imputation be used? *Statistical Methods in Medical Research*, 28(1), 3–19. doi:10.1177/0962280217713032

See Also

`ps_logistic()`, `ps_ordinal()`, `ps_match()`

Examples

```
# Mirrors the SAS template workflow:
# PROC LOGISTIC ... / link=glogit (generalised logit)
# The first factor level is used as the reference category, matching
# SAS REF=first. Change ref_level to match a different reference.

if (requireNamespace("nnet", quietly = TRUE)) {
  dta <- sample_ps_data_nominal(n = 300, seed = 42)
  obj <- ps_nominal(
    rtyp ~ age + female + ef + diabetes,
    data = dta
  )
  print(obj)

  # One probability column per treatment level (analogous to p_cos, p_per,
  # p_dev, p_ce from the PROC TRANSPOSE step in the SAS template).
  head(obj$data[, c("id", "rtyp", "prob_COS", "prob_PER", "prob_DEV", "prob_CE")])

  # Explicitly set a different reference level
  obj_ce <- ps_nominal(
    rtyp ~ age + female + ef + diabetes,
    data = dta,
    ref_level = "CE" # matches REF=last in SAS
  )
  obj_ce$meta$ref_level
}
```

ps_ordinal

Estimate propensity scores via ordinal (cumulative logit) logistic regression

Description

Fits a proportional-odds cumulative logit model via `MASS::polr()` for an ordered treatment variable and appends one probability column per treatment level to the dataset. The model is equivalent to the default PROC LOGISTIC behaviour for ordinal responses in SAS, standard errors included: they come from the expected information, as `fit_logistic()` describes under "Ordinal standard errors".

Usage

```
ps_ordinal(
  formula,
  data,
  treatment_col = NULL,
  id_col = "id",
  imputation_col = NULL,
```

```

score_col_prefix = "prob",
covariates = NULL,
treatment_levels = NULL,
smd_pairs = "reference"
)

```

Arguments

formula	A formula with the ordered factor treatment on the LHS, e.g. <code>nyha_grp ~ age + female + ef</code> . The treatment must be an ordered factor, whose level order is the order the model uses, unless <code>treatment_levels</code> gives that order. Any other treatment is an error: alphabetical order is not a safe guess for an ordinal scale.
data	A data frame (single or stacked MI).
treatment_col	Name of the ordinal treatment column. <code>NULL</code> (default) extracts from formula.
id_col	Patient identifier column. Required when <code>imputation_col</code> is set. Default <code>"id"</code> .
imputation_col	Imputation-index column for stacked MI data. <code>NULL</code> (default) assumes a single complete dataset.
score_col_prefix	Prefix for the per-level output columns. Columns are named <code><prefix>_<level></code> for each level of the treatment. Default <code>"prob"</code> .
covariates	Covariate columns for the balance table. <code>NULL</code> (default) uses every numeric column other than the treatment, score, identifier and strata columns.
treatment_levels	Complete treatment levels, lowest first. <code>NULL</code> (default) takes the levels of the treatment, which must then be an ordered factor.
smd_pairs	Which pairs of treatment levels the balance table compares. <code>"reference"</code> (default) compares each level with the first level; <code>"adjacent"</code> compares each level with the one below it; <code>"all"</code> compares every pair. Give more than one to combine them, for example <code>c("reference", "adjacent")</code> .

Details

When `imputation_col` is supplied, models are fit on each imputed dataset separately and per-level probabilities are averaged per patient across imputations, on the probability scale. This is the "across" approach of Leyrat et al. (2019), not Rubin's rules; `ps_logistic()` describes it and the "within" alternative. `$data` and the `smd` table use the lowest-numbered imputation's covariates.

Value

An object of class `c("ps_ordinal", "ps_data")` with:

`$data` data (the lowest-numbered imputation when `imputation_col` is set, without that column) with one `<score_col_prefix>_<level>` column per treatment level, the marginal probability of that level, and quintile and decile, which rank the probability of the highest level. The treatment column is returned as an ordered factor.

```
$meta Named list: formula, treatment_col, id_col, imputation_col, score_cols, score_col_prefix,
  levels, smd_pairs, treatment_levels, bundle_version, model_family("ordinal"), package_versions,
  cumulative_direction, method("ordinal-logistic" or "ordinal-logistic-MI"), n_imputations,
  n_total.

$tables Named list: smd, group_counts (group, n), and the model tables of fit_logistic():
  estimates, covariance, by_imputation and fit_status. smd has one row per covariate
  and pair of levels: variable, level, versus, smd.

$models The fitted models, one per imputation, named by imputation index: MASS::polr() fits,
  or stats::glm() fits for a two-level treatment.
```

Package dependency

Requires the **MASS** package (a recommended package shipped with base R). Install with `install.packages("MASS")` if missing.

References

Leyrat C, Seaman SR, White IR, Douglas I, Smeeth L, Kim J, Resche-Rigon M, Carpenter JR, Williamson EJ (2019). Propensity score analysis with partially observed covariates: how should multiple imputation be used? *Statistical Methods in Medical Research*, 28(1), 3–19. doi:[10.1177/0962280217713032](https://doi.org/10.1177/0962280217713032)

See Also

`ps_logistic()`, `ps_nominal()`, `ps_match()`

Examples

```
# Mirrors the SAS template workflow:
# PROC LOGISTIC (cumulative logit, default for ordinal response)
# followed by decomposition of cumulative to marginal probabilities
# and rank-based quintile / decile assignment.

if (requireNamespace("MASS", quietly = TRUE)) {
  dta <- sample_ps_data_ordinal(n = 300, seed = 42)
  obj <- ps_ordinal(
    nyha_grp ~ age + female + ef + diabetes,
    data = dta
  )
  print(obj)

  # Each level gets its own probability column (marginal, not cumulative).
  # The SAS template computes: p1=col1; p2=col2-col1; p3=1-col2.
  # ps_ordinal() performs this decomposition internally.
  head(obj$data[, c("id", "nyha_grp", "prob_I", "prob_II", "prob_III")])

  # Quintile and decile columns are appended, ordered by p(highest level).
  table(obj$data$quintile)
}
```

ps_rmst

*Observed-outcome restricted mean survival contrast, weighted***Description**

Design-based comparison of restricted mean survival time (RMST) between treated and control patients using only the observed follow-up: weighted Kaplan-Meier curves (weights from the propensity score) integrated to tau, differenced, with a stratified bootstrap interval.

Usage

```
ps_rmst(
  x,
  time_col,
  event_col,
  tau,
  weights = c("unweighted", "ato"),
  subsets = list(),
  n_boot = 200L,
  refit = FALSE,
  seed = NULL,
  time_unit_days = 365.2425,
  clip = 0.001
)
```

Arguments

x	A <code>ps_data</code> object with a score and treatment column. A <code>ps_match()</code> object is analysed on its matched pairs only, and a <code>ps_weight()</code> object contributes its stored weight column; see Details. A score on the logit scale (scale = "logit", as a <code>ps_match()</code> object can carry) is back-transformed with <code>stats::plogis()</code> before any weights are built.
time_col, event_col	Follow-up time and event indicator (0/1) columns of <code>x\$data</code> .
tau	Horizon, in the units of <code>time_col</code> . It must not exceed the largest observed time among the rows that enter either arm's curve, for any analysed subset and weighting.
weights	Weighting schemes to run: any of "unweighted", "ato" (overlap), "att", "ate". For a <code>ps_match()</code> input the default is "unweighted" alone, and no other scheme is accepted.
subsets	Optional named list of logical vectors (one value per row); each is run as a restricted analysis (for example a common-support subset). The full data is always included as "all".
n_boot	Bootstrap draws. Resampling is by matched pair for a <code>ps_match()</code> input and by row, stratified by arm, otherwise.

refit	If FALSE (default) the score is held fixed in the bootstrap, so the intervals ignore the uncertainty in estimating the propensity score and understate the total uncertainty. If TRUE the score model is refit on each draw (supported for ps_logistic and ps_forest inputs).
seed	Bootstrap seed: NULL (default) or one whole number. With a seed, the random number stream is set to abs(seed) immediately before the draws, the result is reproducible whatever ran earlier in the session, the caller's stream is restored afterwards, and the seed is kept in \$meta\$seed. With NULL the draws come from the caller's stream and advance it, and \$meta\$seed is NA.
time_unit_days	Days per unit of time_col, for the diff_days column. Default 365.2425 (time in years).
clip	Scores are clipped to [clip, 1 - clip] before weighting.

Details

Weights. "unweighted" gives every row weight 1. "ato" weights treated rows by $1 - p$ and controls by p , targeting the overlap population; "att" weights controls by $p / (1 - p)$, targeting the treated; "ate" weights by $1 / p$ and $1 / (1 - p)$, targeting the whole cohort. Here p is the score clipped to [clip, 1 - clip].

Matched input. For a `ps_match()` object only the rows with `match_col == 1` are analysed, so n is twice the number of matched pairs, and the estimand is the one the match targets. Weighting a matched set by the score adjusts twice for the same confounders, so only "unweighted" is accepted. The bootstrap resamples whole pairs: it draws `pair_id` values with replacement, as many as there are pairs, and takes both members of each drawn pair, so a pair drawn twice enters twice. This keeps the within-pair correlation that a row bootstrap ignores. A subset that splits a pair keeps the member it contains, which is then resampled alone.

Weighted input. For a `ps_weight()` object, the scheme that matches its estimand ("ate" for `estimand = "ATE"`, "att" for `"ATT"`) uses the stored column `x$meta$weight_col` instead of re-computing weights from the score, so any trimming is kept. Stabilisation multiplies each arm by a constant, which leaves each arm's Kaplan-Meier curve and effective sample size unchanged. The other schemes are computed from the score, and an `estimand = "ATC"` object has no matching scheme. In the bootstrap each row keeps its stored weight.

Horizon. The weighted Kaplan-Meier curve is not defined past the last observed time in an arm, counting only the rows that enter the curve (time and event present, weight above zero), so a `tau` beyond it stops with an error rather than carrying the last survival value forward. A bootstrap draw can still end short of `tau`, and its curve is then carried flat to `tau`.

Failures. A bootstrap draw that errors, or that leaves an arm empty, gives NA; the interval is taken over the other draws and the number of NA draws is reported as `n_failed`.

Value

A `ps_rmst / ps_data` object. `$tables$estimates` has one row per subset-and-weighting estimator: `estimator`, `subset`, `weighting`, `n` (rows analysed; for a matched input, twice the matched pairs), `ess_treated`, `ess_control`, `rmst_treated`, `rmst_control`, `diff` (treated minus control, positive favours treated), `diff_days`, `lo_days`, `hi_days` (2.5 and 97.5 percent bootstrap) and `n_failed` (bootstrap draws that gave no estimate). `$tables$curves` has the weighted Kaplan-Meier steps (`estimator`, `arm`, `time`, `surv`) for plotting; `$data` is `x$data` plus one weight column

`w_<weighting>` per scheme. `$meta$seed` is the seed used (NA when none was given). `$meta$scale` is the scale of `score_col`, carried from `x` ("probability" when `x` records none). `$meta$resampling` is "pair" for a matched input and "row" (stratified by arm) otherwise, and `$meta$stored_weighting` names the scheme that used a `ps_weight()` object's stored weights (NA when none did).

Examples

```
if (requireNamespace("survival", quietly = TRUE)) {
  dta <- sample_ps_data(n = 200, seed = 5)[, c("id", "tavr", "age", "ef")]
  dta$t <- stats::rexp(nrow(dta), 0.2)
  dta$e <- as.integer(dta$t < 5)
  dta$t <- pmin(dta$t, 5)
  res <- ps_rmst(ps_logistic(tavr ~ age + ef, dta), "t", "e", tau = 4, n_boot = 20,
                seed = 1024)
  res$tables$estimates
}
```

ps_stddiff

Standardized differences between two groups

Description

Computes the standardized difference, group 1 minus group 0, for each named variable, choosing the formula by the variable's type. A port of the CCF %stddiff SAS macro (Artis 2019, from Yang 2012), and intended to reproduce its `est.stddiff` output.

Usage

```
ps_stddiff(
  data,
  treatment_col = "tavr",
  gaussian = NULL,
  nong_ord = NULL,
  binary = NULL,
  categorical = NULL,
  weight_col = NULL,
  treated_level = NULL
)
```

Arguments

<code>data</code>	A data frame.
<code>treatment_col</code>	Name of the group column, coded 0/1 or logical, with both groups present. Default "tavr".
<code>gaussian</code> , <code>nong_ord</code> , <code>binary</code> , <code>categorical</code>	Character vectors of column names, by type. See Details. At least one variable is required, and a variable may appear under one type only. A binary column must be coded 0/1 or logical; a categorical column may have any coding.

weight_col	Optional name of a column of positive weights, for example the inverse-probability weights from <code>ps_weight()</code> or the matching weight <code>mt_wt</code> from <code>ps_logistic()</code> . Weights must be finite.
treated_level	The value of <code>treatment_col</code> that is treated. NULL (default) requires a 0/1 or logical column and takes 1 or TRUE. Give a value, for example "transcatheter", to use a column holding any two values.

Details

Variable types. The type decides the formula, so it is given, not inferred: an ordinal NYHA class stored as an integer would otherwise be treated as Gaussian.

- **gaussian:** difference in means over $\sqrt{(\text{var1} + \text{var0}) / 2}$. The denominator averages the two variances; it does not pool them by sample size, so it differs from a Cohen's d when the groups are unequal.
- **nong_ord:** the same formula on ranks of the pooled sample, ties averaged, as PROC RANK does by default. Ranks are unweighted; only their means and variances use `weight_col`.
- **binary and categorical** are routed by the number of levels observed, as the macro routes its `BINARY=` and `CATG=` lists: one level gives 0, two use the binary formula on the proportion at the last level in sort order (1 for a 0/1 variable, the last level for a factor), and more than two use the Mahalanobis form of Yang and Dalton (2012). For that form the last level in sort order is the one dropped, and S has diagonal $(t_i (1 - t_i) + c_i (1 - c_i)) / 2$ and off-diagonal $-(t_i t_j + c_i c_j) / 2$. A categorical difference is never negative. When S is singular, as when the two groups share no levels, the difference is NA with a warning naming the variable.

Weights. Within each group, the weights are first divided by their mean in that group, so that they average 1 there. Means are then $\sum(w \cdot x) / \sum(w)$ and variances $\sum(w \cdot (x - m)^2) / (n - 1)$, which is what PROC MEANS reports with a WEIGHT statement and VARDEF=DF. The result is therefore the same when every weight in a group is multiplied by one constant, so stabilised and unstabilised weights from `ps_weight()` give the same differences. It equals the `%stddiff` macro's output when the weights already average 1 in each group, and otherwise does not: the macro's Gaussian and rank differences change with the scale of the weights. Proportions, for binary and categorical, are $\sum(w \cdot x) / \sum(w)$ within a group and never depended on that scale. Weights must be positive.

Missing values. Rows with a missing group are dropped. Otherwise a missing value is dropped for that variable only, and a variable with no non-missing values gets NA.

Units. The result is a proportion of a standard deviation, not a percent. The SAS output labels the column "(%)" but does not multiply by 100.

Value

An object of class `c("ps_stddiff", "ps_data")` with:

`$data` The input data frame, unchanged.

`$meta` Named list: `treatment_col`, `weight_col`, `variables` (a list by type), `method`, `n_total`, `n_dropped`.

`$tables` Named list with `stddiff`, a data frame with one row per variable in the order given: `variable`, `label` (the column's "label" attribute, or NA), `type` ("gaussian", "nong_ord", "binary", "categorical" or "onelevel") and `stddiff`, which is NA when the pooled variance is zero.

References

Yang D, Dalton JE. A unified approach to measuring the effect size between two groups using SAS. SAS Global Forum 2012, paper 335-2012.

See Also

[ps_match\(\)](#), [ps_weight\(\)](#)

Examples

```
dta <- sample_ps_data(n = 200, seed = 42)
obj <- ps_stddiff(dta, treatment_col = "tavr",
                 gaussian = c("age", "ef"), binary = c("female", "diabetes"))
obj$tables$stdiff
```

ps_stddiff_perm

Permutation reference for standardized differences

Description

Computes [ps_stddiff\(\)](#) on the observed groups and on `n_perm` random permutations of the group labels, and reports the observed value beside percentiles of its permutation distribution. A port of the CCF %stdiffci SAS macro (Artis 2020).

Usage

```
ps_stddiff_perm(
  data,
  treatment_col = "tavr",
  gaussian = NULL,
  nong_ord = NULL,
  binary = NULL,
  categorical = NULL,
  weight_col = NULL,
  reweight = NULL,
  n_perm = 1000L,
  seed = NULL,
  treated_level = NULL
)
```

Arguments

<code>data</code>	A data frame.
<code>treatment_col</code>	Name of the group column, coded 0/1 or logical, with both groups present. Default "tavr".

gaussian, nong_ord, binary, categorical	Character vectors of column names, by type. See Details. At least one variable is required, and a variable may appear under one type only. A binary column must be coded 0/1 or logical; a categorical column may have any coding.
weight_col	Optional name of the column <code>ps_stddiff()</code> weights by. Requires <code>reweight</code> , which supplies its values.
reweight	A function of a data frame returning one positive, finite weight per row, for example one that refits a propensity model and returns matching weights. Required when <code>weight_col</code> is given, and rejected without it.
n_perm	Number of permutations. Default 1000, the macro's.
seed	NULL (default) or one whole number. With a seed, the random number stream is set to <code>abs(seed)</code> immediately before the draws, the result is reproducible whatever ran earlier in the session, the caller's stream is restored afterwards, and the seed is kept in <code>\$meta\$seed</code> . With NULL the draws come from the caller's stream and advance it, and <code>\$meta\$seed</code> is NA.
treated_level	The value of <code>treatment_col</code> that is treated. NULL (default) requires a 0/1 or logical column and takes 1 or TRUE. Give a value, for example "transcatheter", to use a column holding any two values.

Details

What the percentiles are. They describe the standardized difference you would see if group membership carried no information about the variable. They are not a confidence interval for the observed value, whatever the SAS macro's name suggests: an observed difference outside the 2.5 to 97.5 range is larger than label shuffling alone tends to produce. The 16 and 84 percentiles bound the central 68%, the range the macro's 2021 correction fixed.

Percentile definition. `quantile(type = 2)`, which is SAS `PCTLDEF=5`, the PROC UNIVARIATE default the macro relies on. Permutations where a variable's difference is NA are left out of that variable's percentiles.

Weights. A matching or IPTW weight depends on the group, so a weight carried over unchanged onto permuted groups describes neither the observed design nor the null. `reweight` must therefore be given whenever `weight_col` is: a function taking a data frame and returning one weight per row. It is called once on the unpermuted data, for the observed row, and once on each permuted data set, and its result is written to `weight_col` before `ps_stddiff()` runs. The macro instead expects the permuted weights to exist as columns before it is called.

Missing groups. Rows with a missing group keep it and are not permuted into either group.

Random numbers. With seed, the permutations are reproducible and the caller's random number stream is restored afterwards. Without it, the current stream is used and advanced.

Value

An object of class `c("ps_stddiff_perm", "ps_data")` with:

`$data` The input data frame, unchanged.

`$meta` Named list: `treatment_col`, `weight_col`, `variables`, `method`, `n_perm`, `seed`, `n_total`.

`$tables` Named list with `stddiff_perm`, one row per variable: `variable`, `label`, `type`, `observed` (as `ps_stddiff()` reports it) and the permutation percentiles `p2_5`, `p16`, `p50`, `p84`, `p97_5`.

See Also[ps_stddiff\(\)](#)**Examples**

```
dta <- sample_ps_data(n = 200, seed = 42)
obj <- ps_stddiff_perm(dta, treatment_col = "tavr", gaussian = c("age", "ef"),
                      binary = "female", n_perm = 200, seed = 1)
obj$tables$stddiff_perm
```

ps_support	<i>Compare support (overlap) rules</i>
------------	--

Description

Flags patients with weak support under several rules and measures how well the rules agree. Built-in rules use the propensity score of a `ps_data` object; any other rule (for example an isolation-forest tail flag) can be supplied as a logical vector.

Usage

```
ps_support(x, flags = list(), trim = c(0.1, 0.9), common = TRUE)
```

Arguments

<code>x</code>	A <code>ps_data</code> object with a score and treatment column. The score of a ps_match() object built with <code>scale = "logit"</code> is a logit, and is back-transformed with <code>stats::plogis()</code> , so <code>trim</code> and the common-support interval are on the probability scale.
<code>flags</code>	Optional named list of extra logical vectors, one element per row of <code>x\$data</code> , TRUE meaning the rule would drop that patient. The names "trim" and "common" belong to the built-in rules and are refused.
<code>trim</code>	Length-2 numeric, or NULL. Adds the rule "score outside [<code>trim[1]</code> , <code>trim[2]</code> "]". Both values must lie in [0, 1], increasing. Default <code>c(0.1, 0.9)</code> .
<code>common</code>	Logical; add the rule "score outside the range occupied by both groups" (common support).

Details

Trimming changes the estimand. Dropping the patients a rule flags leaves a population defined by the rule: an effect estimated after trimming at `c(0.1, 0.9)`, or after restricting to common support, describes the patients with adequate overlap, not the full sample, the treated or the controls. Name that population when you report it. This function only flags patients; it drops nobody.

A patient with a missing treatment or a missing score cannot be judged. Those rows are left out of the common-support interval and every rule, get a missing `weak_<rule>` flag, and one warning states how many there are.

Value

A `ps_support / ps_data` object. `$data` is `x$data` plus one logical column `weak_<rule>` per rule. `$tables$counts` gives the number flagged per rule (`n_weak`), the number of rows that rule judged (`n`) and the number it did not (`n_excluded`): rows with a missing treatment or score, plus any missing values in a rule supplied through flags. `$meta$n_excluded` counts the rows with a missing treatment or score; `$tables$agreement` gives, for every pair of rules, `n_both`, `n_neither`, `pct_agree`, Cohen kappa and jaccard (overlap of the two dropped sets). `$meta$common_support` is the interval used, on the probability scale. `$meta$scale` is the scale of `score_col`, carried from `x` ("probability" when `x` records none).

Examples

```
dta <- sample_ps_data(n = 200, seed = 3)[, c("id", "tavr", "age", "ef")]
sup <- ps_support(ps_logistic(tavr ~ age + ef, dta))
sup$tables$agreement
```

ps_weight

Inverse-Probability-of-Treatment Weighting (IPTW)

Description

Computes IPTW weights from a pre-computed propensity score column and appends them to the dataset. `estimand` picks the target: the average treatment effect in the whole sample (ATE, the default), in the treated (ATT) or in the controls (ATC). For the matched population (ATM), use the `mt_wt` column from `ps_logistic()` or `ps_forest()` instead. Returns a `ps_weight` object whose `$tables` slot contains SMD tables (unweighted and weighted) and effective-N diagnostics. Its `$data` feeds `hvtiPlotR::hv_mirror_hist()` in weighted mode; `hvtiPlotR::hv_balance()` needs the SMD tables stacked into long form (see the vignette).

Usage

```
ps_weight(
  data,
  treatment_col = "tavr",
  score_col = "prob_t",
  estimand = c("ATE", "ATT", "ATC"),
  stabilise = TRUE,
  trim = NULL,
  covariates = NULL,
  weight_col = "iptw",
  treated_level = NULL,
  id_col = "id"
)
```

Arguments

<code>data</code>	A data frame. Must contain <code>treatment_col</code> and <code>score_col</code> .
<code>treatment_col</code>	Name of the binary treatment column (0/1 or logical). Default <code>"tavr"</code> .
<code>score_col</code>	Name of the numeric propensity score column (values in [0, 1]). Default <code>"prob_t"</code> .
<code>estimand</code>	Causal estimand. One of: • <code>"ATE"</code> (default): average treatment effect. Treated weights = $1/ps$; control weights = $1/(1-ps)$. • <code>"ATT"</code>: average treatment effect on the treated. Treated weights = 1; control weights = $ps/(1-ps)$. • <code>"ATC"</code>: average treatment effect on the controls. Treated weights = $(1-ps)/ps$; control weights = 1.
<code>stabilise</code>	Logical. If TRUE (default) and <code>estimand = "ATE"</code> , each weight is multiplied by the marginal probability of the patient's own arm ($P(\text{treated})$ or $P(\text{control})$). Stabilisation is defined for the ATE only: for <code>"ATT"</code> and <code>"ATC"</code> it has no effect, so the reference arm keeps weights of exactly 1. <code>\$meta\$stabilised</code> records whether it was applied.
<code>trim</code>	Optional numeric in (0, 0.5). If supplied, weights outside the (<code>trim</code> , $1 - \text{trim}$) quantile range are winsorised. Default NULL (no trimming).
<code>covariates</code>	Character vector of covariate columns for SMD diagnostics, used as given. If NULL (default), every numeric column that varies is used, except <code>treatment_col</code> , <code>score_col</code> , <code>weight_col</code> , <code>id_col</code> and the fixed columns this package appends (<code>prob_t</code> , <code>logit_t</code> , <code>mt_wt</code> , <code>iptw</code> , <code>match</code> , <code>pair_id</code> , <code>quintile</code> , <code>decile</code> , <code>bs</code> , <code>cluster</code>).
<code>weight_col</code>	Name of the output weight column appended to <code>\$data</code> . Default <code>"iptw"</code> . If the column already exists it is overwritten.
<code>treated_level</code>	The value of <code>treatment_col</code> that is treated. NULL (default) requires a 0/1 or logical column and takes 1 or TRUE. Give a value, for example <code>"transcatheter"</code> , to use a column holding any two values.
<code>id_col</code>	The patient identifier column, left out of the default covariates. Default <code>"id"</code> ; it need not exist in data.

Details

A patient with a missing treatment or a missing score cannot be weighted. Those rows stay in `$data` with a missing `weight_col`, are left out of the stabilising treatment proportion, the SMD tables, the group counts and the effective sample sizes, and one warning states how many there are. `$meta$n_excluded` keeps the count.

Value

An object of class `c("ps_weight", "ps_data")` with:

`$data` The input data frame with `weight_col` appended.

`$meta` Named list: `treatment_col`, `treated_level` (the value of `treatment_col` taken as treated), `score_col`, `weight_col`, `estimand`, `stabilised`, `trim`, `method`, `n_total`, `n_excluded` (rows with a missing treatment or score).

`$tables` Named list: `smd_unweighted`, `smd_weighted` (variable, `smd`), `group_counts` (group, treated, n) and `effective_n` (group, treated, `n_effective`, the Kish effective sample size). Group tables name each group by its value in `treatment_col` (0/1, FALSE/TRUE, or the two values when `treated_level` is given) and carry a logical `treated` column.

See Also

`ps_match()`, `sample_ps_data()`, `hvtiPlotR::hv_mirror_hist()`, `hvtiPlotR::hv_balance()`

Examples

```
dta <- sample_ps_data(n = 200, seed = 42)

# 1. ATE weights (default)
obj <- ps_weight(dta)
print(obj)
summary(obj)

# 2. ATT weights: treated patients keep a weight of 1
obj_att <- ps_weight(dta, estimand = "ATT")
summary(obj_att)

# 3. Extract data with weights appended
head(obj$data[, c("id", "tavr", "prob_t", "iptw")])
```

sa_evalue

E-Value Sensitivity Analysis

Description

Computes the E-value for a point estimate and optionally for a confidence interval bound, following VanderWeele & Ding (2017). The E-value is the minimum strength of association (on the risk-ratio scale) that an unmeasured confounder would need with **both** the treatment and the outcome, conditional on measured covariates, to fully explain away the observed association.

Usage

```
sa_evalue(
  estimate,
  ci_lo = NULL,
  ci_hi = NULL,
  type = c("RR", "OR", "HR", "IRR", "RD"),
  p0 = NULL,
  rare = NULL
)
```

Arguments

estimate	Numeric. The point estimate (RR, OR, HR, or RD on the original scale). Must be > 0 for RR/OR/HR; any finite value for RD.
ci_lo	Optional numeric. Lower confidence interval bound.
ci_hi	Optional numeric. Upper confidence interval bound. Both ci_lo and ci_hi must be supplied together to compute the CI E-value, and must satisfy $ci_lo \leq estimate \leq ci_hi$.
type	One of "RR" (default), "OR", "HR", "IRR", or "RD". "IRR" (incidence rate ratio) uses the same formula as "RR"; it is provided as a distinct value for clarity in count-outcome studies.
p0	Baseline outcome probability in the unexposed group. Required when type = "RD", and used only then. Must be in (0, 1).
rare	TRUE or FALSE; required when type = "OR" or "HR". Whether the outcome is rare in the study population (prevalence under about 15% at the end of follow-up). TRUE uses the OR or HR as the RR; FALSE applies the common-outcome conversion. The default NULL aborts for OR and HR. Ignored for other types.

Details

Odds ratios and hazard ratios are converted to the risk-ratio scale before the E-value formula is applied, and the conversion depends on whether the outcome is rare (VanderWeele & Ding 2017; VanderWeele 2017):

- **Rare outcome** (prevalence at most about 15%): the OR or HR approximates the RR and is used directly.
- **Common outcome**: $RR = \sqrt{OR}$ for an odds ratio, and $RR = (1 - 0.5^{\sqrt{HR}}) / (1 - 0.5^{\sqrt{1/HR}})$ for a hazard ratio.

There is no default for this choice: for type = "OR" or "HR" the caller must set rare. Rarity refers to the outcome's prevalence in the study population (under about 15% at the end of follow-up), and it cannot be inferred from the baseline risk p_0 alone: with $OR = 4$ and $p_0 = 0.10$ the exposed risk is about 0.31, and reversing the exposure coding would flip any rule based on p_0 . The same conversion is applied to the confidence interval bounds.

For risk differences a baseline outcome probability p_0 must be supplied, and the conversion $RR = (RD + p_0) / p_0$ treats p_0 as fixed: its own sampling uncertainty is not carried into the CI E-value.

Value

An object of class `c("sa_evalue", "ps_data")`. There is no input data, so `$data` is an empty data frame and `$models` an empty list.

`$meta$evalue_estimate` E-value for the point estimate.

`$meta$evalue_ci` E-value for the confidence interval bound closest to the null. NA if ci_lo/ci_hi are not supplied. 1 if the CI already crosses the null.

`$meta$rr_estimate` The point estimate converted to the risk-ratio scale (same as estimate when type = "RR").

`$meta$type` The measure type supplied.

`$meta$rare` For type = "OR" or "HR", the rare-outcome choice used for the conversion; NA for other types.

`$meta$estimate`, `$meta$ci_lo`, `$meta$ci_hi`, `$meta$p0` The inputs, NA where not supplied.

`$tables$value` Data frame with one row for the estimate and one for the CI bound closest to the null: quantity, value on the original scale, rr on the risk-ratio scale (NA when the CI crosses the null) and eval.

References

VanderWeele, T. J., & Ding, P. (2017). Sensitivity analysis in observational research: Introducing the E-value. *Annals of Internal Medicine*, 167(4), 268-274. doi:10.7326/M162607

VanderWeele, T. J. (2017). On a square-root transformation of the odds ratio for a common outcome. *Epidemiology*, 28(6), e58-e60. doi:10.1097/EDE.0000000000000733

See Also

[sa_rosenbaum\(\)](#), [sa_overlap\(\)](#), [sa_trim_sweep\(\)](#)

Examples

```
# Risk ratio of 2.5 -- how robust is the result?
sa_value(2.5, ci_lo = 1.8, ci_hi = 3.5)

# Odds ratio of 0.6 (protective effect) for a common outcome
sa_value(0.6, ci_lo = 0.4, ci_hi = 0.9, type = "OR", rare = FALSE)

# Hazard ratio of 1.8 for an outcome under 15% at end of follow-up
sa_value(1.8, ci_lo = 1.2, ci_hi = 2.7, type = "HR", rare = TRUE)

# Risk difference of 0.10 with baseline probability 0.20
ev <- sa_value(0.10, ci_lo = 0.02, ci_hi = 0.18, type = "RD", p0 = 0.20)
ev$meta$estimate
ev$tables$value
```

Description

Examines the propensity score (PS) distribution for potential positivity violations – regions of covariate space where the probability of receiving one treatment is near zero – and quantifies the common support region.

Usage

```
sa_overlap(
  x,
  score_col = "prob_t",
  treatment_col = "tavr",
  trim_threshold = 0.05,
  treated_level = NULL
)
```

Arguments

<code>x</code>	A <code>ps_data</code> object or a plain data frame.
<code>score_col</code>	Name of the propensity score column. Taken from <code>x\$meta\$score_col</code> when <code>x</code> is a <code>ps_data</code> object. Default <code>"prob_t"</code> . The score of a <code>ps_match()</code> object built with <code>scale = "logit"</code> is a logit, and is back-transformed with <code>stats::plogis()</code> before use.
<code>treatment_col</code>	Name of the binary treatment column (0/1). Taken from <code>x\$meta\$treatment_col</code> when <code>x</code> is a <code>ps_data</code> object. Default <code>"tavr"</code> .
<code>trim_threshold</code>	Numeric in (0, 0.5). PS values below this threshold or above $1 - \text{trim_threshold}$ are flagged as near-positivity violations. Default <code>0.05</code> .
<code>treated_level</code>	The value of <code>treatment_col</code> that is treated. Read from the metadata of a <code>ps_data</code> object when it records one. <code>NULL</code> (default) requires a 0/1 or logical column and takes 1 or <code>TRUE</code> .

Details

The function reports:

- PS summary statistics by treatment group.
- The empirical overlap region: $[\max(\min_treated, \min_control), \min(\max_treated, \max_control)]$.
- The proportion of each group whose PS lies outside the overlap region.
- The number and proportion of patients with PS below `trim_threshold` or above $1 - \text{trim_threshold}$ (near-positivity-violation flags).

For a `ps_data` input, a supplied `score_col`, `treatment_col` or `treated_level` must agree with the object's metadata; a conflict aborts. Arguments not supplied are read from the metadata.

Value

An object of class `c("sa_overlap", "ps_data")`:

`$data` The input data frame (`x$data` for a `ps_data` object), unchanged.

`$meta` `score_col`, `treatment_col`, `treated_level`, `scale`, `trim_threshold`, `n_total`, `n_score_missing`, `n_used` (patients with a score and a treatment), and `overlap_region`: a named numeric vector of the lower and upper bounds of the empirical common support region.

`$tables$summary` Data frame: one row per treatment group with `group` (the value in `treatment_col`), a logical `treated`, `n`, `ps_min`, `ps_q25`, `ps_median`, `ps_mean`, `ps_q75`, `ps_max`.

`$tables$outside_overlap` Data frame: group, treated, n_outside, pct_outside – patients whose PS is outside the overlap region.

`$tables$positivity_flags` Data frame: group, treated, n_near_zero (PS < trim_threshold), n_near_one (PS > 1 - trim_threshold), pct_near_zero, pct_near_one.

`$models` An empty list.

See Also

[ps_match\(\)](#), [ps_weight\(\)](#), [sa_rosenbaum\(\)](#), [sa_evalue\(\)](#), [sa_trim_sweep\(\)](#)

Examples

```
dta <- sample_ps_data(n = 400, seed = 7)
res <- sa_overlap(dta, score_col = "prob_t", treatment_col = "tavr")
res$meta$overlap_region
res$tables$positivity_flags

# From a ps_match object
m <- ps_match(dta)
res <- sa_overlap(m)
res$tables$outside_overlap
```

sa_rosenbaum

Rosenbaum Sensitivity Bounds for Matched Analyses

Description

Assesses how sensitive a matched-pair analysis is to unmeasured confounding using the Rosenbaum bounds framework. For each value of the sensitivity parameter Gamma – the maximum odds ratio for differential treatment assignment within a matched pair due to hidden bias – the function reports the upper and lower bounds on the p-value of the Wilcoxon signed-rank test.

Usage

```
sa_rosenbaum(
  x,
  outcome_col,
  gamma_max = 3,
  gamma_inc = 0.25,
  alpha = 0.05,
  treatment_col = NULL,
  pair_id_col = "pair_id",
  treated_level = NULL,
  alternative = c("auto", "greater", "less")
)
```

Arguments

<code>x</code>	A <code>ps_match</code> object (returned by <code>ps_match()</code>) or a plain data frame whose rows are matched patients.
<code>outcome_col</code>	Name of the numeric outcome column. Required.
<code>gamma_max</code>	Maximum Gamma to test. Default 3.
<code>gamma_inc</code>	Increment between Gamma values. Default 0.25.
<code>alpha</code>	Significance threshold. Default 0.05.
<code>treatment_col</code>	Name of the binary treatment column (0/1). Taken from <code>x\$meta\$treatment_col</code> when <code>x</code> is a <code>ps_match</code> object. Required when <code>x</code> is a plain data frame.
<code>pair_id_col</code>	Name of the pair-ID column created by <code>ps_match()</code> . Taken from <code>x\$meta\$pair_id_col</code> when recorded. Default "pair_id".
<code>treated_level</code>	The value of <code>treatment_col</code> that is treated. Read from the metadata of a <code>ps_data</code> object when it records one. NULL (default) requires a 0/1 or logical column and takes 1 or TRUE.
<code>alternative</code>	Direction of the test: "auto" (default; chosen from the sign of the median difference, see Details), "greater" (treated outcome above control) or "less".

Details

A Gamma of 1 corresponds to no hidden bias (equivalent to perfect randomisation within pairs). The *sensitivity value* is the largest Gamma at which the upper p-value bound remains below alpha; a larger sensitivity value indicates a more robust result.

The statistic is the Wilcoxon signed-rank sum T^+ of the ranks of $|d_i|$ over pairs with $d_i > 0$, where d_i is the treated minus the control outcome and zero differences are dropped. Tied $|d_i|$ take their average rank. Under hidden bias Gamma each pair is positive with probability at most $p^+ = \Gamma/(1 + \Gamma)$, so the worst-case null distribution has mean $p^+ \sum r_i$ and variance $p^+(1 - p^+) \sum r_i^2$, and the best case uses $p^- = 1/(1 + \Gamma)$. Using $\sum r_i^2$ rather than $n(n + 1)(2n + 1)/6$ corrects the variance for ties; without ties the two agree. The p-values use the normal approximation with a continuity correction of 0.5, so at Gamma = 1 they equal the one-sided `stats::wilcox.test()` p-value with `exact = FALSE` and `correct = TRUE`.

Rosenbaum's method assumes the direction of the test is fixed before the outcome is seen. Set `alternative` to "greater" (treated above control) or "less" to state it. The default "auto" keeps the earlier behaviour and picks the direction from the data: it tests the lower tail, with a warning, when the median nonzero difference is negative. A direction chosen from the data makes the bounds slightly optimistic, so prefer an explicit alternative in a pre-specified analysis.

Column names and the treated level are read from a `ps_match()` object's metadata. A supplied `treatment_col`, `pair_id_col` or `treated_level` must agree with that metadata; a conflict aborts.

Value

An object of class `c("sa_rosenbaum", "ps_data")`:

`$data` The input data frame (`x$data` for a `ps_match` object).

`$meta` Column names, `treated_level`, the settings (`gamma`, the grid tested, `gamma_max`, `gamma_inc`, `alpha`, `alternative`, the direction tested, and `direction_from_data`), and the results: `sensitivity_value`, the largest Gamma for which the upper p-value is below alpha (NA if the analysis is not significant at Gamma = 1), `n_pairs`, the complete matched pairs used, `n_dropped`, `n_nonzero` and the statistic `t_plus`.

`$tables$bounds` Data frame with columns `gamma`, `p_upper`, `p_lower`, `reject_upper` (logical – is the upper bound < alpha?).

`$tables$pairs` One row per complete pair: `pair_id`, treated and control outcomes, and difference.

`$models` An empty list.

References

Rosenbaum, P. R. (2002). *Observational Studies* (2nd ed.). Springer.

See Also

[ps_match\(\)](#), [sa_evalue\(\)](#), [sa_overlap\(\)](#), [sa_trim_sweep\(\)](#)

Examples

```
dta <- sample_ps_data(n = 300, seed = 1)
m <- ps_match(dta)
res <- sa_rosenbaum(m, outcome_col = "ef", gamma_max = 2, alternative = "greater")
res
res$tables$bounds
res$meta$sensitivity_value
```

sa_trim_sweep

IPTW Weight-Trimming Sensitivity Sweep

Description

Re-computes IPTW weights across a range of winsorisation thresholds and reports how the weight distribution and effective sample size (ESS) change as trimming increases. This helps identify a trimming threshold that adequately controls extreme weights without an unacceptable loss of effective sample size.

Usage

```
sa_trim_sweep(
  x,
  trim_seq = seq(0, 0.1, by = 0.01),
  estimand = c("ATE", "ATT", "ATC"),
  stabilise = TRUE,
  score_col = "prob_t",
  treatment_col = "tavr",
  treated_level = NULL
)
```

Arguments

<code>x</code>	A <code>ps_weight</code> object (returned by <code>ps_weight()</code>) or a <code>ps_match</code> object or a plain data frame that contains a propensity score column and a binary treatment column.
<code>trim_seq</code>	Numeric vector of trimming values to evaluate. Each value is the lower tail quantile; weights outside the $(\text{trim}, 1 - \text{trim})$ range are winsorised. 0 means no trimming. Default <code>seq(0, 0.10, by = 0.01)</code> .
<code>estimand</code>	One of "ATE", "ATT", or "ATC" – the causal estimand used to compute IPTW weights. Taken from <code>x\$meta\$estimand</code> when <code>x</code> is a <code>ps_weight</code> object; defaults to "ATE" otherwise.
<code>stabilise</code>	Logical. Whether to stabilise weights. Taken from <code>x\$meta\$stabilised</code> when available; defaults to TRUE. As in <code>ps_weight()</code> , stabilisation applies to the ATE only.
<code>score_col</code>	Name of the propensity score column. Taken from <code>x\$meta\$score_col</code> when <code>x</code> is a <code>ps_data</code> object. Default "prob_t". The score of a <code>ps_match()</code> object built with <code>scale = "logit"</code> is a logit, and is back-transformed with <code>stats::plogis()</code> before use.
<code>treatment_col</code>	Name of the binary treatment column. Taken from <code>x\$meta\$treatment_col</code> when <code>x</code> is a <code>ps_data</code> object. Default "tavr".
<code>treated_level</code>	The value of <code>treatment_col</code> that is treated. Read from the metadata of a <code>ps_data</code> object when it records one. NULL (default) requires a 0/1 or logical column and takes 1 or TRUE.

Details

For a `ps_data` input, a supplied `estimand`, `stabilise`, `score_col`, `treatment_col` or `treated_level` must agree with the object's metadata; a conflict aborts. Arguments not supplied are read from the metadata, then take their defaults.

Value

An object of class `c("sa_trim_sweep", "ps_data")`:

`$data` The input data frame (`x$data` for a `ps_data` object), unchanged.

`$meta` `score_col`, `treatment_col`, `treated_level`, `scale`, `estimand`, `stabilised` (whether stabilisation was applied, so FALSE for the ATT and ATC), `trim_seq` (sorted), `n_total` and `n_used`, the patients with a non-missing treatment and score.

`$tables$sweep` Data frame with one row per trim value and columns `trim` (the winsorisation threshold), `ess_control` and `ess_treated` (effective sample sizes), `max_weight` (maximum weight after trimming), `sd_weight` (standard deviation of the weights after trimming) and `pct_trimmed` (percentage of patients whose weight was winsorised).

`$models` An empty list.

See Also

`ps_weight()`, `sa_rosenbaum()`, `sa_evaluate()`, `sa_overlap()`

Examples

```
dta    <- sample_ps_data(n = 400, seed = 3)
w_obj  <- ps_weight(dta, estimand = "ATE")
sweep  <- sa_trim_sweep(w_obj)
head(sweep$tables$sweep)

# Directly from a data frame
sweep2 <- sa_trim_sweep(dta, score_col = "prob_t", treatment_col = "tavr",
                        estimand = "ATE")
```

sample_ps_data	<i>Generate sample data for propensity score examples</i>
----------------	---

Description

Produces a synthetic data frame that resembles a typical SAS export from a cardiac-surgery propensity score analysis. Propensity scores are generated from a logistic model driven by the supplied covariates, so the resulting prob_t column has realistic overlap and the SMD values are non-trivial.

Usage

```
sample_ps_data(n = 500L, seed = 42L, separation = 1)
```

Arguments

n	Number of patients per treatment arm. Total rows = 2 * n. Must be a positive integer. Default 500.
seed	Random seed for reproducibility. Default 42L. The fixed default gives the same demo dataset on every call. NULL draws from the caller's random-number stream, as <code>ps_match()</code> does: the data differ from call to call, and <code>set.seed()</code> before the call reproduces them. A negative seed gives the same data as its absolute value.
separation	Numeric scalar controlling how well the covariates separate the two groups. Higher values give less overlap. Default 1.0.

Value

A data frame with 2 * n rows and the following columns:

id Integer patient identifier.
 tavr Binary treatment indicator: 0 = SAVR, 1 = TF-TAVR.
 age Continuous: age in years.
 female Binary: 1 = female.
 ef Continuous: ejection fraction (%).
 diabetes Binary: 1 = diabetic.

hypertension Binary: 1 = hypertensive.
 probab_t Estimated propensity score (numeric, 0 to 1).
 match Match indicator, initialised to 0. Populated by `ps_match()`.

See Also

`ps_match()`, `ps_weight()`

Examples

```
dta <- sample_ps_data(n = 200, seed = 42)
head(dta)
table(dta$tavr)
hist(dta$probab_t, breaks = 20, main = "Propensity score distribution")
```

sample_ps_data_count *Generate sample data with a count outcome for bs_count() examples*

Description

Produces a synthetic data frame with a Poisson-distributed count outcome (`rbc_tot`, total intra-operative RBC units transfused), along with baseline covariates and an optional multiply-imputed stacked format. Mirrors the dataset used in `tp.pm.count.balancing_score.sas`.

Usage

```
sample_ps_data_count(n = 500L, seed = 42L, n_imputations = 1L)
```

Arguments

<code>n</code>	Number of patients. Default 500L.
<code>seed</code>	Random seed. Default 42L. The fixed default gives the same demo dataset on every call. NULL draws from the caller's random-number stream, as <code>ps_match()</code> does: the data differ from call to call, and <code>set.seed()</code> before the call reproduces them. A negative seed gives the same data as its absolute value.
<code>n_imputations</code>	If > 1 , returns <code>n_imputations</code> datasets stacked in the long format, with an <code>_IMPUTATION_</code> index column. Each is an independent draw of <code>n</code> patients, covariates and outcome alike, that reuses the <code>id</code> values 1 to <code>n</code> . They exercise the stacked format; they do not resemble real imputations, in which one patient's observed values and outcome agree across datasets. Default 1L (single complete dataset).

Value

A data frame with columns: `id`, `rbc_tot` (count), `age`, `female`, `diabetes`, `hypertension`, `hct_pr` (pre-op haematocrit), `cpb_time` (cardiopulmonary bypass time, minutes), and optionally `_IMPUTATION_` when `n_imputations > 1`.

See Also

[bs_count\(\)](#), [bs_continuous\(\)](#)

Examples

```
dta <- sample_ps_data_count(n = 200, seed = 7)
hist(dta$rbc_tot, main = "RBC units distribution")

# Stacked multiply-imputed version (3 imputations)
dta_mi <- sample_ps_data_count(n = 200, seed = 7, n_imputations = 3)
table(dta_mi[["_IMPUTATION_"]])
```

sample_ps_data_nominal

Generate sample data with a nominal treatment for ps_nominal() examples

Description

Produces a synthetic data frame with a four-level nominal treatment variable (rtyp: tricuspid repair types COS, PER, DEV, CE), mimicking the dataset used in `tp.lm.logistic.propensity.polytomous.sas`.

Usage

```
sample_ps_data_nominal(n = 200L, seed = 42L)
```

Arguments

n	Number of patients per treatment level. Total rows = 4 * n. Default 200L.
seed	Random seed. Default 42L. The fixed default gives the same demo dataset on every call. NULL draws from the caller's random-number stream, as ps_match() does: the data differ from call to call, and <code>set.seed()</code> before the call reproduces them. A negative seed gives the same data as its absolute value.

Value

A data frame with 4 * n rows and columns: id, rtyp (unordered factor), age, female, ef, diabetes, hypertension.

See Also

[ps_nominal\(\)](#)

Examples

```
dta <- sample_ps_data_nominal(n = 100, seed = 7)
table(dta$rtyp)
str(dta)
```

```
sample_ps_data_ordinal
```

Generate sample data with an ordinal treatment for ps_ordinal() examples

Description

Produces a synthetic data frame with a three-level ordered treatment variable (nyha_grp: NYHA functional classes I, II, III), mimicking a typical cardiac-surgery dataset used with `tp.lm.logistic_propensity_ordinal`.

Usage

```
sample_ps_data_ordinal(n = 300L, seed = 42L)
```

Arguments

n	Number of patients per treatment level. Total rows = $3 * n$. Default 300L.
seed	Random seed. Default 42L. The fixed default gives the same demo dataset on every call. NULL draws from the caller's random-number stream, as <code>ps_match()</code> does: the data differ from call to call, and <code>set.seed()</code> before the call reproduces them. A negative seed gives the same data as its absolute value.

Value

A data frame with $3 * n$ rows and columns: id, nyha_grp (ordered factor: I < II < III), age, female, ef, diabetes, hypertension.

See Also

[ps_ordinal\(\)](#)

Examples

```
dta <- sample_ps_data_ordinal(n = 100, seed = 7)
table(dta$nyha_grp)
str(dta)
```

summary.ps_data	<i>Summarise a ps_data object</i>
-----------------	-----------------------------------

Description

Returns (and prints) a list of the available diagnostic tables.

Usage

```
## S3 method for class 'ps_data'
summary(object, ...)
```

Arguments

object	A ps_data object.
...	Ignored.

Value

object\$tables, invisibly.

validate_logistic	<i>Validate a saved binary logistic model</i>
-------------------	---

Description

Scores a declared validation cohort with every fitted model retained in a binary `fit_logistic()` bundle. It does not refit or modify the source bundle.

Usage

```
validate_logistic(
  model,
  data,
  outcome_col = NULL,
  prediction_col = "predicted",
  groups = 10L
)
```

Arguments

model	A binary <code>lm_fit</code> bundle with bundle version 1.
data	Validation data containing the stored predictors and outcome.
outcome_col	Outcome column. Defaults to the bundle's outcome column.
prediction_col	Output column for averaged predicted probabilities.
groups	Number of equal-frequency calibration groups.

Value

An object of class `c("lm_validation", "ps_data")` with:

`$data` data with `prediction_col` appended: the predicted probability of the bundle's `event_level`, averaged across the stored models on the probability scale. A row a model cannot score, such as one missing a predictor, gets NA.

`$meta` Named list: `bundle_version`, `model_family`, `formula`, `outcome_col`, `outcome_levels`, `event_level`, `predictors` and `package_versions`, carried from model; `prediction_col`, `groups`, `n_input`, `n_analyzed` (rows with an outcome and a prediction) and `n_excluded`.

`$tables` Named list: `calibration`, one row per equal-frequency group of the predicted probability (group, n, observed events, expected events); and `performance`, one row (n, observed, expected, `oe_ratio`, `auc` (the c-statistic) and `brier`).

`$models` The stored models from model, unchanged.

Examples

```
dta <- sample_ps_data(n = 150, seed = 42)
odd <- seq_len(nrow(dta)) %% 2 == 1
# Fit on one half, then score the other half without refitting.
fit <- fit_logistic(
  tavr ~ age + female + ef,
  data      = dta[odd, ],
  family    = "binary",
  outcome_levels = c(0, 1),
  event_level  = 1
)
val <- validate_logistic(fit, dta[!odd, ])
val$tables$performance
val$tables$calibration
```

Index

bs_continuous, 3
bs_continuous(), 8, 10, 49
bs_count, 5
bs_count(), 5, 49

fit_linear, 9
fit_linear(), 4, 5
fit_logistic, 10
fit_logistic(), 7, 10, 18, 19, 26, 27, 29, 51

is_ps_data, 13

MASS::glm.nb(), 5–7
MASS::polr(), 11, 27, 29

nnet::multinom(), 5, 7, 10, 11, 13, 25, 26

print.ps_data, 14
ps_forest, 15
ps_forest(), 24, 37
ps_logistic, 16
ps_logistic(), 5, 8, 12, 15, 16, 20, 24, 26, 28, 29, 33, 37
ps_match, 20
ps_match(), 15, 18, 19, 26, 29–31, 34, 36, 39, 42–50
ps_mw_var, 23
ps_mw_var(), 18
ps_nominal, 25
ps_nominal(), 19, 29, 49
ps_ordinal, 27
ps_ordinal(), 19, 26, 50
ps_rmst, 30
ps_stddiff, 32
ps_stddiff(), 18, 24, 34–36
ps_stddiff_perm, 34
ps_support, 36
ps_weight, 37
ps_weight(), 15, 18, 19, 22, 24, 30–34, 43, 46, 48

randomForestSRC::rfsrc(), 15, 16

sa_evalue, 39
sa_evalue(), 43, 45, 46
sa_overlap, 41
sa_overlap(), 15, 41, 45, 46
sa_rosenbaum, 43
sa_rosenbaum(), 41, 43, 46
sa_trim_sweep, 45
sa_trim_sweep(), 41, 43, 45
sample_ps_data, 47
sample_ps_data(), 5, 19, 22, 39
sample_ps_data_count, 48
sample_ps_data_nominal, 49
sample_ps_data_ordinal, 50
stats::glm(), 5–7, 11, 18, 29
stats::lm(), 4
stats::plogis(), 30, 36, 42, 46
stats::wilcox.test(), 44
summary.ps_data, 51

validate_logistic, 51
validate_logistic(), 12