

Package ‘cobalt’

August 25, 2026

Title Covariate Balance Tables and Plots

Version 5.0.0

Description Generate balance tables and plots for covariates of groups preprocessed through matching, weighting or subclassification, for example, using propensity scores. Includes integration with 'MatchIt', 'WeightIt', 'MatchThem', 'twang', 'Matching', 'optmatch', 'CBPS', 'ebal', 'cem', 'sbw', and 'designmatch' for assessing balance on the output of their preprocessing functions. Users can also specify data for balance assessment not generated through the above packages. Also included are methods for assessing balance in clustered or multiply imputed data sets or data sets with multi-category, continuous, or longitudinal treatments.

Depends R (>= 4.1.0)

Imports stats, utils, grid, grDevices, ggplot2 (>= 4.0.0), gridExtra (>= 2.3), arg (>= 0.1.1), rlang (>= 1.2.0), cli (>= 3.6.5)

Suggests MatchIt (>= 4.0.0), WeightIt (>= 1.0.0), twang (>= 1.6), twangContinuous, Matching, optmatch, ebal, CBPS (>= 0.17), designmatch, optweight, mice (>= 3.8.0), MatchThem (>= 0.9.3), cem (>= 1.1.30), sbw (>= 1.1.5), gbm (>= 2.1.7), brglm2 (>= 0.9), caret, gtable (>= 0.3.6), knitr, rmarkdown, testthat (>= 3.0.0)

License GPL (>= 2)

Language en-US

Encoding UTF-8

LazyData true

VignetteBuilder knitr

URL <https://ngreifer.github.io/cobalt/>,
<https://github.com/ngreifer/cobalt>

BugReports <https://github.com/ngreifer/cobalt/issues>

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Noah Greifer [aut, cre] (ORCID:
<https://orcid.org/0000-0003-3067-7154>)

Maintainer Noah Greifer <noah.greifer@gmail.com>

Repository CRAN

Date/Publication 2026-08-25 21:10:16 UTC

Contents

.cens	3
bal.compute	4
bal.plot	8
bal.tab	12
bal.tab.CBPS	19
bal.tab.cem.match	22
bal.tab.default	25
bal.tab.designmatch	30
bal.tab.ebalance	33
bal.tab.formula	36
bal.tab.Match	42
bal.tab.matchit	46
bal.tab.mimids	49
bal.tab.optmatch	52
bal.tab.ps	56
bal.tab.sbwcau	59
bal.tab.time.list	62
bal.tab.weightit	67
balance-statistics	70
balance-summary	72
class-bal.tab.cens	81
class-bal.tab.cluster	83
class-bal.tab.imp	84
class-bal.tab.msm	86
class-bal.tab.multi	88
class-bal.tab.subclass	90
display-options	91
extract.bal.tab	95
f.build	97
get.w	98
lalonge	101
love.plot	102
print.bal.tab	108
set.cobalt.options	111
splitfactor	113
var.names	115

Index

118

.cens

Mark a Censoring Indicator

Description

`.cens()` marks a variable as a censoring indicator rather than a treatment, so that `bal.tab()` assesses the balance of the units still under observation against the full at-risk sample. It is most often used on the left side of a formula, as in `.cens(C) ~ x1 + x2`, but it can also be called directly and supplied to `bal.tab()`'s `treat` argument.

Usage

```
.cens(x)
```

Arguments

<code>x</code>	a censoring indicator: a numeric variable taking only the values 0 (still under observation) and 1 (censored), a logical variable, or a factor with levels 0/1 or FALSE/TRUE. Missing values are allowed and preserved.
----------------	---

Details

Censoring is considered its own treatment type in **cobalt**, distinct from binary, multi-category, and continuous treatments. What matters is whether the units still under observation resemble the full sample once weighted. See `class-bal.tab.cens` for the output this produces and the arguments that control it.

This function is deliberately identical in name and contract to `WeightIt::.cens()`, so that the same code works whichever package is attached; **cobalt** defines its own only to avoid depending on **WeightIt**. `bal.tab()` recognizes an indicator tagged by either.

Value

`x` coerced to a 0/1 numeric vector of class `treat` (see `treat`) with a `"treat.type"` attribute of `"censoring"`. Any value other than 0, 1, or NA throws an error.

Inside a formula the marker is stripped before the formula is processed, so `.cens()` is not actually evaluated there and the treatment name remains that of the indicator itself (e.g., `C` rather than `.cens(C)`).

Note on the coding convention

The survival convention is used: 1 means the unit is *censored* (drops out of observation) and 0 means it remains under observation. This is the opposite of an "observed" or "event" indicator.

Missing values are permitted because a unit censored at an earlier time point has no later indicator. Units with a missing indicator are not at risk, so they are in neither sample.

See Also

- [class-bal.tab.cens](#) for the output of `bal.tab()` with a censoring indicator
- [bal.tab\(\)](#)

Examples

```
data("lalonge", package = "cobalt")

# A censoring indicator: 1 = lost to follow-up
set.seed(1234)
lalonge$C <- rbinom(nrow(lalonge), 1,
                  prob = plogis(-1.5 + 0.05 * lalonge$age))

# Inverse probability of censoring weights
W <- WeightIt::weightit(.cens(C) ~ age + educ + race,
                      data = lalonge, method = "glm")

# Balance of the weighted uncensored units against the
# full at-risk sample
bal.tab(W, un = TRUE)

# The same thing without WeightIt, using the weights directly
bal.tab(.cens(C) ~ age + educ + race, data = lalonge,
       weights = W$weights, un = TRUE)
```

bal.compute

Efficiently compute scalar balance statistics

Description

These are functions primarily designed for programmers who want to be able to quickly compute one of several scalar (single number) sample balance statistics, e.g., for use in selecting a tuning parameter when estimating balancing weights. `bal.compute()` computes a scalar balance statistics from the supplied inputs. `bal.init()` initializes the input so that when `bal.compute()` is used on the output along with a set of weights, the computation of the balance statistic is fast. `vignette("optimizing-balance")` provides an overview and more examples of how to use these functions. `available.stats()` returns the balance statistics available for the given treatment type.

Usage

```
bal.compute(x, ...)

## S3 method for class 'bal.init'
bal.compute(x, weights = NULL, ...)

## Default S3 method:
bal.compute(x, treat = NULL, stat, s.weights = NULL, weights = NULL, ...)
```

```
bal.init(x, treat = NULL, stat, s.weights = NULL, ...)
```

```
available.stats(treat.type = "binary")
```

Arguments

<code>x</code>	for <code>bal.compute()</code> , a <code>bal.init</code> object created by <code>bal.init()</code> or a matrix or data frame containing the covariates. For <code>bal.init()</code> , a matrix or data frame containing the covariates.
<code>...</code>	other arguments used to specify options for the balance statistic. See Details for which arguments are allowed with each balance statistic. Ignored for the <code>bal.init</code> method of <code>bal.compute()</code> .
<code>weights</code>	a vector of balancing weights to compute the weighted statistics.
<code>treat</code>	a vector containing the treatment variable. Can be <code>NULL</code> to compute target balance statistics.
<code>stat</code>	string; the name of the statistic to compute. See Details.
<code>s.weights</code>	optional; a vector of sampling weights.
<code>treat.type</code>	string; the treatment type, either "binary", "multinomial", "continuous", or "target". Abbreviations allowed.

Details

The following list contains the allowable balance statistics that can be supplied to `bal.init()` or the default method of `bal.compute()`, the additional arguments that can be used with each one, and the treatment types allowed with each one. For all balance statistics, lower values indicate better balance. Target balance refers to the similarity between a sample and the same sample but weighted.

`smd.mean`, `smd.max`, `smd.rms` The mean, maximum, or root-mean-squared absolute standardized mean difference, computed using `col_w_smd()`. The other allowable arguments include `estimand` ("ATE", "ATT", or "ATC") to select the estimand (default is "ATE"), `focal` to identify the focal treatment group when the ATT is the estimand and the treatment has more than two categories, and `pairwise` to select whether mean differences should be computed between each pair of treatment groups or between each treatment group and the target group identified by `estimand` (default `TRUE`). Can be used with binary and multi-category treatments and for target balance.

`ks.mean`, `ks.max`, `ks.rms` The mean, maximum, or root-mean-squared Kolmogorov-Smirnov statistic, computed using `col_w_ks()`. The other allowable arguments include `estimand` ("ATE", "ATT", or "ATC") to select the estimand (default is "ATE"), `focal` to identify the focal treatment group when the ATT is the estimand and the treatment has more than two categories, and `pairwise` to select whether statistics should be computed between each pair of treatment groups or between each treatment group and the target group identified by `estimand` (default `TRUE`). Can be used with binary and multi-category treatments and for target balance.

`ovl.mean`, `ovl.max`, `ovl.rms` The mean, maximum, or root-mean-squared overlapping coefficient complement, computed using `col_w_ovl()`. The other allowable arguments include `estimand` ("ATE", "ATT", or "ATC") to select the estimand (default is "ATE"), `integrate` to select whether integration is done using `integrate()` (`TRUE`, the default) or a midpoint Riemann sum (`FALSE`), `focal` to identify the focal treatment group when the ATT is the estimand and

the treatment has more than two categories, pairwise to select whether statistics should be computed between each pair of treatment groups or between each treatment group and the target group identified by estimand (default TRUE). Can be used with binary and multi-category treatments and for target balance.

mahalanobis The Mahalanobis distance between the treatment group means. This is similar to **smd.rms** but the covariates are standardized to remove correlations between them and de-emphasize redundant covariates. The other allowable arguments include **estimand** ("ATE", "ATT", or "ATC") to select the estimand (default is "ATE") and **focal** to identify the focal treatment group when the ATT is the estimand. Can only be used with binary treatments and for target balance.

energy.dist The total energy distance between each treatment group and the target sample, which is a scalar measure of the similarity between two multivariate distributions. The other allowable arguments include **estimand** ("ATE", "ATT", "ATC", or NULL) to select the estimand (default is NULL), **focal** to identify the focal treatment group when the ATT is the estimand and the treatment has more than two categories, and **improved** to select whether the "improved" energy distance should be used when **estimand** = "ATE", which emphasizes difference between treatment groups in addition to difference between each treatment group and the target sample (default TRUE). When **estimand** = NULL, only the energy distance between the treatment groups will be computed (i.e., as opposed to the energy distance between each treatment groups and the target sample). Can be used with binary and multi-category treatments and for target balance.

kernel.dist The kernel distance between the treatment groups, which is a scalar measure of the similarity between two multivariate distributions. Can only be used with binary treatments.

l1.med The median L1 statistic computed across a random selection of possible coarsening of the data. The other allowable arguments include **estimand** ("ATE", "ATT", or "ATC") to select the estimand (default is "ATE"), **focal** to identify the focal treatment group when the ATT is the estimand and the treatment has more than two categories, **l1.min.bin** (default 2) and **l1.max.bin** default (12) to select the minimum and maximum number of bins with which to bin continuous variables and **l1.n** (default 101) to select the number of binnings used to select the binning at the median. **covs** should be supplied without splitting factors into dummies to ensure the binning works correctly; for simplicity, the **.covs** argument can be supplied, which will override **covs** but isn't used by other statistics. Can be used with binary and multi-category treatments.

r2, r2.2, r2.3 The post-weighting R^2 of a model for the treatment. The other allowable arguments include **poly** to add polynomial terms of the supplied order to the model and **int** (default FALSE) to add two-way interaction between covariates into the model. Using **r2.2** is a shortcut to requesting squares, and using **r2.3** is a shortcut to requesting cubes. Can be used with binary and continuous treatments. For binary treatments, the McKelvey and Zavoina R^2 from a logistic regression is used; for continuous treatments, the R^2 from a linear regression is used.

p.mean, p.max, p.rms The mean, maximum, or root-mean-squared absolute Pearson correlation between the treatment and covariates, computed using **col_w_corr()**. Can only be used with continuous treatments.

s.mean, s.max, s.rms The mean, maximum, or root-mean-squared absolute Spearman correlation between the treatment and covariates, computed using **col_w_corr()**. Can only be used with continuous treatments.

distance.cov, distance.cor The distance covariance or distance correlation, respectively, between the scaled covariates and treatment, which is a scalar measure of the independence of

two possibly multivariate distributions. The distance correlation is scale-free and ranges from 0 (completely independent) to 1 (perfectly associated). Can only be used with continuous treatments.

Although statistics can be computed directly using `bal.compute()` alone, the intended workflow is to use `bal.init()` to initialize a `bal.init` object, which can then be passed to `bal.compute()` many times with different sets of weights, thereby minimizing the processing that `bal.init()` does because it is only done once. In contrast, using `bal.compute()` on covariates directly (i.e., using the default method) calls `bal.init()` internally each time, which can slow down evaluation. When speed isn't of interest or to calculate a balance statistic outside the context of balance optimization, the default method of `bal.compute()` can be a quick shortcut to avoid having to create a `bal.init` object first.

Value

For `bal.compute()`, a single numeric value. For `bal.init()`, a `bal.init` object containing the components created in the initialization and the function used to compute the balance statistic. For `available.stats()`, a character vector of available statistics.

See Also

[balance-summary](#), [bal.tab\(\)](#)

See `vignette("optimizing-balance")` for references and definitions of some of the above quantities.

Examples

```
# Select the optimal number of subclasses for
# subclassification:
data("lalonge")
covs <- c("age", "educ", "race", "married",
          "nodegree", "re74", "re75")

# Estimate propensity score
p <- glm(reformulate(covs, "treat"),
        data = lalonge,
        family = "binomial")$fitted.values

# Function to compute subclassification weights
subclass_ATE <- function(treat, p, nsub) {
  m <- MatchIt::matchit(treat ~ 1,
                        data = lalonge,
                        distance = p,
                        method = "subclass",
                        estimand = "ATE",
                        subclass = nsub)

  m$weights
}

# Initialize balance statistic; largest KS statistic
init <- bal.init(lalonge[covs],
```

```

      treat = lalonde$treat,
      stat = "ks.max",
      estimand = "ATE")

# Statistic prior to subclassification:
bal.compute(init)

# Testing 4 to 50 subclasses
nsubs <- 4:50
stats <- vapply(nsubs, function(n) {
  w <- subclass_ATE(lalonde$treat, p, n)
  bal.compute(init, w)
}, numeric(1L))

plot(stats ~ nsubs)

# 6 subclass gives lowest ks.max value (.238)
nsubs[which.min(stats)]
stats[which.min(stats)]

# See which statistics are available
available.stats("binary")
available.stats("multinomial")
available.stats("continuous")
available.stats("target")

```

bal.plot

Visualize Distributional Balance

Description

Generates density plots, bar graphs, or scatterplots displaying distributional balance between treatment and covariates using **ggplot2**.

Usage

```

bal.plot(
  x,
  var.name,
  ...,
  which,
  which.sub = NULL,
  cluster = NULL,
  which.cluster = NULL,
  imp = NULL,
  which.imp = NULL,
  which.treat = NULL,
  which.time = NULL,

```

```

mirror = FALSE,
type = "density",
colors = NULL,
grid = FALSE,
sample.names,
position = "right",
facet.formula = NULL,
disp.means = getOption("cobalt_disp.means", FALSE),
alpha.weight = TRUE
)

```

Arguments

<code>x</code>	the object for which balance is to be assessed; can be any object for which there is support in <code>bal.tab()</code> .
<code>var.name</code>	character; the name of the variable whose values are to be plotted. To view distributions of the distance measure (e.g., propensity score), if any, use "distance" as the argument unless the distance variable has been named. If there are duplicate variable names across inputs, <code>bal.plot()</code> will first look in the covariate data.frame from <code>x</code> , followed by <code>add1</code> , and then <code>distance</code> , if any. If not specified, will use the first covariate available with a warning.
<code>...</code>	other arguments to define the variable, treatment, and weights. Some inputs are required depending on the method. See Additional Arguments. Can also be used to supply the <code>bw</code> , <code>adjust</code> , <code>kernel</code> , and <code>n</code> arguments for <code>ggplot2::geom_density()</code> and the <code>bins</code> argument for <code>ggplot2::geom_histogram()</code> .
<code>which</code>	whether to display distributional balance for the adjusted ("adjusted") or unadjusted sample ("unadjusted") or both at the same time ("both"). When multiple weights are present, the names of the weights can be supplied, too. The default is to display balance for the adjusted sample only unless no weights, subclasses, or matching strata are specified. Multiple values and abbreviations allowed.
<code>which.sub</code>	numeric; if subclassification is used, a vector corresponding to the subclass(es) for which the distributions are to be displayed. If <code>.all</code> (the default), distributions from all subclasses are displayed in a grid.
<code>cluster</code>	optional; a vector of cluster membership, or the name of a variable in an available data set passed to <code>bal.plot()</code> that contains cluster membership.
<code>which.cluster</code>	if clusters are used, which cluster(s) to display. Can be cluster names or numerical indices for which to display balance. Indices correspond to the alphabetical order of cluster names. If <code>.all</code> (the default), all clusters are displayed. If <code>.none</code> , cluster information is ignored and the marginal distribution of the covariates is displayed.
<code>imp</code>	optional; a vector of imputation indices, or the name of a variable in an available data set passed to <code>bal.plot()</code> that contains imputation indices.
<code>which.imp</code>	if imputations are used, which imputations(s) to display. Must be numerical indices for which to display balance. If <code>.all</code> (the default), all imputations are displayed. If <code>.none</code> , data from all imputations are combined into one distribution.

which.treat	which treatment groups to display. If NULL (the default) or NA, all treatment groups are displayed.
which.time	for longitudinal treatments, which time points to display. Can be treatment names or time period indices. If NULL (the default) or NA, all time points are displayed.
mirror	logical; if the treatment is binary, the covariate is continuous, and densities or histograms are requested, whether to display mirrored densities/histograms or overlapping densities/histograms. Ignored otherwise.
type	character; for binary and multi-category treatments with a continuous covariate, whether to display densities ("density"), histograms ("histogram"), or empirical cumulative density function plots ("ecdf"). The default is to display densities. Abbreviations are allowed.
colors	a vector of colors for the plotted densities/histograms. See 'Color Specification' at graphics::par() . Defaults to the default ggplot2 colors.
grid	logical; whether gridlines should be shown on the plot. Default is TRUE.
sample.names	character; new names to be given to the samples (i.e., in place of "Unadjusted Sample" and "Adjusted Sample"). For example, when matching it used, it may be useful to enter <code>c("Unmatched", "Matched")</code> .
position	the position of the legend. This can be any value that would be appropriate as an argument to <code>legend.position</code> in ggplot2::theme() .
facet.formula	a formula designating which facets should be on the rows and columns. This should be of the "historical" formula interface to ggplot2::facet_grid() . If of the form <code>a ~ b</code> , <code>a</code> will be faceted on the rows and <code>b</code> on the columns. To only facet on the rows, provide a one-sided formula with an empty left-hand side. To only facet on the columns, the formula should be of the form <code>a ~ .</code> (i.e., with only <code>.</code> on the right-hand side). The allowable facets depend on which arguments have been supplied to <code>bal.plot()</code> ; possible values include <code>which</code> , <code>cluster</code> , <code>imp</code> , and (for longitudinal treatments) <code>time</code> . If NULL, <code>bal.plot()</code> will decide what looks best; this argument exists in case you disagree with its choice.
disp.means	logical; for a categorical treatment with a continuous covariate, whether a line should be drawn for each treatment level denoting the (weighted) mean of the covariate. Ignored if <code>type</code> is not "density" or "histogram". Default is FALSE.
alpha.weight	logical; if both the treatment and the covariate are continuous, whether points should be shaded according to their weight. Fainter points are those that have smaller weights. Default is TRUE.

Details

`bal.plot()` uses [ggplot2::ggplot\(\)](#) from the **ggplot2** package, and (invisibly) returns a "ggplot" object. For categorical treatments with continuous covariates or continuous treatments with categorical covariates, density plots are created using [ggplot2::geom_density\(\)](#), histograms are created using [ggplot2::geom_histogram\(\)](#), and empirical CDF plots are created using [ggplot2::geom_step\(\)](#); for categorical treatments with categorical covariates, bar graphs are created using [ggplot2::geom_bar\(\)](#); for continuous treatments with continuous covariates, scatterplots are created using [ggplot2::geom_point\(\)](#).

For continuous treatments with continuous covariates, four additional lines are presented for aid in balance assessment. The red line is the linear fit line. The blue line is a smoothing curve generated

with `ggplot2`'s `ggplot2::geom_smooth()` with `method = "auto"`. The horizontal black line is a horizontal reference line intercepting the (unweighted) treatment mean. The vertical black line is a reference line intercepting the (unweighted) treatment mean. Balance is indicated by the flatness of both fit lines and whether they pass through the intersection of the two black reference lines.

When multiple plots are to be displayed (i.e., when requesting subclass balance, cluster balance, or imputation balance, or when multiple sets of weights are provided or `which = "both"`, or when treatment is longitudinal), the plots will be displayed in a grid using `ggplot2`'s `ggplot2::facet_grid()`. Subclassification cannot be used with clusters or multiply imputed data.

To change the plot and axis titles, use `ggplot2::labs()`. Because the output is a `ggplot` object, other elements can be changed using `ggplot2` functions; see [here](#) for an example.

Value

A "ggplot" object, returned invisibly.

Additional Arguments

`bal.plot()` works like `bal.tab()` in that it can take a variety of types of inputs and yield the same output for each. Depending on what kind of input is given, different additional parameters are required in For details on what is required and allowed for each additional input and their defaults, see the help file for the `bal.tab()` method associated with the input. The following are the required additional arguments based on each input type:

- For `matchit` objects: None
- For `weightit` objects: None
- For `ps`, `ps.cont`, `mnps`, and `iptw` objects: `stop.method` (see [defaults](#)).
- For `Match` objects: formula and data, or covs and treat.
- For `optmatch` objects: formula and data, or covs (treat is not required).
- For `CBPS` objects: None
- For `ebalance` objects: formula and data, or covs and treat.
- For formulas: data
- For `data.frames`: treat
- For `designmatch` objects: formula and data, or covs and treat.
- For `sbw` objects: None
- For `mimids` and `wimids` objects: None, but an argument to `which.imp` should be specified.
- For other objects processed through `bal.tab()`'s default method, whichever arguments are required to identify treatment, variables, and a conditioning method (if any).

See Also

[bal.tab\(\)](#), [love.plot\(\)](#)

Examples

```
data("lalonge", package = "cobalt")

#Nearest Neighbor Matching
library(MatchIt)
m.out <- matchit(treat ~ age + educ + race + married +
                 nodegree + re74 + re75,
                 data = lalonge)

bal.plot(m.out, "age", which = "both")
bal.plot(m.out, "re74", which = "both", type = "ecdf")
bal.plot(m.out, "race", which = "both")
bal.plot(m.out, "distance", which = "both", mirror = TRUE,
         type = "histogram", colors = c("white", "black"))

#Entropy balancing with a continuous treatment
library(WeightIt)
w.out <- weightit(re75 ~ age + I(age^2) + educ +
                 race + married + nodegree,
                 data = lalonge, method = "ebal")

bal.plot(w.out, "age", which = "both")
bal.plot(w.out, "married", which = "both")
```

bal.tab

Display Balance Statistics in a Table

Description

Generates balance statistics on covariates in relation to an observed treatment variable. It is a generic function that dispatches to the method corresponding to the class of the first argument.

Usage

```
bal.tab(x, ...)

## # Arguments common across all input types:
## bal.tab(x,
##       stats,
##       int = FALSE,
##       poly = 1,
##       distance = NULL,
##       add1 = NULL,
##       data = NULL,
##       continuous,
##       binary,
```

```
##      s.d.denom,
##      thresholds = NULL,
##      weights = NULL,
##      cluster = NULL,
##      imp = NULL,
##      pairwise = TRUE,
##      s.weights = NULL,
##      abs = FALSE,
##      subset = NULL,
##      quick = TRUE,
##      ...)
```

Arguments

x	an input object on which to assess balance. Can be the output of a call to a balancing function in another package or a formula or data frame. Input to this argument will determine which <code>bal.tab()</code> method is used. Each input type has its own documentation page, which is linked in the See Also section below. Some input types require or allow additional arguments to be specified. For inputs with no dedicated method, the default method will be dispatched. See Details below.
...	for some input types, other arguments that are required or allowed. Otherwise, further arguments to control display of output. See display options for details.
stats	character; which statistic(s) should be reported. See stats for allowable options. For binary and multi-category treatments, "mean.diffs" (i.e., mean differences) is the default. For continuous treatments, "correlations" (i.e., treatment-covariate Pearson correlations) is the default. Multiple options are allowed.
int	logical or numeric; whether or not to include 2-way interactions of covariates included in covs and in add1. If numeric, will be passed to poly as well.
poly	numeric; the highest polynomial of each continuous covariate to display. For example, if 2, squares of each continuous covariate will be displayed (in addition to the covariate itself); if 3, squares and cubes of each continuous covariate will be displayed, etc. If 1, the default, only the base covariate will be displayed. If int is numeric, poly will take on the value of int.
distance	an optional formula or data frame containing distance values (e.g., propensity scores) or a character vector containing their names. If a formula or variable names are specified, <code>bal.tab()</code> will look in the argument to data, if specified. For longitudinal treatments, can be a list of allowable arguments, one for each time point.
add1	an optional formula or data frame containing additional covariates for which to present balance or a character vector containing their names. If a formula or variable names are specified, <code>bal.tab()</code> will look in the arguments to the input object, covs, and data, if specified. For longitudinal treatments, can be a list of allowable arguments, one for each time point.
data	an optional data frame containing variables named in other arguments. For some input object types, this is required.

continuous	whether mean differences for continuous variables should be standardized ("std") or raw ("raw"). Default "std". Abbreviations allowed. This option can be set globally using <code>set.cobalt.options()</code> .
binary	whether mean differences for binary variables (i.e., difference in proportion) should be standardized ("std") or raw ("raw"). Default "raw". Abbreviations allowed. This option can be set globally using <code>set.cobalt.options()</code> .
s.d.denom	character; how the denominator for standardized mean differences should be calculated, if requested. See <code>col_w_smd()</code> for allowable options. If weights are supplied, each set of weights should have a corresponding entry to s.d.denom. Abbreviations allowed. If left blank and weights, subclasses, or matching strata are supplied, <code>bal.tab()</code> will figure out which one is best based on the estimand, if given (for ATT, "treated"; for ATC, "control"; otherwise "pooled") and other clues if not.
thresholds	a named vector of balance thresholds, where the name corresponds to the statistic (i.e., in stats) that the threshold applies to. For example, to request thresholds on mean differences and variance ratios, one can set <code>thresholds = c(m = .05, v = 2)</code> . Requesting a threshold automatically requests the display of that statistic. When specified, extra columns are inserted into the Balance table describing whether the requested balance statistics exceeded the threshold or not. Summary tables tallying the number of variables that exceeded and were within the threshold and displaying the variables with the greatest imbalance on that balance measure are added to the output.
weights	a vector, list, or data.frame containing weights for each unit, or a string containing the names of the weights variables in data, or an object with a <code>get.w()</code> method or a list thereof. The weights can be, e.g., inverse probability weights or matching weights resulting from a matching algorithm.
cluster	either a vector containing cluster membership for each unit or a string containing the name of the cluster membership variable in data or the input object. See <code>class-bal.tab.cluster</code> for details.
imp	either a vector containing imputation indices for each unit or a string containing the name of the imputation index variable in data or the input object. See <code>class-bal.tab.imp</code> for details. Not necessary if data is a mids object.
pairwise	whether balance should be computed for pairs of treatments or for each treatment against all groups combined. See <code>bal.tab.multi()</code> for details. This can also be used with a binary treatment to assess balance with respect to the full sample.
s.weights	Optional; either a vector containing sampling weights for each unit or a string containing the name of the sampling weight variable in data. These function like regular weights except that both the adjusted and unadjusted samples will be weighted according to these weights if weights are used.
abs	logical; whether displayed balance statistics should be in absolute value or not.
subset	a logical or numeric vector denoting whether each observation should be included or which observations should be included. If logical, it should have length equal to the number of units. NAs will be treated as FALSE. This can be used as an alternative to cluster to examine balance on subsets of the data.
quick	logical; if TRUE, will not compute any values that will not be displayed. Set to FALSE if computed values not displayed will be used later.

Details

`bal.tab()` performs various calculations on the the data objects given. This page details the arguments and calculations that are used across `bal.tab()` methods.

With Binary Point Treatments:

Balance statistics can be requested with the `stats` argument. The default balance statistic for mean differences for continuous variables is the standardized mean difference, which is the difference in the means divided by a measure of spread (i.e., a d-type effect size measure). This is the default because it puts the mean differences on the same scale for comparison with each other and with a given threshold. For binary variables, the default balance statistic is the raw difference in proportion. Although standardized differences in proportion can be computed, raw differences in proportion for binary variables are already on the same scale, and computing the standardized difference in proportion can obscure the true difference in proportion by dividing the difference in proportion by a number that is itself a function of the observed proportions.

Standardized mean differences are calculated using `col_w_smd()` as follows: the numerator is the mean of the treated group minus the mean of the control group, and the denominator is a measure of spread calculated in accordance with the argument to `s.d.denom` or the default of the specific method used. Common approaches in the literature include using the standard deviation of the treated group or using the "pooled" standard deviation (i.e., the square root of the mean of the group variances) in calculating standardized mean differences. The computed spread `bal.tab()` uses is always that of the full, unadjusted sample (i.e., before matching, weighting, or subclassification), as recommended by Stuart (2010).

Prior to computation, all variables are checked for variable type, which allows users to differentiate balance statistic calculations based on type using the arguments to `continuous` and `binary`. First, if a given covariate is numeric and has only 2 levels, it is converted into a binary (0,1) variable. If 0 is a value in the original variable, it retains its value and the other value is converted to 1; otherwise, the lower value is converted to 0 and the other to 1. Next, if the covariate is not numeric or logical (i.e., is a character or factor variable), it will be split into new binary variables, named with the original variable and the value, separated by an underscore. Otherwise, the covariate will be used as is and treated as a continuous variable.

When weighting or matching are used, an "effective sample size" is calculated for each group using the following formula: $(\sum w)^2 / \sum w^2$. The effective sample size is "approximately the number of observations from a simple random sample that yields an estimate with sampling variation equal to the sampling variation obtained with the weighted comparison observations" (Ridgeway et al., 2016). The calculated number tends to underestimate the true effective sample size of the weighted samples. The number depends on the variability of the weights, so sometimes trimming units with large weights can actually increase the effective sample size, even though units are being down-weighted. When matching is used, an additional "unweighted" sample size will be displayed indicating the total number of units contributing to the weighted sample.

When subclassification is used, the balance tables for each subclass stored in `$Subclass.Balance` use values calculated as described above. For the aggregate balance table stored in `$Balance.Across.Subclass`, the values of each statistic are computed as a weighted average of the statistic across subclasses, weighted by the proportion of units in each subclass. See `class-bal.tab.subclass` for more details.

With Continuous Point Treatments:

When continuous treatment variables are considered, the balance statistic calculated is the Pearson correlation between the covariate and treatment. The correlation after adjustment is computed

using `col_w_cov()` as the weighted covariance between the covariate and treatment divided by the product of the standard deviations of the unweighted covariate and treatment, in an analogous way to how the weighted standardized mean difference uses an unweighted measure of spread in its denominator, with the purpose of avoiding the analogous paradox (i.e., where the covariance decreases but is accompanied by a change in the standard deviations, thereby distorting the actual resulting balance computed using the weighted standard deviations). This can sometimes yield correlations greater than 1 in absolute value; these usually indicate degenerate cases anyway.

With Multi-Category Point Treatments:

For information on using `bal.tab()` with multi-category treatments, see `class-bal.tab.multi`. Essentially, `bal.tab()` compares pairs of treatment groups in a standard way.

With Longitudinal Treatments:

For information on using `bal.tab()` with longitudinal treatments, see `class-bal.tab.msm` and `vignette("longitudinal-treat")`. Essentially, `bal.tab()` summarizes balance at each time point and summarizes across time points.

With a Censoring Indicator:

A treatment supplied as `.cens()` is a censoring indicator rather than a treatment, and there is no second treatment group to compare against. `bal.tab()` instead compares the units still under observation, weighted, against the full at-risk sample. See `class-bal.tab.cens`.

With Clustered or Multiply Imputed Data:

For information on using `bal.tab()` with clustered data, see `class-bal.tab.cluster`. For information on using `bal.tab()` with multiply imputed data, see `class-bal.tab.imp`.

quick:

Calculations can take some time, especially when there are many variables, interactions, or clusters. When certain values are not printed, by default they are not computed. In particular, summary tables are not computed when their display has not been requested. This can speed up the overall production of the output when these values are not to be used later. However, when they are to be used later, such as when output is to be further examined with `print()` or is to be used in some other way after the original call to `bal.tab()`, it may be useful to compute them even if they are not to be printed initially. To do so, users can set `quick = FALSE`, which will cause `bal.tab()` to calculate all values and components it can. Note that `love.plot()` is fully functional even when `quick = TRUE` and values are requested that are otherwise not computed in `bal.tab()` with `quick = TRUE`.

Missing Data:

If there is missing data in the covariates (i.e., NAs in the covariates provided to `bal.tab()`), a few additional things happen. A warning will appear mentioning that missing values were present in the data set. The computed balance summaries will be for the variables ignoring the missing values. New variables will be created representing missingness indicators for each variable, named `var: <NA>` (with `var` replaced by the actual name of the variable). If `int = TRUE`, balance for the pairwise interactions between the missingness indicators will also be computed. These variables are treated like regular variables once created.

Value

An object of class "bal.tab". The use of continuous treatments, subclasses, clusters, and/or imputations will also cause the object to inherit other classes. The class "bal.tab" has its own `print()` method (`print.bal.tab()`), which formats the output nicely and in accordance with print-related options given in the call to `bal.tab()`, and which can be called with its own options.

For scenarios with binary point treatments and no subclasses, imputations, or clusters, the following are the elements of the `bal.tab` object:

Balance	A data frame containing balance information for each covariate. Balance contains the following columns, with additional columns present when other balance statistics are requested, and some columns omitted when not requested: • <code>Type</code>: Whether the covariate is binary, continuous, or a measure of distance (e.g., the propensity score). • <code>M.0.Un</code>: The mean of the control group prior to adjusting. • <code>SD.0.Un</code>: The standard deviation of the control group prior to adjusting. • <code>M.1.Un</code>: The mean of the treated group prior to adjusting. • <code>SD.1.Un</code>: The standard deviation of the treated group prior to adjusting. • <code>Diff.Un</code>: The (standardized) difference in means between the two groups prior to adjusting. See the binary and continuous arguments on the <code>bal.tab</code> method pages to determine whether standardized or raw mean differences are being reported. By default, the standardized mean difference is displayed for continuous variables and the raw mean difference (difference in proportion) is displayed for binary variables. • <code>M.0.Adj</code>: The mean of the control group after adjusting. • <code>SD.0.Adj</code>: The standard deviation of the control group after adjusting. • <code>M.1.Adj</code>: The mean of the treated group after adjusting. • <code>SD.1.Adj</code>: The standard deviation of the treated group after adjusting. • <code>Diff.Adj</code>: The (standardized) difference in means between the two groups after adjusting. See the binary and continuous arguments on the <code>bal.tab</code> method pages to determine whether standardized or raw mean differences are being reported. By default, the standardized mean difference is displayed for continuous variables and the raw mean difference (difference in proportion) is displayed for binary variables. • <code>M.Threshold</code>: Whether or not the calculated mean difference after adjusting exceeds or is within the threshold given by <code>thresholds</code>. If a threshold for mean differences is not specified, this column will be NA.
Balanced.Means	If a threshold on mean differences is specified, a table tallying the number of variables that exceed or are within the threshold.
Max.Imbalance.Means	If a threshold on mean differences is specified, a table displaying the variable with the greatest absolute mean difference.
Observations	A table displaying the sample sizes before and after adjusting. Often the effective sample size (ESS) will be displayed. See Details.
call	The original function call, if adjustment was performed by a function in another package.

If the treatment is continuous, instead of producing mean differences, `bal.tab()` will produce correlations between the covariates and the treatment. The default corresponding entries in the output will be "Corr.Un", "Corr.Adj", and "R.Threshold" (and accordingly for the balance tally and maximum imbalance tables).

If multiple weights are supplied, "Adj" in Balance will be replaced by the provided names of the sets of weights, and extra columns will be added for each set of weights. Additional columns and rows for other items in the output will be created as well.

For `bal.tab` output with subclassification, see [class-bal.tab subclass](#).

References

- Ridgeway, G., McCaffrey, D., Morral, A., Burgette, L., & Griffin, B. A. (2016). Toolkit for Weighting and Analysis of Nonequivalent Groups: A tutorial for the `twang` package. R vignette. RAND.
- Stuart, E. A. (2010). Matching Methods for Causal Inference: A Review and a Look Forward. *Statistical Science*, 25(1), 1-21. doi:[10.1214/09STS313](#)

See Also

For information on the use of `bal.tab()` with specific types of objects, use the following links:

- [bal.tab.matchit\(\)](#) for the method for objects returned by **MatchIt**.
- [bal.tab.weightit\(\)](#) for the method for `weightit` and `weightitMSM` objects returned by **WeightIt**.
- [bal.tab.ps\(\)](#) for the method for `ps`, `mnps`, and `iptw` objects returned by **twang** and for `ps.cont` objects returned by **twangContinuous**.
- [bal.tab.Match\(\)](#) for the method for objects returned by **Matching**.
- [bal.tab.optmatch\(\)](#) for the method for objects returned by **optmatch**.
- [bal.tab.cem.match\(\)](#) for the method for objects returned by **cem**.
- [bal.tab.CBPS\(\)](#) for the method for objects returned by **CBPS**.
- [bal.tab.ebalance\(\)](#) for the method for objects returned by **ebal**.
- [bal.tab.designmatch\(\)](#) for the method for objects returned by **designmatch**.
- [bal.tab.mimids\(\)](#) for the method for objects returned by **MatchThem**.
- [bal.tab.sbwcau\(\)](#) for the method for objects returned by **sbw**.
- [bal.tab.formula\(\)](#) and [bal.tab.data.frame\(\)](#) for the methods for `formula` and `data.frame` interfaces when the user has covariate values and weights (including matching weights) or subclasses or wants to evaluate balance on an unconditioned data set. For data that corresponds to a longitudinal treatment (i.e., to be analyzed with a marginal structural model), see [bal.tab.time.list\(\)](#).

See `vignette("faq")` for answers to frequently asked questions about `bal.tab()`.

Examples

```
## See individual pages above for examples with
## different inputs, or see `vignette("cobalt")`
```

bal.tab.CBPS

*Balance Statistics for CBPS Objects***Description**

Generates balance statistics for CBPS and CBMSM objects from the **CBPS** package.

Usage

```
## S3 method for class 'CBPS'
bal.tab(
  x,
  stats,
  int = FALSE,
  poly = 1,
  distance = NULL,
  add1 = NULL,
  data = NULL,
  continuous,
  binary,
  s.d.denom,
  thresholds = NULL,
  weights = NULL,
  cluster = NULL,
  imp = NULL,
  pairwise = TRUE,
  s.weights = NULL,
  abs = FALSE,
  subset = NULL,
  quick = TRUE,
  ...
)
```

Arguments

<code>x</code>	a CBPS or CBMSM object; the output of a call to <code>CBPS::CBPS()</code> or <code>CBPS::CBMSM()</code> .
<code>stats</code>	character; which statistic(s) should be reported. See <code>stats</code> for allowable options. For binary and multi-category treatments, "mean.diffs" (i.e., mean differences) is the default. For continuous treatments, "correlations" (i.e., treatment-covariate Pearson correlations) is the default. Multiple options are allowed.
<code>int</code>	logical or numeric; whether or not to include 2-way interactions of covariates included in <code>covs</code> and in <code>add1</code> . If numeric, will be passed to <code>poly</code> as well.
<code>poly</code>	numeric; the highest polynomial of each continuous covariate to display. For example, if 2, squares of each continuous covariate will be displayed (in addition to the covariate itself); if 3, squares and cubes of each continuous covariate will

	be displayed, etc. If 1, the default, only the base covariate will be displayed. If <code>int</code> is numeric, <code>poly</code> will take on the value of <code>int</code> .
<code>distance</code>	an optional formula or data frame containing distance values (e.g., propensity scores) or a character vector containing their names. If a formula or variable names are specified, <code>bal.tab()</code> will look in the argument to <code>data</code> , if specified. Propensity scores generated by <code>CBPS()</code> and <code>CBMSM()</code> are automatically included and named <code>"prop.score"</code> . For <code>CBMSM</code> objects, can be a list of allowable arguments, one for each time point, but each dataset in the list supplied to <code>distance</code> must have one row per individual, unlike the data frame in the original call to <code>CBMSM()</code> .
<code>add1</code>	an optional formula or data frame containing additional covariates for which to present balance or a character vector containing their names. If a formula or variable names are specified, <code>bal.tab()</code> will look in the arguments to the input object, <code>covs</code> , and <code>data</code> , if specified. For <code>CBMSM</code> objects, can be a list of allowable arguments, one for each time point, but each dataset in the list supplied to <code>add1</code> must have one row per individual, unlike the data frame in the original call to <code>CBMSM()</code> .
<code>data</code>	an optional data frame containing variables named in other arguments. For some input object types, this is required.
<code>continuous</code>	whether mean differences for continuous variables should be standardized (<code>"std"</code>) or raw (<code>"raw"</code>). Default <code>"std"</code> . Abbreviations allowed. This option can be set globally using <code>set.cobalt.options()</code> .
<code>binary</code>	whether mean differences for binary variables (i.e., difference in proportion) should be standardized (<code>"std"</code>) or raw (<code>"raw"</code>). Default <code>"raw"</code> . Abbreviations allowed. This option can be set globally using <code>set.cobalt.options()</code> .
<code>s.d.denom</code>	character; how the denominator for standardized mean differences should be calculated, if requested. See <code>col_w_smd()</code> for allowable options. Abbreviations allowed. If not specified, will be set to <code>"treated"</code> if the estimand of the call to <code>CBPS()</code> is the ATT and <code>"pooled"</code> if the estimand is the ATE.
<code>thresholds</code>	a named vector of balance thresholds, where the name corresponds to the statistic (i.e., in <code>stats</code>) that the threshold applies to. For example, to request thresholds on mean differences and variance ratios, one can set <code>thresholds = c(m = .05, v = 2)</code> . Requesting a threshold automatically requests the display of that statistic. When specified, extra columns are inserted into the Balance table describing whether the requested balance statistics exceeded the threshold or not. Summary tables tallying the number of variables that exceeded and were within the threshold and displaying the variables with the greatest imbalance on that balance measure are added to the output.
<code>weights</code>	a vector, list, or <code>data.frame</code> containing weights for each unit, or a string containing the names of the weights variables in <code>data</code> , or an object with a <code>get.w()</code> method or a list thereof. The weights can be, e.g., inverse probability weights or matching weights resulting from a matching algorithm.
<code>cluster</code>	either a vector containing cluster membership for each unit or a string containing the name of the cluster membership variable in <code>data</code> or the input object. See <code>class-bal.tab.cluster</code> for details.

imp	either a vector containing imputation indices for each unit or a string containing the name of the imputation index variable in data or the input object. See class-bal.tab.imp for details. Not necessary if data is a mids object.
pairwise	whether balance should be computed for pairs of treatments or for each treatment against all groups combined. See bal.tab.multi() for details. This can also be used with a binary treatment to assess balance with respect to the full sample.
s.weights	Optional; either a vector containing sampling weights for each unit or a string containing the name of the sampling weight variable in data. These function like regular weights except that both the adjusted and unadjusted samples will be weighted according to these weights if weights are used. If sampling weights are used in <code>CBPS()</code> , it is critical that they are specified in <code>bal.tab()</code> as well using the <code>s.weights</code> argument.
abs	logical; whether displayed balance statistics should be in absolute value or not.
subset	a logical or numeric vector denoting whether each observation should be included or which observations should be included. If logical, it should have length equal to the number of units. NAs will be treated as FALSE. This can be used as an alternative to <code>cluster</code> to examine balance on subsets of the data.
quick	logical; if TRUE, will not compute any values that will not be displayed. Set to FALSE if computed values not displayed will be used later.
...	for some input types, other arguments that are required or allowed. Otherwise, further arguments to control display of output. See display options for details.

Details

`bal.tab.CBPS()` and `bal.tab.CBMSM()` generate a list of balance summaries for the CBPS or CBMSM object given and functions similarly to [CBPS::balance\(\)](#).

Value

For point treatments, if clusters are not specified, an object of class "bal.tab" containing balance summaries for the CBPS object. See [bal.tab\(\)](#) for details.

If clusters are specified, an object of class "bal.tab.cluster" containing balance summaries within each cluster and a summary of balance across clusters. See [class-bal.tab.cluster](#) for details.

If `CBPS()` is used with multi-category treatments, an object of class "bal.tab.multi" containing balance summaries for each pairwise treatment comparison and a summary of balance across pairwise comparisons. See [bal.tab.multi\(\)](#) for details.

If `CBMSM()` is used for longitudinal treatments, an object of class "bal.tab.msm" containing balance summaries for each time period and a summary of balance across time periods. See [class-bal.tab.msm](#) for details.

See Also

- [bal.tab\(\)](#) for details of calculations.
- [class-bal.tab.cluster](#) for more information on clustered data.

- `bal.tab.multi()` for more information on multi-category treatments.
- `class-bal.tab.msm` for more information on longitudinal treatments.

Examples

```
data("lalonge", package = "cobalt")

## Using CBPS() for generating covariate balancing
## propensity score weights
library(CBPS)
cbps.out <- CBPS(treat ~ age + educ + married + race +
                 nodegree + re74 + re75,
                 data = lalonge)

bal.tab(cbps.out)
```

bal.tab.cem.match	<i>Balance Statistics for cem Objects</i>
-------------------	---

Description

Generates balance statistics for `cem.match` objects from **cem**.

Usage

```
## S3 method for class 'cem.match'
bal.tab(
  x,
  data,
  stats,
  int = FALSE,
  poly = 1,
  distance = NULL,
  add1 = NULL,
  continuous,
  binary,
  s.d.denom,
  thresholds = NULL,
  weights = NULL,
  cluster = NULL,
  imp = NULL,
  pairwise = TRUE,
  s.weights = NULL,
  abs = FALSE,
  subset = NULL,
  quick = TRUE,
  ...
)
```

Arguments

<code>x</code>	a <code>cem.match</code> or <code>cem.match.list</code> object; the output of a call to <code>cem::cem()</code> .
<code>data</code>	a data frame containing variables named in other arguments. An argument to <code>data</code> is required . It must be the same data used in the call to <code>cem()</code> or a <code>mids</code> object from which the data supplied to <code>datalist</code> in the <code>cem()</code> call originated.
<code>stats</code>	character; which statistic(s) should be reported. See stats for allowable options. For binary and multi-category treatments, "mean.diffs" (i.e., mean differences) is the default. For continuous treatments, "correlations" (i.e., treatment-covariate Pearson correlations) is the default. Multiple options are allowed.
<code>int</code>	logical or numeric; whether or not to include 2-way interactions of covariates included in <code>covs</code> and in <code>add1</code> . If numeric, will be passed to <code>poly</code> as well.
<code>poly</code>	numeric; the highest polynomial of each continuous covariate to display. For example, if 2, squares of each continuous covariate will be displayed (in addition to the covariate itself); if 3, squares and cubes of each continuous covariate will be displayed, etc. If 1, the default, only the base covariate will be displayed. If <code>int</code> is numeric, <code>poly</code> will take on the value of <code>int</code> .
<code>distance</code>	an optional formula or data frame containing distance values (e.g., propensity scores) or a character vector containing their names. If a formula or variable names are specified, <code>bal.tab()</code> will look in the argument to <code>data</code> , if specified. For longitudinal treatments, can be a list of allowable arguments, one for each time point.
<code>add1</code>	an optional formula or data frame containing additional covariates for which to present balance or a character vector containing their names. If a formula or variable names are specified, <code>bal.tab()</code> will look in the arguments to the input object, <code>covs</code> , and <code>data</code> , if specified. For longitudinal treatments, can be a list of allowable arguments, one for each time point.
<code>continuous</code>	whether mean differences for continuous variables should be standardized ("std") or raw ("raw"). Default "std". Abbreviations allowed. This option can be set globally using set.cobalt.options() .
<code>binary</code>	whether mean differences for binary variables (i.e., difference in proportion) should be standardized ("std") or raw ("raw"). Default "raw". Abbreviations allowed. This option can be set globally using set.cobalt.options() .
<code>s.d.denom</code>	character; how the denominator for standardized mean differences should be calculated, if requested. See col_w_smd() for allowable options. Abbreviations allowed. If not specified, will be set to "treated", where the treated group corresponds to the <code>baseline.group</code> in the call to <code>cem()</code> .
<code>thresholds</code>	a named vector of balance thresholds, where the name corresponds to the statistic (i.e., in <code>stats</code>) that the threshold applies to. For example, to request thresholds on mean differences and variance ratios, one can set <code>thresholds = c(m = .05, v = 2)</code> . Requesting a threshold automatically requests the display of that statistic. When specified, extra columns are inserted into the Balance table describing whether the requested balance statistics exceeded the threshold or not. Summary tables tallying the number of variables that exceeded and were within the threshold and displaying the variables with the greatest imbalance on that balance measure are added to the output.

weights	a vector, list, or data.frame containing weights for each unit, or a string containing the names of the weights variables in data, or an object with a <code>get.w()</code> method or a list thereof. The weights can be, e.g., inverse probability weights or matching weights resulting from a matching algorithm.
cluster	either a vector containing cluster membership for each unit or a string containing the name of the cluster membership variable in data or the input object. See class-bal.tab.cluster for details.
imp	either a vector containing imputation indices for each unit or a string containing the name of the imputation index variable in data or the input object. See class-bal.tab.imp for details. Not necessary if data is a mids object.
pairwise	whether balance should be computed for pairs of treatments or for each treatment against all groups combined. See bal.tab.multi() for details. This can also be used with a binary treatment to assess balance with respect to the full sample.
s.weights	Optional; either a vector containing sampling weights for each unit or a string containing the name of the sampling weight variable in data. These function like regular weights except that both the adjusted and unadjusted samples will be weighted according to these weights if weights are used.
abs	logical; whether displayed balance statistics should be in absolute value or not.
subset	a logical or numeric vector denoting whether each observation should be included or which observations should be included. If logical, it should have length equal to the number of units. NAs will be treated as FALSE. This can be used as an alternative to cluster to examine balance on subsets of the data.
quick	logical; if TRUE, will not compute any values that will not be displayed. Set to FALSE if computed values not displayed will be used later.
...	for some input types, other arguments that are required or allowed. Otherwise, further arguments to control display of output. See display options for details.

Details

`bal.tab.cem.match()` generates a list of balance summaries for the `cem.match` object given, and functions similarly to `cem::imbalance()`.

Value

If clusters and imputations are not specified, an object of class `"bal.tab"` containing balance summaries for the `cem.match` object. See [bal.tab\(\)](#) for details.

If imputations are specified, an object of class `"bal.tab.imp"` containing balance summaries for each imputation and a summary of balance across imputations. See [class-bal.tab.imp](#) for details.

If `cem()` is used with multi-category treatments, an object of class `"bal.tab.multi"` containing balance summaries for each pairwise treatment comparison. See [bal.tab.multi\(\)](#) for details.

If clusters are specified, an object of class `"bal.tab.cluster"` containing balance summaries within each cluster and a summary of balance across clusters. See [class-bal.tab.cluster](#) for details.

See Also

[bal.tab\(\)](#) for details of calculations.

Examples

```
data("lalonge", package = "cobalt")

## Coarsened exact matching
library(cem)
cem.out <- cem("treat",
              data = lalonge,
              drop = "re78")

bal.tab(cem.out, data = lalonge, un = TRUE,
        stats = c("m", "k"))
```

bal.tab.default

Balance Statistics for Other Objects

Description

Generates balance statistics using an object for which there is not a defined method.

Usage

```
## Default S3 method:
bal.tab(
  x,
  stats,
  int = FALSE,
  poly = 1,
  distance = NULL,
  add1 = NULL,
  data = NULL,
  continuous,
  binary,
  s.d.denom,
  thresholds = NULL,
  weights = NULL,
  cluster = NULL,
  imp = NULL,
  pairwise = TRUE,
  s.weights = NULL,
  abs = FALSE,
  subset = NULL,
  quick = TRUE,
  ...
)
```

Arguments

<code>x</code>	An object containing information about conditioning. See Details.
<code>stats</code>	character; which statistic(s) should be reported. See stats for allowable options. For binary and multi-category treatments, "mean.diffs" (i.e., mean differences) is the default. For continuous treatments, "correlations" (i.e., treatment-covariate Pearson correlations) is the default. Multiple options are allowed.
<code>int</code>	logical or numeric; whether or not to include 2-way interactions of covariates included in <code>covs</code> and in <code>add1</code> . If numeric, will be passed to <code>poly</code> as well.
<code>poly</code>	numeric; the highest polynomial of each continuous covariate to display. For example, if 2, squares of each continuous covariate will be displayed (in addition to the covariate itself); if 3, squares and cubes of each continuous covariate will be displayed, etc. If 1, the default, only the base covariate will be displayed. If <code>int</code> is numeric, <code>poly</code> will take on the value of <code>int</code> .
<code>distance</code>	an optional formula or data frame containing distance values (e.g., propensity scores) or a character vector containing their names. If a formula or variable names are specified, <code>bal.tab()</code> will look in the argument to <code>data</code> , if specified. For longitudinal treatments, can be a list of allowable arguments, one for each time point.
<code>add1</code>	an optional formula or data frame containing additional covariates for which to present balance or a character vector containing their names. If a formula or variable names are specified, <code>bal.tab()</code> will look in the arguments to the input object, <code>covs</code> , and <code>data</code> , if specified. For longitudinal treatments, can be a list of allowable arguments, one for each time point.
<code>data</code>	an optional data frame containing variables named in other arguments. For some input object types, this is required.
<code>continuous</code>	whether mean differences for continuous variables should be standardized ("std") or raw ("raw"). Default "std". Abbreviations allowed. This option can be set globally using set.cobalt.options() .
<code>binary</code>	whether mean differences for binary variables (i.e., difference in proportion) should be standardized ("std") or raw ("raw"). Default "raw". Abbreviations allowed. This option can be set globally using set.cobalt.options() .
<code>s.d.denom</code>	character; how the denominator for standardized mean differences should be calculated, if requested. See col_w_smd() for allowable options. If weights are supplied, each set of weights should have a corresponding entry to <code>s.d.denom</code> . Abbreviations allowed. If left blank and weights, subclasses, or matching strata are supplied, <code>bal.tab()</code> will figure out which one is best based on the estimand, if given (for ATT, "treated"; for ATC, "control"; otherwise "pooled") and other clues if not.
<code>thresholds</code>	a named vector of balance thresholds, where the name corresponds to the statistic (i.e., in <code>stats</code>) that the threshold applies to. For example, to request thresholds on mean differences and variance ratios, one can set <code>thresholds = c(m = .05, v = 2)</code> . Requesting a threshold automatically requests the display of that statistic. When specified, extra columns are inserted into the Balance table describing whether the requested balance statistics exceeded the threshold or not.

	Summary tables tallying the number of variables that exceeded and were within the threshold and displaying the variables with the greatest imbalance on that balance measure are added to the output.
weights	a vector, list, or <code>data.frame</code> containing weights for each unit, or a string containing the names of the weights variables in data, or an object with a <code>get.w()</code> method or a list thereof. The weights can be, e.g., inverse probability weights or matching weights resulting from a matching algorithm.
cluster	either a vector containing cluster membership for each unit or a string containing the name of the cluster membership variable in data or the input object. See <code>class-bal.tab.cluster</code> for details.
imp	either a vector containing imputation indices for each unit or a string containing the name of the imputation index variable in data or the input object. See <code>class-bal.tab.imp</code> for details. Not necessary if data is a <code>mids</code> object.
pairwise	whether balance should be computed for pairs of treatments or for each treatment against all groups combined. See <code>bal.tab.multi()</code> for details. This can also be used with a binary treatment to assess balance with respect to the full sample.
s.weights	Optional; either a vector containing sampling weights for each unit or a string containing the name of the sampling weight variable in data. These function like regular weights except that both the adjusted and unadjusted samples will be weighted according to these weights if weights are used.
abs	logical; whether displayed balance statistics should be in absolute value or not.
subset	a logical or numeric vector denoting whether each observation should be included or which observations should be included. If logical, it should have length equal to the number of units. NAs will be treated as FALSE. This can be used as an alternative to <code>cluster</code> to examine balance on subsets of the data.
quick	logical; if TRUE, will not compute any values that will not be displayed. Set to FALSE if computed values not displayed will be used later.
...	other arguments that would be passed to <code>bal.tab.formula()</code> , <code>bal.tab.data.frame()</code> , or <code>bal.tab.time.list()</code> . See Details.

Details

`bal.tab.default()` processes its input and attempt to extract enough information from it to display covariate balance for `x`. The purpose of this method is to allow users who have created their own objects containing conditioning information (i.e., weights, subclasses, treatments, covariates, etc.) to access the capabilities of `bal.tab()` without having a special method written for them. By including the correct items in `x`, `bal.tab.default()` can present balance tables as if the input was the output of one of the specifically supported packages (e.g., **MatchIt**, **twang**, etc.).

The function will search `x` for the following named items and attempt to process them:

`treat` A vector (numeric, character, factor) containing the values of the treatment for each unit or the name of the column in data containing them. Essentially the same input to `treat` in `bal.tab.data.frame()`.

`treat.list` A list of vectors (numeric, character, factor) containing, for each time point, the values of the treatment for each unit or the name of the column in data containing them. Essentially the same input to `treat.list` in `bal.tab.time.list()`.

- `covs` A data.frame containing the values of the covariates for each unit. Essentially the same input to `covs` in `bal.tab.data.frame()`.
- `covs.list` A list of data.frames containing, for each time point, the values of the covariates for each unit. Essentially the same input to `covs.list` in `bal.tab.time.list()`.
- `formula` A formula with the treatment variable as the response and the covariates for which balance is to be assessed as the terms. Essentially the same input to `formula` in `bal.tab.formula()`.
- `formula.list` A list of formulas with, for each time point, the treatment variable as the response and the covariates for which balance is to be assessed as the terms. Essentially the same input to `formula.list` in `bal.tab.time.list()`.
- `data` A data.frame containing variables with the names used in other arguments and components (e.g., `formula`, `weights`, etc.). Essentially the same input to `data` in `bal.tab.formula()`, `bal.tab.data.frame()`, or `bal.tab.time.list()`.
- `weights` A vector, list, or data.frame containing weights for each unit or a string containing the names of the weights variables in `data`. Essentially the same input to `weights` in `bal.tab.data.frame()` or `bal.tab.time.list()`.
- `distance` A vector, formula, or data frame containing distance values (e.g., propensity scores) or a character vector containing their names. If a formula or variable names are specified, `bal.tab()` will look in the argument to `data`, if specified. Essentially the same input to `distance` in `bal.tab.data.frame()`.
- `formula.list` A list of vectors or data.frames containing, for each time point, distance values (e.g., propensity scores) for each unit or a string containing the name of the distance variable in `data`. Essentially the same input to `distance.list` in `bal.tab.time.list()`.
- `subclass` A vector containing subclass membership for each unit or a string containing the name of the subclass variable in `data`. Essentially the same input to `subclass` in `bal.tab.data.frame()`.
- `match.strata` A vector containing matching stratum membership for each unit or a string containing the name of the matching stratum variable in `data`. Essentially the same input to `match.strata` in `bal.tab.data.frame()`.
- `estimand` A character vector; whether the desired estimand is the "ATT", "ATC", or "ATE" for each set of weights. Essentially the same input to `estimand` in `bal.tab.data.frame()`.
- `s.weights` A vector containing sampling weights for each unit or a string containing the name of the sampling weight variable in `data`. Essentially the same input to `s.weights` in `bal.tab.data.frame()` or `bal.tab.time.list()`.
- `focal` The name of the focal treatment when multi-category treatments are used. Essentially the same input to `focal` in `bal.tab.data.frame()`.
- `call` A call object containing the function call, usually generated by using `match.call()` inside the function that created `x`.

Any of these items can also be supplied directly to `bal.tab.default`, e.g., `bal.tab.default(x, formula = treat ~ x1 + x2)`. If supplied, it will override the object with the same role in `x`. In addition, any arguments to `bal.tab.formula()`, `bal.tab.data.frame()`, and `bal.tab.time.list()` are allowed and perform the same function.

At least some inputs containing information to create the treatment and covariates are required (e.g., `formula` and `data` or `covs` and `treat`). All other arguments are optional and have the same defaults as those in `bal.tab.data.frame()` or `bal.tab.time.list()`. If `treat.list`, `covs.list`, or `formula.list` are supplied in `x` or as an argument to `bal.tab.default()`, the function will

proceed considering a longitudinal treatment. Otherwise, it will proceed considering a point treatment.

`bal.tab.default()`, like other `bal.tab()` methods, is just a shortcut to supply arguments to `bal.tab.data.frame()` or `bal.tab.time.list()`. Therefore, any matters regarding argument priority or function are described in the documentation for these methods.

Value

For point treatments, if clusters and imputations are not specified, an object of class `"bal.tab"` containing balance summaries for the specified treatment and covariates. See [bal.tab\(\)](#) for details.

If clusters are specified, an object of class `"bal.tab.cluster"` containing balance summaries within each cluster and a summary of balance across clusters. See [class-bal.tab.cluster](#) for details.

If imputations are specified, an object of class `"bal.tab.imp"` containing balance summaries for each imputation and a summary of balance across imputations, just as with clusters. See [class-bal.tab.imp](#) for details.

If multi-category treatments are used, an object of class `"bal.tab.multi"` containing balance summaries for each pairwise treatment comparison and a summary of balance across pairwise comparisons. See [bal.tab.multi\(\)](#) for details.

If longitudinal treatments are used, an object of class `"bal.tab.msm"` containing balance summaries at each time point. Each balance summary is its own `bal.tab` object. See [class-bal.tab.msm](#) for more details.

See Also

- [bal.tab.formula\(\)](#) and [bal.tab.time.list\(\)](#) for additional arguments to be supplied.
- [bal.tab\(\)](#) for output and details of calculations.
- [class-bal.tab.cluster](#) for more information on clustered data.
- [class-bal.tab.imp](#) for more information on multiply imputed data.
- [bal.tab.multi\(\)](#) for more information on multi-category treatments.

Examples

```
data("lalonge", package = "cobalt")
covs <- subset(lalonge, select = -c(treat, re78))

##Writing a function the produces output for direct
##use in bal.tab.default

ate.weights <- function(treat, covs) {
  data <- data.frame(treat, covs)
  formula <- formula(data)
  ps <- glm(formula, data = data,
            family = "binomial")$fitted.values
  weights <- treat/ps + (1-treat)/(1-ps)
  call <- match.call()
  out <- list(treat = treat,
             covs = covs,
```

```

        distance = ps,
        weights = weights,
        estimand = "ATE",
        call = call)
    return(out)
}

out <- ate.weights(lalonde$treat, covs)

bal.tab(out, un = TRUE)

```

bal.tab.designmatch *Balance Statistics for designmatch Objects*

Description

Generates balance statistics for output objects from **designmatch**.

Usage

```

## S3 method for class 'designmatch'
bal.tab(
  x,
  formula = NULL,
  data = NULL,
  treat = NULL,
  covs = NULL,
  stats,
  int = FALSE,
  poly = 1,
  distance = NULL,
  addl = NULL,
  continuous,
  binary,
  s.d.denom,
  thresholds = NULL,
  weights = NULL,
  cluster = NULL,
  imp = NULL,
  pairwise = TRUE,
  s.weights = NULL,
  abs = FALSE,
  subset = NULL,
  quick = TRUE,
  ...
)

```

Arguments

x	the output of a call to <code>designmatch::bmatch()</code> or related wrapper functions from the designmatch package.
formula	a formula with the treatment variable as the response and the covariates for which balance is to be assessed as the predictors. All named variables must be in data. See Details.
data	a data frame containing variables named in formula, if supplied, and other arguments.
treat	a vector of treatment statuses. See Details.
covs	a data frame of covariate values for which to check balance. See Details.
stats	character; which statistic(s) should be reported. See <code>stats</code> for allowable options. For binary and multi-category treatments, "mean.diffs" (i.e., mean differences) is the default. For continuous treatments, "correlations" (i.e., treatment-covariate Pearson correlations) is the default. Multiple options are allowed.
int	logical or numeric; whether or not to include 2-way interactions of covariates included in covs and in addl. If numeric, will be passed to poly as well.
poly	numeric; the highest polynomial of each continuous covariate to display. For example, if 2, squares of each continuous covariate will be displayed (in addition to the covariate itself); if 3, squares and cubes of each continuous covariate will be displayed, etc. If 1, the default, only the base covariate will be displayed. If int is numeric, poly will take on the value of int.
distance	an optional formula or data frame containing distance values (e.g., propensity scores) or a character vector containing their names. If a formula or variable names are specified, <code>bal.tab()</code> will look in the argument to data, if specified. For longitudinal treatments, can be a list of allowable arguments, one for each time point.
addl	an optional formula or data frame containing additional covariates for which to present balance or a character vector containing their names. If a formula or variable names are specified, <code>bal.tab()</code> will look in the arguments to the input object, covs, and data, if specified. For longitudinal treatments, can be a list of allowable arguments, one for each time point.
continuous	whether mean differences for continuous variables should be standardized ("std") or raw ("raw"). Default "std". Abbreviations allowed. This option can be set globally using <code>set.cobalt.options()</code> .
binary	whether mean differences for binary variables (i.e., difference in proportion) should be standardized ("std") or raw ("raw"). Default "raw". Abbreviations allowed. This option can be set globally using <code>set.cobalt.options()</code> .
s.d.denom	character; how the denominator for standardized mean differences should be calculated, if requested. See <code>col_w_smd()</code> for allowable options. Abbreviations allowed. If not specified, will be set to "treated".
thresholds	a named vector of balance thresholds, where the name corresponds to the statistic (i.e., in stats) that the threshold applies to. For example, to request thresholds on mean differences and variance ratios, one can set <code>thresholds = c(m =</code>

	.05, v = 2). Requesting a threshold automatically requests the display of that statistic. When specified, extra columns are inserted into the Balance table describing whether the requested balance statistics exceeded the threshold or not. Summary tables tallying the number of variables that exceeded and were within the threshold and displaying the variables with the greatest imbalance on that balance measure are added to the output.
weights	a vector, list, or data.frame containing weights for each unit, or a string containing the names of the weights variables in data, or an object with a <code>get.w()</code> method or a list thereof. The weights can be, e.g., inverse probability weights or matching weights resulting from a matching algorithm.
cluster	either a vector containing cluster membership for each unit or a string containing the name of the cluster membership variable in data or the input object. See <code>class-bal.tab.cluster</code> for details.
imp	either a vector containing imputation indices for each unit or a string containing the name of the imputation index variable in data or the input object. See <code>class-bal.tab.imp</code> for details. Not necessary if data is a mids object.
pairwise	whether balance should be computed for pairs of treatments or for each treatment against all groups combined. See <code>bal.tab.multi()</code> for details. This can also be used with a binary treatment to assess balance with respect to the full sample.
s.weights	Optional; either a vector containing sampling weights for each unit or a string containing the name of the sampling weight variable in data. These function like regular weights except that both the adjusted and unadjusted samples will be weighted according to these weights if weights are used.
abs	logical; whether displayed balance statistics should be in absolute value or not.
subset	a logical or numeric vector denoting whether each observation should be included or which observations should be included. If logical, it should have length equal to the number of units. NAs will be treated as FALSE. This can be used as an alternative to cluster to examine balance on subsets of the data.
quick	logical; if TRUE, will not compute any values that will not be displayed. Set to FALSE if computed values not displayed will be used later.
...	for some input types, other arguments that are required or allowed. Otherwise, further arguments to control display of output. See <code>display options</code> for details.

Details

`bal.tab()` generates a list of balance summaries for the object given, and functions similarly to `designmatch::meantab()`. Note that output objects from **designmatch** do not have their own class; `bal.tab()` first checks whether the object meets the criteria to be treated as a designmatch object before dispatching the correct method. Renaming or removing items from the output object can create unintended consequences.

The input to `bal.tab.designmatch()` must include either both formula and data or both covs and treat. Using the covs + treat input mirrors how `designmatch::meantab()` is used (note that to see identical results to `meantab()`, `s.d.denom` must be set to "pooled").

Value

If clusters and imputations are not specified, an object of class "bal.tab" containing balance summaries for the given object. See [bal.tab\(\)](#) for details.

If clusters are specified, an object of class "bal.tab.cluster" containing balance summaries within each cluster and a summary of balance across clusters. See [class-bal.tab.cluster](#) for details.

See Also

[bal.tab\(\)](#) for details of calculations.

Examples

```
## Not run:
library(designmatch)
data("lalonge", package = "cobalt")

covariates <- as.matrix(lalonge[c("age", "educ", "re74", "re75")])
treat <- lalonge$treat
dmout <- bmatch(treat,
               total_groups = sum(treat == 1),
               mom = list(covs = covariates,
                         tols = absstdif(covariates,
                                         treat, .05))
               )

## Using treat and covs
bal.tab(dmout, treat = treat, covs = covariates)

## End(Not run)
```

bal.tab.ebalance

Balance Statistics for ebalance Objects

Description

Generates balance statistics for output objects from **ebal**.

Usage

```
## S3 method for class 'ebalance'
bal.tab(
  x,
  formula = NULL,
  data = NULL,
  treat = NULL,
  covs = NULL,
```

```

stats,
int = FALSE,
poly = 1,
distance = NULL,
add1 = NULL,
continuous,
binary,
s.d.denom,
thresholds = NULL,
weights = NULL,
cluster = NULL,
imp = NULL,
pairwise = TRUE,
s.weights = NULL,
abs = FALSE,
subset = NULL,
quick = TRUE,
...
)

```

Arguments

x	an ebalance object (the output of a call to <code>ebal::ebalance()</code> or <code>ebal::ebalance.trim()</code>).
formula	a formula with the treatment variable as the response and the covariates for which balance is to be assessed as the predictors. All named variables must be in data. See Details.
data	a data frame containing variables named in formula, if supplied, and other arguments.
treat	a vector of treatment statuses. See Details.
covs	a data frame of covariate values for which to check balance. See Details.
stats	character; which statistic(s) should be reported. See stats for allowable options. For binary and multi-category treatments, "mean.diffs" (i.e., mean differences) is the default. For continuous treatments, "correlations" (i.e., treatment-covariate Pearson correlations) is the default. Multiple options are allowed.
int	logical or numeric; whether or not to include 2-way interactions of covariates included in covs and in add1. If numeric, will be passed to poly as well.
poly	numeric; the highest polynomial of each continuous covariate to display. For example, if 2, squares of each continuous covariate will be displayed (in addition to the covariate itself); if 3, squares and cubes of each continuous covariate will be displayed, etc. If 1, the default, only the base covariate will be displayed. If int is numeric, poly will take on the value of int.
distance	an optional formula or data frame containing distance values (e.g., propensity scores) or a character vector containing their names. If a formula or variable names are specified, <code>bal.tab()</code> will look in the argument to data, if specified. For longitudinal treatments, can be a list of allowable arguments, one for each time point.

addl	an optional formula or data frame containing additional covariates for which to present balance or a character vector containing their names. If a formula or variable names are specified, <code>bal.tab()</code> will look in the arguments to the input object, <code>covs</code> , and <code>data</code> , if specified. For longitudinal treatments, can be a list of allowable arguments, one for each time point.
continuous	whether mean differences for continuous variables should be standardized ("std") or raw ("raw"). Default "std". Abbreviations allowed. This option can be set globally using <code>set.cobalt.options()</code> .
binary	whether mean differences for binary variables (i.e., difference in proportion) should be standardized ("std") or raw ("raw"). Default "raw". Abbreviations allowed. This option can be set globally using <code>set.cobalt.options()</code> .
s.d.denom	character; how the denominator for standardized mean differences should be calculated, if requested. See <code>col_w_smd()</code> for allowable options. Abbreviations allowed. If not specified, will be set to "treated".
thresholds	a named vector of balance thresholds, where the name corresponds to the statistic (i.e., in stats) that the threshold applies to. For example, to request thresholds on mean differences and variance ratios, one can set <code>thresholds = c(m = .05, v = 2)</code> . Requesting a threshold automatically requests the display of that statistic. When specified, extra columns are inserted into the Balance table describing whether the requested balance statistics exceeded the threshold or not. Summary tables tallying the number of variables that exceeded and were within the threshold and displaying the variables with the greatest imbalance on that balance measure are added to the output.
weights	a vector, list, or <code>data.frame</code> containing weights for each unit, or a string containing the names of the weights variables in <code>data</code> , or an object with a <code>get.w()</code> method or a list thereof. The weights can be, e.g., inverse probability weights or matching weights resulting from a matching algorithm.
cluster	either a vector containing cluster membership for each unit or a string containing the name of the cluster membership variable in <code>data</code> or the input object. See <code>class-bal.tab.cluster</code> for details.
imp	either a vector containing imputation indices for each unit or a string containing the name of the imputation index variable in <code>data</code> or the input object. See <code>class-bal.tab.imp</code> for details. Not necessary if <code>data</code> is a <code>mids</code> object.
pairwise	whether balance should be computed for pairs of treatments or for each treatment against all groups combined. See <code>bal.tab.multi()</code> for details. This can also be used with a binary treatment to assess balance with respect to the full sample.
s.weights	Optional; either a vector containing sampling weights for each unit or a string containing the name of the sampling weight variable in <code>data</code> . These function like regular weights except that both the adjusted and unadjusted samples will be weighted according to these weights if weights are used.
abs	logical; whether displayed balance statistics should be in absolute value or not.
subset	a logical or numeric vector denoting whether each observation should be included or which observations should be included. If logical, it should have length equal to the number of units. NAs will be treated as FALSE. This can be used as an alternative to <code>cluster</code> to examine balance on subsets of the data.

- quick logical; if TRUE, will not compute any values that will not be displayed. Set to FALSE if computed values not displayed will be used later.
- ... for some input types, other arguments that are required or allowed. Otherwise, further arguments to control display of output. See [display options](#) for details.

Details

`bal.tab()` generates a list of balance summaries for the object given. The input to `bal.tab.ebalance()` must include either both `formula` and `data` or both `covs` and `treat`.

Value

If clusters and imputations are not specified, an object of class "bal.tab" containing balance summaries for the given object. See [bal.tab\(\)](#) for details.

If clusters are specified, an object of class "bal.tab.cluster" containing balance summaries within each cluster and a summary of balance across clusters. See [class-bal.tab.cluster](#) for details.

See Also

[bal.tab\(\)](#) for details of calculations.

Examples

```
data("lalonge", package = "cobalt")

covs <- subset(lalonge, select = -c(re78, treat))
covs0 <- splitfactor(covs)

e.out <- ebal::ebalance(lalonge$treat, covs0)

## Using formula and data
bal.tab(e.out, formula = treat ~ age + educ + race +
        married + nodegree + re74 + re75,
        data = lalonge)

## Using treat and covs
bal.tab(e.out, treat = lalonge$treat, covs = covs)
```

bal.tab.formula	<i>Balance Statistics for Data Sets</i>
-----------------	---

Description

Generates balance statistics for unadjusted, matched, weighted, or stratified data using either a `data.frame` or `formula` interface.

Usage

```
## S3 method for class 'formula'
bal.tab(
  x,
  data = NULL,
  stats,
  int = FALSE,
  poly = 1,
  distance = NULL,
  add1 = NULL,
  continuous,
  binary,
  s.d.denom,
  thresholds = NULL,
  weights = NULL,
  cluster = NULL,
  imp = NULL,
  pairwise = TRUE,
  s.weights = NULL,
  abs = FALSE,
  subset = NULL,
  quick = TRUE,
  subclass = NULL,
  match.strata = NULL,
  method,
  estimand = NULL,
  focal = NULL,
  ...
)

## S3 method for class 'data.frame'
bal.tab(
  x,
  treat,
  stats,
  int = FALSE,
  poly = 1,
  distance = NULL,
  add1 = NULL,
  data = NULL,
  continuous,
  binary,
  s.d.denom,
  thresholds = NULL,
  weights = NULL,
  cluster = NULL,
  imp = NULL,
  pairwise = TRUE,
```

```

    s.weights = NULL,
    abs = FALSE,
    subset = NULL,
    quick = TRUE,
    subclass = NULL,
    match.strata = NULL,
    method,
    estimand = NULL,
    focal = NULL,
    ...
)

## S3 method for class 'matrix'
bal.tab(
  x,
  treat,
  stats,
  int = FALSE,
  poly = 1,
  distance = NULL,
  add1 = NULL,
  data = NULL,
  continuous,
  binary,
  s.d.denom,
  thresholds = NULL,
  weights = NULL,
  cluster = NULL,
  imp = NULL,
  pairwise = TRUE,
  s.weights = NULL,
  abs = FALSE,
  subset = NULL,
  quick = TRUE,
  subclass = NULL,
  match.strata = NULL,
  method,
  estimand = NULL,
  focal = NULL,
  ...
)

```

Arguments

x either a `data.frame` containing covariate values for each unit or a formula with the treatment variable as the response and the covariates for which balance is to be assessed as the terms. If a formula is supplied, all terms must be present as variable names in data or the global environment.

data	an optional data frame containing variables named in other arguments. For some input object types, this is required.
stats	character; which statistic(s) should be reported. See stats for allowable options. For binary and multi-category treatments, "mean.diffs" (i.e., mean differences) is the default. For continuous treatments, "correlations" (i.e., treatment-covariate Pearson correlations) is the default. Multiple options are allowed.
int	logical or numeric; whether or not to include 2-way interactions of covariates included in covs and in addl. If numeric, will be passed to poly as well.
poly	numeric; the highest polynomial of each continuous covariate to display. For example, if 2, squares of each continuous covariate will be displayed (in addition to the covariate itself); if 3, squares and cubes of each continuous covariate will be displayed, etc. If 1, the default, only the base covariate will be displayed. If int is numeric, poly will take on the value of int.
distance	an optional formula or data frame containing distance values (e.g., propensity scores) or a character vector containing their names. If a formula or variable names are specified, bal.tab() will look in the argument to data, if specified. For longitudinal treatments, can be a list of allowable arguments, one for each time point.
addl	an optional formula or data frame containing additional covariates for which to present balance or a character vector containing their names. If a formula or variable names are specified, bal.tab() will look in the arguments to the input object, covs, and data, if specified. For longitudinal treatments, can be a list of allowable arguments, one for each time point.
continuous	whether mean differences for continuous variables should be standardized ("std") or raw ("raw"). Default "std". Abbreviations allowed. This option can be set globally using set.cobalt.options() .
binary	whether mean differences for binary variables (i.e., difference in proportion) should be standardized ("std") or raw ("raw"). Default "raw". Abbreviations allowed. This option can be set globally using set.cobalt.options() .
s.d.denom	character; how the denominator for standardized mean differences should be calculated, if requested. See col_w_smd() for allowable options. Abbreviations allowed. If weights are supplied, each set of weights should have a corresponding entry to s.d.denom; a single entry will be recycled to all sets of weights. If left blank and one of weights, subclass, or match.strata are supplied, bal.tab() will figure out which one is best based on estimand, if given (for ATT, "treated"; for ATC, "control"; otherwise "pooled") and other clues if not.
thresholds	a named vector of balance thresholds, where the name corresponds to the statistic (i.e., in stats) that the threshold applies to. For example, to request thresholds on mean differences and variance ratios, one can set thresholds = c(m = .05, v = 2). Requesting a threshold automatically requests the display of that statistic. When specified, extra columns are inserted into the Balance table describing whether the requested balance statistics exceeded the threshold or not. Summary tables tallying the number of variables that exceeded and were within the threshold and displaying the variables with the greatest imbalance on that balance measure are added to the output.

weights	a vector, list, or data.frame containing weights for each unit, or a string containing the names of the weights variables in data, or an object with a <code>get.w()</code> method or a list thereof. The weights can be, e.g., inverse probability weights or matching weights resulting from a matching algorithm.
cluster	either a vector containing cluster membership for each unit or a string containing the name of the cluster membership variable in data or the input object. See class-bal.tab.cluster for details.
imp	either a vector containing imputation indices for each unit or a string containing the name of the imputation index variable in data or the input object. See class-bal.tab.imp for details. Not necessary if data is a mids object.
pairwise	whether balance should be computed for pairs of treatments or for each treatment against all groups combined. See bal.tab.multi() for details. This can also be used with a binary treatment to assess balance with respect to the full sample.
s.weights	Optional; either a vector containing sampling weights for each unit or a string containing the name of the sampling weight variable in data. These function like regular weights except that both the adjusted and unadjusted samples will be weighted according to these weights if weights are used.
abs	logical; whether displayed balance statistics should be in absolute value or not.
subset	a logical or numeric vector denoting whether each observation should be included or which observations should be included. If logical, it should have length equal to the number of units. NAs will be treated as FALSE. This can be used as an alternative to cluster to examine balance on subsets of the data.
quick	logical; if TRUE, will not compute any values that will not be displayed. Set to FALSE if computed values not displayed will be used later.
subclass	optional; either a vector containing subclass membership for each unit or a string containing the name of the subclass variable in data.
match.strata	optional; either a vector containing matching stratum membership for each unit or a string containing the name of the matching stratum variable in data. See Details.
method	character; the method of adjustment, if any. If weights are specified, the user can specify either "matching" or "weighting"; "weighting" is the default. If multiple sets of weights are used, each must have a corresponding value for method, but if they are all of the same type, only one value is required. If subclass is specified, "subclassification" is the default. Abbreviations allowed. The only distinction between "matching" and "weighting" is how sample sizes are displayed.
estimand	character; whether the desired estimand is the "ATT", "ATC", or "ATE" for each set of weights. This argument can be used in place of s.d.denom to specify how standardized differences are calculated.
focal	the name of the focal treatment when multi-category treatments are used. See bal.tab.multi() for details.
...	for some input types, other arguments that are required or allowed. Otherwise, further arguments to control display of output. See display options for details.

`treat` either a vector containing treatment status values for each unit or a string containing the name of the treatment variable in `data`. Required for the `data.frame` method.

Details

`bal.tab.data.frame()` generates a list of balance summaries for the covariates and treatment status values given. `bal.tab.formula()` does the same but uses a formula interface instead. When the formula interface is used, the formula and data are reshaped into a treatment vector and `data.frame` of covariates and then simply passed through the `data.frame` method.

If `weights`, `subclass` and `match.strata` are all `NULL`, balance information will be presented only for the unadjusted sample.

The argument to `match.strata` corresponds to a factor vector containing the name or index of each pair/stratum for units conditioned through matching, for example, using the **optmatch** package. If more than one of `weights`, `subclass`, or `match.strata` are specified, `bal.tab()` will attempt to figure out which one to apply. Currently only one of these can be applied at a time. `bal.tab()` behaves differently depending on whether subclasses are used in conditioning or not. If they are used, `bal.tab()` creates balance statistics for each subclass and for the sample in aggregate. See [class-bal.tab.subclass](#) for more information.

Multiple sets of weights can be supplied simultaneously by entering a `data.frame` or a character vector containing the names of weight variables found in `data` or a list of weights vectors or names. The arguments to `method`, `s.d.denom`, and `estimand`, if any, must be either the same length as the number of sets of weights or of length one, where the sole entry is applied to all sets. When standardized differences are computed for the unadjusted group, they are done using the first entry to `s.d.denom` or `estimand`. When only one set of weights is supplied, the output for the adjusted group will simply be called "Adj", but otherwise will be named after each corresponding set of weights. Specifying multiple sets of weights will also add components to other outputs of `bal.tab()`.

Value

For point treatments, if clusters and imputations are not specified, an object of class "`bal.tab`" containing balance summaries for the specified treatment and covariates. See [bal.tab\(\)](#) for details.

If imputations are specified, an object of class "`bal.tab.imp`" containing balance summaries for each imputation and a summary of balance across imputations. See [class-bal.tab.imp](#) for details.

If multi-category treatments are used, an object of class "`bal.tab.multi`" containing balance summaries for each pairwise treatment comparison. See [bal.tab.multi\(\)](#) for details.

If clusters are specified, an object of class "`bal.tab.cluster`" containing balance summaries within each cluster and a summary of balance across clusters. See [class-bal.tab.cluster](#) for details.

See Also

- [bal.tab\(\)](#) for details of calculations.
- [class-bal.tab.cluster](#) for more information on clustered data.
- [class-bal.tab.imp](#) for more information on multiply imputed data.
- [bal.tab.multi\(\)](#) for more information on multi-category treatments.

Examples

```
data("lalonge", package = "cobalt")
lalonge$p.score <- glm(treat ~ age + educ + race, data = lalonge,
                      family = "binomial")$fitted.values
covariates <- subset(lalonge, select = c(age, educ, race))

## Propensity score weighting using IPTW
lalonge$iptw.weights <- ifelse(lalonge$treat==1,
                              1/lalonge$p.score,
                              1/(1-lalonge$p.score))

# data frame interface:
bal.tab(covariates, treat = "treat", data = lalonge,
        weights = "iptw.weights", s.d.denom = "pooled")

# Formula interface:
bal.tab(treat ~ age + educ + race, data = lalonge,
        weights = "iptw.weights", s.d.denom = "pooled")

## Propensity score subclassification
lalonge$subclass <- findInterval(lalonge$p.score,
                                quantile(lalonge$p.score,
                                           (0:6)/6), all.inside = TRUE)

# data frame interface:
bal.tab(covariates, treat = "treat", data = lalonge,
        subclass = "subclass", disp.subclass = TRUE,
        s.d.denom = "pooled")

# Formula interface:
bal.tab(treat ~ age + educ + race, data = lalonge,
        subclass = "subclass", disp.subclass = TRUE,
        s.d.denom = "pooled")
```

bal.tab.Match

Balance Statistics for Matching Objects

Description

Generates balance statistics for output objects from **Matching**.

Usage

```
## S3 method for class 'Match'
bal.tab(
  x,
  formula = NULL,
  data = NULL,
  treat = NULL,
```

```

    covs = NULL,
    stats,
    int = FALSE,
    poly = 1,
    distance = NULL,
    addl = NULL,
    continuous,
    binary,
    s.d.denom,
    thresholds = NULL,
    weights = NULL,
    cluster = NULL,
    imp = NULL,
    pairwise = TRUE,
    s.weights = NULL,
    abs = FALSE,
    subset = NULL,
    quick = TRUE,
    ...
)

```

Arguments

x	a Match object (the output of a call to <code>Matching::Match()</code> or <code>Matching::Matchby()</code>).
formula	a formula with the treatment variable as the response and the covariates for which balance is to be assessed as the predictors. All named variables must be in data. See Details.
data	a data frame containing variables named in formula, if supplied, and other arguments.
treat	a vector of treatment statuses. See Details.
covs	a data frame of covariate values for which to check balance. See Details.
stats	character; which statistic(s) should be reported. See <code>stats</code> for allowable options. For binary and multi-category treatments, "mean.diffs" (i.e., mean differences) is the default. For continuous treatments, "correlations" (i.e., treatment-covariate Pearson correlations) is the default. Multiple options are allowed.
int	logical or numeric; whether or not to include 2-way interactions of covariates included in covs and in addl. If numeric, will be passed to poly as well.
poly	numeric; the highest polynomial of each continuous covariate to display. For example, if 2, squares of each continuous covariate will be displayed (in addition to the covariate itself); if 3, squares and cubes of each continuous covariate will be displayed, etc. If 1, the default, only the base covariate will be displayed. If int is numeric, poly will take on the value of int.
distance	an optional formula or data frame containing distance values (e.g., propensity scores) or a character vector containing their names. If a formula or variable names are specified, <code>bal.tab()</code> will look in the argument to data, if specified.

	For longitudinal treatments, can be a list of allowable arguments, one for each time point.
addl	an optional formula or data frame containing additional covariates for which to present balance or a character vector containing their names. If a formula or variable names are specified, <code>bal.tab()</code> will look in the arguments to the input object, <code>covs</code> , and <code>data</code> , if specified. For longitudinal treatments, can be a list of allowable arguments, one for each time point.
continuous	whether mean differences for continuous variables should be standardized ("std") or raw ("raw"). Default "std". Abbreviations allowed. This option can be set globally using <code>set.cobalt.options()</code> .
binary	whether mean differences for binary variables (i.e., difference in proportion) should be standardized ("std") or raw ("raw"). Default "raw". Abbreviations allowed. This option can be set globally using <code>set.cobalt.options()</code> .
s.d.denom	character; how the denominator for standardized mean differences should be calculated, if requested. See <code>col_w_smd()</code> for allowable options. Abbreviations allowed. If not specified, <code>bal.tab()</code> will use "treated" if the estimand of the call to <code>Match()</code> is the ATT, "pooled" if the estimand is the ATE, and "control" if the estimand is the ATC.
thresholds	a named vector of balance thresholds, where the name corresponds to the statistic (i.e., in <code>stats</code>) that the threshold applies to. For example, to request thresholds on mean differences and variance ratios, one can set <code>thresholds = c(m = .05, v = 2)</code> . Requesting a threshold automatically requests the display of that statistic. When specified, extra columns are inserted into the Balance table describing whether the requested balance statistics exceeded the threshold or not. Summary tables tallying the number of variables that exceeded and were within the threshold and displaying the variables with the greatest imbalance on that balance measure are added to the output.
weights	a vector, list, or <code>data.frame</code> containing weights for each unit, or a string containing the names of the weights variables in <code>data</code> , or an object with a <code>get.w()</code> method or a list thereof. The weights can be, e.g., inverse probability weights or matching weights resulting from a matching algorithm.
cluster	either a vector containing cluster membership for each unit or a string containing the name of the cluster membership variable in <code>data</code> or the input object. See <code>class-bal.tab.cluster</code> for details.
imp	either a vector containing imputation indices for each unit or a string containing the name of the imputation index variable in <code>data</code> or the input object. See <code>class-bal.tab.imp</code> for details. Not necessary if <code>data</code> is a <code>mids</code> object.
pairwise	whether balance should be computed for pairs of treatments or for each treatment against all groups combined. See <code>bal.tab.multi()</code> for details. This can also be used with a binary treatment to assess balance with respect to the full sample.
s.weights	Optional; either a vector containing sampling weights for each unit or a string containing the name of the sampling weight variable in <code>data</code> . These function like regular weights except that both the adjusted and unadjusted samples will be weighted according to these weights if weights are used.

abs	logical; whether displayed balance statistics should be in absolute value or not.
subset	a logical or numeric vector denoting whether each observation should be included or which observations should be included. If logical, it should have length equal to the number of units. NAs will be treated as FALSE. This can be used as an alternative to cluster to examine balance on subsets of the data.
quick	logical; if TRUE, will not compute any values that will not be displayed. Set to FALSE if computed values not displayed will be used later.
...	for some input types, other arguments that are required or allowed. Otherwise, further arguments to control display of output. See display options for details.

Details

bal.tab() generates a list of balance summaries for the object given, and functions similarly to [Matching::MatchBalance\(\)](#). The input to bal.tab.Match() must include either both formula and data or both covs and treat. Using the formula + data inputs mirrors how [Matching::MatchBalance\(\)](#) is used.

cobalt functions do not support Match object with sampling weights, i.e., with an argument passed to the weights argument of [Matching::Match\(\)](#).

Value

If clusters and imputations are not specified, an object of class "bal.tab" containing balance summaries for the given object. See [bal.tab\(\)](#) for details.

If clusters are specified, an object of class "bal.tab.cluster" containing balance summaries within each cluster and a summary of balance across clusters. See [class-bal.tab.cluster](#) for details.

See Also

[bal.tab\(\)](#) for details of calculations.

Examples

```
data("lalonge", package = "cobalt")
library(Matching)

## Estimate propensity score
p.fit <- glm(treat ~ age + educ + race +
            married + nodegree + re74 + re75,
            data = lalonge,
            family = "binomial")

Match.out <- Match(Tr = lalonge$treat,
                  X = fitted(p.fit))

## Using formula and data
bal.tab(Match.out, formula = treat ~ age + educ + race +
        married + nodegree + re74 + re75,
        data = lalonge)
```

bal.tab.matchit	<i>Balance Statistics for MatchIt Objects</i>
-----------------	---

Description

Generates balance statistics for matchit objects from **MatchIt**.

Usage

```
## S3 method for class 'matchit'
bal.tab(
  x,
  stats,
  int = FALSE,
  poly = 1,
  distance = NULL,
  add1 = NULL,
  data = NULL,
  continuous,
  binary,
  s.d.denom,
  thresholds = NULL,
  weights = NULL,
  cluster = NULL,
  imp = NULL,
  pairwise = TRUE,
  s.weights = NULL,
  abs = FALSE,
  subset = NULL,
  quick = TRUE,
  method,
  ...
)
```

Arguments

x	a matchit object; the output of a call to <code>MatchIt::matchit()</code> .
stats	character; which statistic(s) should be reported. See <code>stats</code> for allowable options. For binary and multi-category treatments, "mean.diffs" (i.e., mean differences) is the default. For continuous treatments, "correlations" (i.e., treatment-covariate Pearson correlations) is the default. Multiple options are allowed.
int	logical or numeric; whether or not to include 2-way interactions of covariates included in covs and in add1. If numeric, will be passed to poly as well.
poly	numeric; the highest polynomial of each continuous covariate to display. For example, if 2, squares of each continuous covariate will be displayed (in addition

	to the covariate itself); if 3, squares and cubes of each continuous covariate will be displayed, etc. If 1, the default, only the base covariate will be displayed. If <code>int</code> is numeric, <code>poly</code> will take on the value of <code>int</code> .
<code>distance</code>	an optional formula or data frame containing distance values (e.g., propensity scores) or a character vector containing their names. If a formula or variable names are specified, <code>bal.tab()</code> will look in the argument to <code>data</code> , if specified. The distance measure (e.g., propensity score) generated by <code>matchit()</code> is automatically included and named "distance".
<code>addl</code>	an optional formula or data frame containing additional covariates for which to present balance or a character vector containing their names. If a formula or variable names are specified, <code>bal.tab()</code> will look in the arguments to the input object, <code>covs</code> , and <code>data</code> , if specified. For longitudinal treatments, can be a list of allowable arguments, one for each time point.
<code>data</code>	an optional data frame containing variables named in other arguments. For some input object types, this is required.
<code>continuous</code>	whether mean differences for continuous variables should be standardized ("std") or raw ("raw"). Default "std". Abbreviations allowed. This option can be set globally using <code>set.cobalt.options()</code> .
<code>binary</code>	whether mean differences for binary variables (i.e., difference in proportion) should be standardized ("std") or raw ("raw"). Default "raw". Abbreviations allowed. This option can be set globally using <code>set.cobalt.options()</code> .
<code>s.d.denom</code>	character; how the denominator for standardized mean differences should be calculated, if requested. See <code>col_w_smd()</code> for allowable options. Abbreviations allowed. If not specified, <code>bal.tab()</code> will figure out which one is best based on the estimand of the <code>matchit</code> object: if ATT, "treated"; if ATC, "control", otherwise "pooled".
<code>thresholds</code>	a named vector of balance thresholds, where the name corresponds to the statistic (i.e., in <code>stats</code>) that the threshold applies to. For example, to request thresholds on mean differences and variance ratios, one can set <code>thresholds = c(m = .05, v = 2)</code> . Requesting a threshold automatically requests the display of that statistic. When specified, extra columns are inserted into the Balance table describing whether the requested balance statistics exceeded the threshold or not. Summary tables tallying the number of variables that exceeded and were within the threshold and displaying the variables with the greatest imbalance on that balance measure are added to the output.
<code>weights</code>	a vector, list, or <code>data.frame</code> containing weights for each unit, or a string containing the names of the weights variables in <code>data</code> , or an object with a <code>get.w()</code> method or a list thereof. The weights can be, e.g., inverse probability weights or matching weights resulting from a matching algorithm.
<code>cluster</code>	either a vector containing cluster membership for each unit or a string containing the name of the cluster membership variable in <code>data</code> or the input object. See <code>class-bal.tab.cluster</code> for details.
<code>imp</code>	either a vector containing imputation indices for each unit or a string containing the name of the imputation index variable in <code>data</code> or the input object. See <code>class-bal.tab.imp</code> for details. Not necessary if <code>data</code> is a <code>mids</code> object.

<code>pairwise</code>	whether balance should be computed for pairs of treatments or for each treatment against all groups combined. See bal.tab.multi() for details. This can also be used with a binary treatment to assess balance with respect to the full sample.
<code>s.weights</code>	Optional; either a vector containing sampling weights for each unit or a string containing the name of the sampling weight variable in data. These function like regular weights except that both the adjusted and unadjusted samples will be weighted according to these weights if weights are used. If <code>s.weights</code> was specified in the call to <code>matchit()</code> , they will automatically be included and do not need be specified again (though there is no harm if they are).
<code>abs</code>	logical; whether displayed balance statistics should be in absolute value or not.
<code>subset</code>	a logical or numeric vector denoting whether each observation should be included or which observations should be included. If logical, it should have length equal to the number of units. NAs will be treated as FALSE. This can be used as an alternative to <code>cluster</code> to examine balance on subsets of the data.
<code>quick</code>	logical; if TRUE, will not compute any values that will not be displayed. Set to FALSE if computed values not displayed will be used later.
<code>method</code>	a character vector containing the method of adjustment. Ignored unless subclassification was used in the original call to <code>matchit()</code> . If "weighting", the subclassification weights will be used and subclasses will be ignored. If "subclassification", balance will be assessed using the subclasses (see class-bal.tab.subclass for details). Abbreviations allowed.
<code>...</code>	for some input types, other arguments that are required or allowed. Otherwise, further arguments to control display of output. See display options for details.

Details

`bal.tab.matchit()` generates a list of balance summaries for the `matchit` object given, and functions similarly to [MatchIt::summary.matchit\(\)](#). `bal.tab()` behaves differently depending on whether subclasses are used in conditioning or not. If they are used, `bal.tab()` creates balance statistics for each subclass and for the sample in aggregate; see [class-bal.tab.subclass](#) for more information.

Value

If subclassification is used and `method` is set to "subclassification", an object of class "`bal.tab.subclass`" containing balance summaries within and across subclasses. See [class-bal.tab.subclass](#) for details.

If matching is used and clusters are not specified, an object of class "`bal.tab`" containing balance summaries for the `matchit` object. See [bal.tab\(\)](#) for details.

If clusters are specified, an object of class "`bal.tab.cluster`" containing balance summaries within each cluster and a summary of balance across clusters. See [class-bal.tab.cluster](#) for details.

See Also

- [bal.tab\(\)](#) for details of calculations.

Examples

```

library(MatchIt)
data("lalonge", package = "cobalt")

## Nearest Neighbor matching
m.out1 <- matchit(treat ~ age + educ + race +
                  married + nodegree + re74 + re75,
                  data = lalonge,
                  method = "nearest")

bal.tab(m.out1, un = TRUE,
        thresholds = c(m = .1, v = 2))

## Subclassification
m.out2 <- matchit(treat ~ age + educ + race +
                  married + nodegree + re74 + re75,
                  data = lalonge,
                  method = "subclass")

bal.tab(m.out2, disp.subclass = TRUE)

```

bal.tab.mimids

Balance Statistics for MatchThem Objects

Description

Generates balance statistics for mimids and wimids objects from **MatchThem**.

Usage

```

## S3 method for class 'mimids'
bal.tab(
  x,
  stats,
  int = FALSE,
  poly = 1,
  distance = NULL,
  add1 = NULL,
  data = NULL,
  continuous,
  binary,
  s.d.denom,
  thresholds = NULL,
  weights = NULL,
  cluster = NULL,
  pairwise = TRUE,
  s.weights = NULL,

```

```

abs = FALSE,
subset = NULL,
quick = TRUE,
...
)

```

Arguments

x	a mimids or wimids object; the output of a call to <code>MatchThem::matchthem()</code> or <code>MatchThem::weightthem()</code> .
stats	character; which statistic(s) should be reported. See <code>stats</code> for allowable options. For binary and multi-category treatments, "mean.diffs" (i.e., mean differences) is the default. For continuous treatments, "correlations" (i.e., treatment-covariate Pearson correlations) is the default. Multiple options are allowed.
int	logical or numeric; whether or not to include 2-way interactions of covariates included in covs and in add1. If numeric, will be passed to poly as well.
poly	numeric; the highest polynomial of each continuous covariate to display. For example, if 2, squares of each continuous covariate will be displayed (in addition to the covariate itself); if 3, squares and cubes of each continuous covariate will be displayed, etc. If 1, the default, only the base covariate will be displayed. If int is numeric, poly will take on the value of int.
distance	an optional formula or data frame containing distance values (e.g., propensity scores) or a character vector containing their names. If a formula or variable names are specified, <code>bal.tab()</code> will look in the argument to data, if specified. The distance measure generated by <code>matchthem()</code> or <code>weightthem()</code> is automatically included and named "distance" or "prop.score", respectively.
add1	an optional formula or data frame containing additional covariates for which to present balance or a character vector containing their names. If a formula or variable names are specified, <code>bal.tab()</code> will look in the arguments to the input object, covs, and data, if specified. For longitudinal treatments, can be a list of allowable arguments, one for each time point.
data	an optional data frame containing variables named in other arguments. For some input object types, this is required.
continuous	whether mean differences for continuous variables should be standardized ("std") or raw ("raw"). Default "std". Abbreviations allowed. This option can be set globally using <code>set.cobalt.options()</code> .
binary	whether mean differences for binary variables (i.e., difference in proportion) should be standardized ("std") or raw ("raw"). Default "raw". Abbreviations allowed. This option can be set globally using <code>set.cobalt.options()</code> .
s.d.denom	character; how the denominator for standardized mean differences should be calculated, if requested. See <code>col_w_smd()</code> for allowable options. Abbreviations allowed. If not specified, the defaults depend on the options specified in the original function calls; see <code>bal.tab.matchit()</code> and <code>bal.tab.weightit()</code> for details on the defaults.

thresholds	a named vector of balance thresholds, where the name corresponds to the statistic (i.e., in stats) that the threshold applies to. For example, to request thresholds on mean differences and variance ratios, one can set <code>thresholds = c(m = .05, v = 2)</code> . Requesting a threshold automatically requests the display of that statistic. When specified, extra columns are inserted into the Balance table describing whether the requested balance statistics exceeded the threshold or not. Summary tables tallying the number of variables that exceeded and were within the threshold and displaying the variables with the greatest imbalance on that balance measure are added to the output.
weights	a vector, list, or <code>data.frame</code> containing weights for each unit, or a string containing the names of the weights variables in data, or an object with a <code>get.w()</code> method or a list thereof. The weights can be, e.g., inverse probability weights or matching weights resulting from a matching algorithm.
cluster	either a vector containing cluster membership for each unit or a string containing the name of the cluster membership variable in data or the input object. See class-bal.tab.cluster for details.
pairwise	whether balance should be computed for pairs of treatments or for each treatment against all groups combined. See bal.tab.multi() for details. This can also be used with a binary treatment to assess balance with respect to the full sample.
s.weights	Optional; either a vector containing sampling weights for each unit or a string containing the name of the sampling weight variable in data. These function like regular weights except that both the adjusted and unadjusted samples will be weighted according to these weights if weights are used.
abs	logical; whether displayed balance statistics should be in absolute value or not.
subset	a logical or numeric vector denoting whether each observation should be included or which observations should be included. If logical, it should have length equal to the number of units. NAs will be treated as FALSE. This can be used as an alternative to cluster to examine balance on subsets of the data.
quick	logical; if TRUE, will not compute any values that will not be displayed. Set to FALSE if computed values not displayed will be used later.
...	for some input types, other arguments that are required or allowed. Otherwise, further arguments to control display of output. See display options for details.

Details

`bal.tab.mimids()` and `bal.tab.wimids()` generate a list of balance summaries for the `mimids` or `wimids` object given.

Value

If clusters are not specified, an object of class `"bal.tab.imp"` containing balance summaries for each imputation and a summary of balance across imputations. See [class-bal.tab.imp](#) for details.

If clusters are specified, an object of class `"bal.tab.imp.cluster"` containing summaries between and across all clusters and imputations.

See Also

- [bal.tab\(\)](#) for details of calculations
- [bal.tab.matchit\(\)](#) and [bal.tab.weightit\(\)](#)

Examples

```
library(MatchThem)

data("lalonge_mis", package = "cobalt")

# Imputing the missing data
imp <- mice::mice(lalonge_mis, m = 5,
  print = FALSE)

# Matching using within-imputation propensity scores
mt.out1 <- matchthem(treat ~ age + educ + race +
  married + nodegree + re74 + re75,
  data = imp, approach = "within")
bal.tab(mt.out1)

# Matching using across-imputation average propensity scores
mt.out2 <- matchthem(treat ~ age + educ + race +
  married + nodegree + re74 + re75,
  data = imp, approach = "across")

bal.tab(mt.out2)

# Weighting using within-imputation propensity scores
wt.out <- weightthem(treat ~ age + educ + race +
  married + nodegree + re74 + re75,
  data = imp, approach = "within",
  estimand = "ATT")

bal.tab(wt.out)
```

bal.tab.optmatch

Balance Statistics for optmatch Objects

Description

Generates balance statistics for output objects from **optmatch**.

Usage

```
## S3 method for class 'optmatch'
bal.tab(
  x,
  formula = NULL,
```

```

    data = NULL,
    treat = NULL,
    covs = NULL,
    estimand = NULL,
    stats,
    int = FALSE,
    poly = 1,
    distance = NULL,
    add1 = NULL,
    continuous,
    binary,
    s.d.denom,
    thresholds = NULL,
    weights = NULL,
    cluster = NULL,
    imp = NULL,
    pairwise = TRUE,
    s.weights = NULL,
    abs = FALSE,
    subset = NULL,
    quick = TRUE,
    ...
  )

```

Arguments

<code>x</code>	an <code>optmatch</code> object (the output of a call to <code>optmatch::pairmatch()</code> or <code>optmatch::fullmatch()</code>).
<code>formula</code>	a formula with the treatment variable as the response and the covariates for which balance is to be assessed as the predictors. All named variables must be in data. See Details.
<code>data</code>	a data frame containing variables named in formula, if supplied, and other arguments.
<code>treat</code>	a vector of treatment statuses. See Details.
<code>covs</code>	a data frame of covariate values for which to check balance. See Details.
<code>estimand</code>	character; whether the desired estimand is the "ATT", "ATC", or "ATE". Default is "ATT".
<code>stats</code>	character; which statistic(s) should be reported. See <code>stats</code> for allowable options. For binary and multi-category treatments, "mean.diffs" (i.e., mean differences) is the default. For continuous treatments, "correlations" (i.e., treatment-covariate Pearson correlations) is the default. Multiple options are allowed.
<code>int</code>	logical or numeric; whether or not to include 2-way interactions of covariates included in covs and in add1. If numeric, will be passed to poly as well.
<code>poly</code>	numeric; the highest polynomial of each continuous covariate to display. For example, if 2, squares of each continuous covariate will be displayed (in addition to the covariate itself); if 3, squares and cubes of each continuous covariate will

	be displayed, etc. If 1, the default, only the base covariate will be displayed. If <code>int</code> is numeric, <code>poly</code> will take on the value of <code>int</code> .
<code>distance</code>	an optional formula or data frame containing distance values (e.g., propensity scores) or a character vector containing their names. If a formula or variable names are specified, <code>bal.tab()</code> will look in the argument to <code>data</code> , if specified. For longitudinal treatments, can be a list of allowable arguments, one for each time point.
<code>add1</code>	an optional formula or data frame containing additional covariates for which to present balance or a character vector containing their names. If a formula or variable names are specified, <code>bal.tab()</code> will look in the arguments to the input object, <code>covs</code> , and <code>data</code> , if specified. For longitudinal treatments, can be a list of allowable arguments, one for each time point.
<code>continuous</code>	whether mean differences for continuous variables should be standardized (" <code>std</code> ") or raw (" <code>raw</code> "). Default " <code>std</code> ". Abbreviations allowed. This option can be set globally using <code>set.cobalt.options()</code> .
<code>binary</code>	whether mean differences for binary variables (i.e., difference in proportion) should be standardized (" <code>std</code> ") or raw (" <code>raw</code> "). Default " <code>raw</code> ". Abbreviations allowed. This option can be set globally using <code>set.cobalt.options()</code> .
<code>s.d.denom</code>	character; how the denominator for standardized mean differences should be calculated, if requested. See <code>col_w_smd()</code> for allowable options. Abbreviations allowed. If unspecified, <code>bal.tab()</code> will figure out which one is best based on the estimand, if given (for ATT, " <code>treated</code> "; for ATC, " <code>control</code> "; otherwise " <code>pooled</code> ") and other clues if not.
<code>thresholds</code>	a named vector of balance thresholds, where the name corresponds to the statistic (i.e., in <code>stats</code>) that the threshold applies to. For example, to request thresholds on mean differences and variance ratios, one can set <code>thresholds = c(m = .05, v = 2)</code> . Requesting a threshold automatically requests the display of that statistic. When specified, extra columns are inserted into the Balance table describing whether the requested balance statistics exceeded the threshold or not. Summary tables tallying the number of variables that exceeded and were within the threshold and displaying the variables with the greatest imbalance on that balance measure are added to the output.
<code>weights</code>	a vector, list, or <code>data.frame</code> containing weights for each unit, or a string containing the names of the weights variables in <code>data</code> , or an object with a <code>get.w()</code> method or a list thereof. The weights can be, e.g., inverse probability weights or matching weights resulting from a matching algorithm.
<code>cluster</code>	either a vector containing cluster membership for each unit or a string containing the name of the cluster membership variable in <code>data</code> or the input object. See <code>class-bal.tab.cluster</code> for details.
<code>imp</code>	either a vector containing imputation indices for each unit or a string containing the name of the imputation index variable in <code>data</code> or the input object. See <code>class-bal.tab.imp</code> for details. Not necessary if <code>data</code> is a <code>mids</code> object.
<code>pairwise</code>	whether balance should be computed for pairs of treatments or for each treatment against all groups combined. See <code>bal.tab.multi()</code> for details. This can also be used with a binary treatment to assess balance with respect to the full sample.

s.weights	Optional; either a vector containing sampling weights for each unit or a string containing the name of the sampling weight variable in data. These function like regular weights except that both the adjusted and unadjusted samples will be weighted according to these weights if weights are used.
abs	logical; whether displayed balance statistics should be in absolute value or not.
subset	a logical or numeric vector denoting whether each observation should be included or which observations should be included. If logical, it should have length equal to the number of units. NAs will be treated as FALSE. This can be used as an alternative to cluster to examine balance on subsets of the data.
quick	logical; if TRUE, will not compute any values that will not be displayed. Set to FALSE if computed values not displayed will be used later.
...	for some input types, other arguments that are required or allowed. Otherwise, further arguments to control display of output. See display options for details.

Details

`bal.tab()` generates a list of balance summaries for the object given. The input to `bal.tab.optmatch()` must include either both formula and data or just covs (treat is not necessary).

Value

If clusters and imputations are not specified, an object of class "bal.tab" containing balance summaries for the given object. See [bal.tab\(\)](#) for details.

If clusters are specified, an object of class "bal.tab.cluster" containing balance summaries within each cluster and a summary of balance across clusters. See [class-bal.tab.cluster](#) for details.

See Also

[bal.tab\(\)](#) for details of calculations.

Examples

```
data("lalonge", package = "cobalt")

lalonge$prop.score <- glm(treat ~ age + educ + race +
                        married + nodegree + re74 + re75,
                        data = lalonge, family = binomial)$fitted.values
pm <- optmatch::pairmatch(treat ~ prop.score, data = lalonge)

## Using formula and data; LHS of formula not required
bal.tab(pm, formula = ~ age + educ + race +
        married + nodegree + re74 + re75,
        data = lalonge)

## Using covs
covs <- subset(lalonge, select = -c(re78, treat))
bal.tab(pm, covs = covs)
```

bal.tab.ps

*Balance Statistics for twang Objects***Description**

Generates balance statistics for ps, mnps, and iptw objects from **twang** and for ps.cont objects from **twangContinuous**.

Usage

```
## S3 method for class 'ps'
bal.tab(
  x,
  stop.method,
  stats,
  int = FALSE,
  poly = 1,
  distance = NULL,
  add1 = NULL,
  data = NULL,
  continuous,
  binary,
  s.d.denom,
  thresholds = NULL,
  weights = NULL,
  cluster = NULL,
  imp = NULL,
  pairwise = TRUE,
  s.weights = NULL,
  abs = FALSE,
  subset = NULL,
  quick = TRUE,
  ...
)
```

Arguments

x	a ps, mnps, iptw, or ps.cont object; the output of a call to <code>twang::ps()</code> , <code>twang::mnps()</code> , <code>twang::iptw()</code> or <code>twangContinuous::ps.cont()</code> .
stop.method	a string containing the names of the stopping methods used in the original call to <code>ps()</code> , <code>mnps()</code> , or <code>iptw()</code> . Examples include "es.max" or "ks.mean" for ps and mnps objects. <code>bal.tab()</code> will assess balance for the weights created by those stopping methods. The names can be abbreviated as long as the abbreviations are specific enough. If no stopping methods are provided, <code>bal.tab()</code> will default to displaying balance for all available stopping methods. Ignored for ps.cont objects.

stats	character; which statistic(s) should be reported. See stats for allowable options. For binary and multi-category treatments, "mean.diffs" (i.e., mean differences) is the default. For continuous treatments, "correlations" (i.e., treatment-covariate Pearson correlations) is the default. Multiple options are allowed.
int	logical or numeric; whether or not to include 2-way interactions of covariates included in covs and in add1. If numeric, will be passed to poly as well.
poly	numeric; the highest polynomial of each continuous covariate to display. For example, if 2, squares of each continuous covariate will be displayed (in addition to the covariate itself); if 3, squares and cubes of each continuous covariate will be displayed, etc. If 1, the default, only the base covariate will be displayed. If int is numeric, poly will take on the value of int.
distance	an optional formula or data frame containing distance values (e.g., propensity scores) or a character vector containing their names. If a formula or variable names are specified, bal.tab() will look in the argument to data, if specified. The propensity scores generated by ps() and iptw() (but not mnps() or ps.cont()) are automatically included and named "prop.score.{stop.method}".
add1	an optional formula or data frame containing additional covariates for which to present balance or a character vector containing their names. If a formula or variable names are specified, bal.tab() will look in the arguments to the input object, covs, and data, if specified. For longitudinal treatments, can be a list of allowable arguments, one for each time point.
data	an optional data frame containing variables named in other arguments. For some input object types, this is required.
continuous	whether mean differences for continuous variables should be standardized ("std") or raw ("raw"). Default "std". Abbreviations allowed. This option can be set globally using set.cobalt.options() .
binary	whether mean differences for binary variables (i.e., difference in proportion) should be standardized ("std") or raw ("raw"). Default "raw". Abbreviations allowed. This option can be set globally using set.cobalt.options() .
s.d.denom	character; how the denominator for standardized mean differences should be calculated, if requested. See col_w_smd() for allowable options. Abbreviations allowed. If not specified, for ps objects, bal.tab() will use "treated" if the estimand of the call to ps() is the ATT and "pooled" if the estimand is the ATE; for mnps objects, bal.tab() will use "treated" if treatATT was specified in the original call to mnps and "pooled" otherwise. Use "all" to get the same values computed by bal.table() in twang .
thresholds	a named vector of balance thresholds, where the name corresponds to the statistic (i.e., in stats) that the threshold applies to. For example, to request thresholds on mean differences and variance ratios, one can set thresholds = c(m = .05, v = 2). Requesting a threshold automatically requests the display of that statistic. When specified, extra columns are inserted into the Balance table describing whether the requested balance statistics exceeded the threshold or not. Summary tables tallying the number of variables that exceeded and were within the threshold and displaying the variables with the greatest imbalance on that balance measure are added to the output.

weights	a vector, list, or data.frame containing weights for each unit, or a string containing the names of the weights variables in data, or an object with a <code>get.w()</code> method or a list thereof. The weights can be, e.g., inverse probability weights or matching weights resulting from a matching algorithm.
cluster	either a vector containing cluster membership for each unit or a string containing the name of the cluster membership variable in data or the input object. See class-bal.tab.cluster for details.
imp	either a vector containing imputation indices for each unit or a string containing the name of the imputation index variable in data or the input object. See class-bal.tab.imp for details. Not necessary if data is a mids object.
pairwise	whether balance should be computed for pairs of treatments or for each treatment against all groups combined. See bal.tab.multi() for details. This can also be used with a binary treatment to assess balance with respect to the full sample.
s.weights	Optional; either a vector containing sampling weights for each unit or a string containing the name of the sampling weight variable in data. These function like regular weights except that both the adjusted and unadjusted samples will be weighted according to these weights if weights are used. If sampw was supplied in the call to <code>ps()</code> , <code>mnpes()</code> , <code>iptw()</code> , or <code>ps.cont()</code> , they will automatically be supplied to <code>s.weights</code> and do not need be specified again (though there is no harm if they are).
abs	logical; whether displayed balance statistics should be in absolute value or not.
subset	a logical or numeric vector denoting whether each observation should be included or which observations should be included. If logical, it should have length equal to the number of units. NAs will be treated as FALSE. This can be used as an alternative to <code>cluster</code> to examine balance on subsets of the data.
quick	logical; if TRUE, will not compute any values that will not be displayed. Set to FALSE if computed values not displayed will be used later.
...	for some input types, other arguments that are required or allowed. Otherwise, further arguments to control display of output. See display options for details.

Details

`bal.tab.ps()` generates a list of balance summaries for the input object given, and functions similarly to [twang::bal.table\(\)](#). The variances used in the denominator of the standardized mean differences computed in `twang::bal.table()` are weighted and computed using `survey::svyvar()` and are unweighted here (except when `s.weights` are specified, in which case [col_w_sd\(\)](#) is used). **twang** also uses "all" as the default `s.d.denom` when the estimand is the ATE; the default here is "pooled". For these reasons, results may differ slightly between the two packages.

Value

For binary or continuous point treatments, if clusters are not specified, an object of class "bal.tab" containing balance summaries for the `ps` object. See [bal.tab\(\)](#) for details.

If clusters are specified, an object of class "bal.tab.cluster" containing balance summaries within each cluster and a summary of balance across clusters. See [class-bal.tab.cluster](#) for details.

If `mnpes()` is used with multi-category treatments, an object of class `"bal.tab.multi"` containing balance summaries for each pairwise treatment comparison and a summary of balance across pairwise comparisons. See `bal.tab.multi()` for details.

See Also

- `bal.tab()` for details of calculations.
- `class-bal.tab.cluster` for more information on clustered data.
- `bal.tab.multi()` for more information on multi-category treatments.
- `class-bal.tab.msm` for more information on longitudinal treatments.

Examples

```
library(twang)
data("lalonge", package = "cobalt")

## Using ps() for generalized boosted modeling
ps.out <- ps(treat ~ age + educ + married + race +
             nodegree + re74 + re75, data = lalonge,
             stop.method = c("ks.mean", "es.mean"),
             estimand = "ATT", verbose = FALSE)

bal.tab(ps.out, stop.method = "ks.mean", un = TRUE,
        stats = c("m", "ks"),
        thresholds = c(m = .1))
```

bal.tab.sbwcau	<i>Balance Statistics for sbw Objects</i>
----------------	---

Description

Generates balance statistics for `sbwcau` objects from **sbw**.

Usage

```
## S3 method for class 'sbwcau'
bal.tab(
  x,
  stats,
  int = FALSE,
  poly = 1,
  distance = NULL,
  add1 = NULL,
  data = NULL,
  continuous,
  binary,
```

```

s.d.denom,
thresholds = NULL,
weights = NULL,
cluster = NULL,
imp = NULL,
pairwise = TRUE,
s.weights = NULL,
abs = FALSE,
subset = NULL,
quick = TRUE,
...
)

```

Arguments

x	an sbwcau object; the output of a call to <code>sbw::sbw()</code> .
stats	character; which statistic(s) should be reported. See <code>stats</code> for allowable options. For binary and multi-category treatments, "mean.diffs" (i.e., mean differences) is the default. For continuous treatments, "correlations" (i.e., treatment-covariate Pearson correlations) is the default. Multiple options are allowed.
int	logical or numeric; whether or not to include 2-way interactions of covariates included in covs and in add1. If numeric, will be passed to poly as well.
poly	numeric; the highest polynomial of each continuous covariate to display. For example, if 2, squares of each continuous covariate will be displayed (in addition to the covariate itself); if 3, squares and cubes of each continuous covariate will be displayed, etc. If 1, the default, only the base covariate will be displayed. If int is numeric, poly will take on the value of int.
distance	an optional formula or data frame containing distance values (e.g., propensity scores) or a character vector containing their names. If a formula or variable names are specified, <code>bal.tab()</code> will look in the argument to data, if specified. For longitudinal treatments, can be a list of allowable arguments, one for each time point.
add1	an optional formula or data frame containing additional covariates for which to present balance or a character vector containing their names. If a formula or variable names are specified, <code>bal.tab()</code> will look in the arguments to the input object, covs, and data, if specified. For longitudinal treatments, can be a list of allowable arguments, one for each time point.
data	an optional data frame containing variables named in other arguments. For some input object types, this is required.
continuous	whether mean differences for continuous variables should be standardized ("std") or raw ("raw"). Default "std". Abbreviations allowed. This option can be set globally using <code>set.cobalt.options()</code> .
binary	whether mean differences for binary variables (i.e., difference in proportion) should be standardized ("std") or raw ("raw"). Default "raw". Abbreviations allowed. This option can be set globally using <code>set.cobalt.options()</code> .

s.d.denom	character; how the denominator for standardized mean differences should be calculated, if requested. See <code>col_w_smd()</code> for allowable options. Abbreviations allowed. If not specified, <code>bal.tab()</code> will figure out which one is best based on the par component of the sbwcau object: if "att", "treated"; if "atc", "control"; otherwise "pooled".
thresholds	a named vector of balance thresholds, where the name corresponds to the statistic (i.e., in stats) that the threshold applies to. For example, to request thresholds on mean differences and variance ratios, one can set <code>thresholds = c(m = .05, v = 2)</code> . Requesting a threshold automatically requests the display of that statistic. When specified, extra columns are inserted into the Balance table describing whether the requested balance statistics exceeded the threshold or not. Summary tables tallying the number of variables that exceeded and were within the threshold and displaying the variables with the greatest imbalance on that balance measure are added to the output.
weights	a vector, list, or data.frame containing weights for each unit, or a string containing the names of the weights variables in data, or an object with a <code>get.w()</code> method or a list thereof. The weights can be, e.g., inverse probability weights or matching weights resulting from a matching algorithm.
cluster	either a vector containing cluster membership for each unit or a string containing the name of the cluster membership variable in data or the input object. See <code>class-bal.tab.cluster</code> for details.
imp	either a vector containing imputation indices for each unit or a string containing the name of the imputation index variable in data or the input object. See <code>class-bal.tab.imp</code> for details. Not necessary if data is a mids object.
pairwise	whether balance should be computed for pairs of treatments or for each treatment against all groups combined. See <code>bal.tab.multi()</code> for details. This can also be used with a binary treatment to assess balance with respect to the full sample.
s.weights	Optional; either a vector containing sampling weights for each unit or a string containing the name of the sampling weight variable in data. These function like regular weights except that both the adjusted and unadjusted samples will be weighted according to these weights if weights are used.
abs	logical; whether displayed balance statistics should be in absolute value or not.
subset	a logical or numeric vector denoting whether each observation should be included or which observations should be included. If logical, it should have length equal to the number of units. NAs will be treated as FALSE. This can be used as an alternative to cluster to examine balance on subsets of the data.
quick	logical; if TRUE, will not compute any values that will not be displayed. Set to FALSE if computed values not displayed will be used later.
...	for some input types, other arguments that are required or allowed. Otherwise, further arguments to control display of output. See <code>display options</code> for details.

Details

`bal.tab.sbwcau()` generates a list of balance summaries for the sbwcau object given, and functions similarly to `sbw::summarize()`.

Value

If clusters are not specified, an object of class "bal.tab" containing balance summaries for the sbwcau object. See [bal.tab\(\)](#) for details.

If clusters are specified, an object of class "bal.tab.cluster" containing balance summaries within each cluster and a summary of balance across clusters. See [class-bal.tab.cluster](#) for details.

See Also

- [bal.tab\(\)](#) for details of calculations.

Examples

```
library(sbw)
data("lalonge", package = "cobalt")

## Stable balancing weights for the ATT
sbw.out <- sbw(splitfactor(lalonge, drop.first = "if2"),
               ind = "treat",
               bal = list(bal_cov = c("age", "educ", "race_black",
                                     "race_hispan", "race_white",
                                     "married", "nodegree",
                                     "re74", "re75"),
                           bal_alg = FALSE,
                           bal_tol = .001),
               par = list(par_est = "att"))

bal.tab(sbw.out, un = TRUE, poly = 2)
```

bal.tab.time.list

Balance Statistics for Longitudinal Datasets

Description

Generates balance statistics for data coming from a longitudinal treatment scenario. The primary input is in the form of a list of formulas or data.frames contain the covariates at each time point. `bal.tab()` automatically classifies this list as either a `data.frame.list` or `formula.list`, respectively.

Usage

```
## S3 method for class 'formula.list'
bal.tab(
  x,
  stats,
  int = FALSE,
  poly = 1,
```

```

        distance = NULL,
        addl = NULL,
        data = NULL,
        continuous,
        binary,
        s.d.denom,
        thresholds = NULL,
        weights = NULL,
        cluster = NULL,
        imp = NULL,
        pairwise = TRUE,
        s.weights = NULL,
        abs = FALSE,
        subset = NULL,
        quick = TRUE,
        ...
    )

## S3 method for class 'data.frame.list'
bal.tab(
  x,
  treat.list,
  stats,
  int = FALSE,
  poly = 1,
  distance = NULL,
  addl = NULL,
  data = NULL,
  continuous,
  binary,
  s.d.denom,
  thresholds = NULL,
  weights = NULL,
  cluster = NULL,
  imp = NULL,
  pairwise = TRUE,
  s.weights = NULL,
  abs = FALSE,
  subset = NULL,
  quick = TRUE,
  ...
)

```

Arguments

x either a list of data frames containing all the covariates to be assessed at each time point or a list of formulas with the treatment for each time period on the left and the covariates for which balance is to be displayed on the right. Covariates to be assessed at multiple points must be included in the entries for each time

point. Data must be in the "wide" format, with one row per unit. If a formula list is supplied, an argument to data is required unless all objects in the formulas exist in the environment.

stats	character; which statistic(s) should be reported. See stats for allowable options. For binary and multi-category treatments, "mean.diffs" (i.e., mean differences) is the default. For continuous treatments, "correlations" (i.e., treatment-covariate Pearson correlations) is the default. Multiple options are allowed.
int	logical or numeric; whether or not to include 2-way interactions of covariates included in covs and in add1. If numeric, will be passed to poly as well.
poly	numeric; the highest polynomial of each continuous covariate to display. For example, if 2, squares of each continuous covariate will be displayed (in addition to the covariate itself); if 3, squares and cubes of each continuous covariate will be displayed, etc. If 1, the default, only the base covariate will be displayed. If int is numeric, poly will take on the value of int.
distance	an optional formula or data frame containing distance values (e.g., propensity scores) or a character vector containing their names. If a formula or variable names are specified, <code>bal.tab()</code> will look in the argument to data, if specified. For longitudinal treatments, can be a list of allowable arguments, one for each time point.
add1	an optional formula or data frame containing additional covariates for which to present balance or a character vector containing their names. If a formula or variable names are specified, <code>bal.tab()</code> will look in the arguments to the input object, covs, and data, if specified. For longitudinal treatments, can be a list of allowable arguments, one for each time point.
data	an optional data frame containing variables named in other arguments. For some input object types, this is required.
continuous	whether mean differences for continuous variables should be standardized ("std") or raw ("raw"). Default "std". Abbreviations allowed. This option can be set globally using set.cobalt.options() .
binary	whether mean differences for binary variables (i.e., difference in proportion) should be standardized ("std") or raw ("raw"). Default "raw". Abbreviations allowed. This option can be set globally using set.cobalt.options() .
s.d.denom	character; how the denominator for standardized mean differences should be calculated, if requested. See col_w_smd() for allowable options. Abbreviations allowed. It is recommended not to set this argument for longitudinal treatments.
thresholds	a named vector of balance thresholds, where the name corresponds to the statistic (i.e., in stats) that the threshold applies to. For example, to request thresholds on mean differences and variance ratios, one can set <code>thresholds = c(m = .05, v = 2)</code> . Requesting a threshold automatically requests the display of that statistic. When specified, extra columns are inserted into the Balance table describing whether the requested balance statistics exceeded the threshold or not. Summary tables tallying the number of variables that exceeded and were within the threshold and displaying the variables with the greatest imbalance on that balance measure are added to the output.

weights	a vector, list, or data.frame containing weights for each unit, or a string containing the names of the weights variables in data, or an object with a <code>get.w()</code> method or a list thereof. The weights can be, e.g., inverse probability weights or matching weights resulting from a matching algorithm.
cluster	either a vector containing cluster membership for each unit or a string containing the name of the cluster membership variable in data or the input object. See class-bal.tab.cluster for details.
imp	either a vector containing imputation indices for each unit or a string containing the name of the imputation index variable in data or the input object. See class-bal.tab.imp for details. Not necessary if data is a mids object.
pairwise	whether balance should be computed for pairs of treatments or for each treatment against all groups combined. See bal.tab.multi() for details. This can also be used with a binary treatment to assess balance with respect to the full sample.
s.weights	Optional; either a vector containing sampling weights for each unit or a string containing the name of the sampling weight variable in data. These function like regular weights except that both the adjusted and unadjusted samples will be weighted according to these weights if weights are used.
abs	logical; whether displayed balance statistics should be in absolute value or not.
subset	a logical or numeric vector denoting whether each observation should be included or which observations should be included. If logical, it should have length equal to the number of units. NAs will be treated as FALSE. This can be used as an alternative to cluster to examine balance on subsets of the data.
quick	logical; if TRUE, will not compute any values that will not be displayed. Set to FALSE if computed values not displayed will be used later.
...	for some input types, other arguments that are required or allowed. Otherwise, further arguments to control display of output. See display options for details.
treat.list	treatment status for each unit at each time point. This can be specified as a list or data frame of vectors, each of which contains the treatment status of each individual at each time point, or a list or vector of the names of variables in data that contain treatment at each time point. Required for the <code>data.frame.list</code> method.

Details

`bal.tab.formula.list()` and `bal.tab.data.frame.list()` generate a list of balance summaries for each time point based on the treatments and covariates provided. All data must be in the "wide" format, with exactly one row per unit and columns representing variables at different time points. See the [WeightIt::weightitMSM\(\)](#) documentation for an example of how to transform long data into wide data using [reshape\(\)](#).

Multiple sets of weights can be supplied simultaneously by including entering a data frame or a character vector containing the names of weight variables found in data or a list thereof. When only one set of weights is supplied, the output for the adjusted group will simply be called "Adj", but otherwise will be named after each corresponding set of weights. Specifying multiple sets of weights will also add components to other outputs of `bal.tab()`.

Value

An object of class `bal.tab.msm` containing balance summaries at each time point. Each balance summary is its own `bal.tab` object. See [class-bal.tab.msm](#) for more details.

See [bal.tab\(\)](#) [base methods\(\)](#) for more detailed information on the value of the `bal.tab` objects produced for each time point.

See Also

- [bal.tab\(\)](#) for details of calculations.
- [class-bal.tab.msm](#) for output and related options.
- [class-bal.tab.cluster](#) for more information on clustered data.
- [class-bal.tab.imp](#) for more information on multiply imputed data.
- [bal.tab.multi\(\)](#) for more information on multi-category treatments.

Examples

```
data("msmdata", package = "WeightIt")

## Estimating longitudinal propensity scores and weights
ps1 <- glm(A_1 ~ X1_0 + X2_0,
           data = msmdata,
           family = "binomial")$fitted.values
w1 <- ifelse(msmdata$A_1 == 1, 1 / ps1, 1 / (1 - ps1))

ps2 <- glm(A_2 ~ X1_1 + X2_1 +
           A_1 + X1_0 + X2_0,
           data = msmdata,
           family = "binomial")$fitted.values
w2 <- ifelse(msmdata$A_2 == 1, 1 / ps2, 1 / (1 - ps2))

ps3 <- glm(A_3 ~ X1_2 + X2_2 +
           A_2 + X1_1 + X2_1 +
           A_1 + X1_0 + X2_0,
           data = msmdata,
           family = "binomial")$fitted.values
w3 <- ifelse(msmdata$A_3 == 1, 1 / ps3, 1 / (1 - ps3))

w <- w1 * w2 * w3

# Formula interface plus add1:
bal.tab(list(A_1 ~ X1_0 + X2_0,
            A_2 ~ X1_1 + X2_1 +
              A_1 + X1_0 + X2_0,
            A_3 ~ X1_2 + X2_2 +
              A_2 + X1_1 + X2_1 +
              A_1 + X1_0 + X2_0),
         data = msmdata,
         weights = w,
         distance = list(~ps1, ~ps2, ~ps3),
         add1 = ~X1_0 * X2_0,
```

```

        un = TRUE)

# data frame interface:
bal.tab(list(msmdata[c("X1_0", "X2_0")],
                    msmdata[c("X1_1", "X2_1", "A_1", "X1_0", "X2_0")],
                    msmdata[c("X1_2", "X2_2", "A_2", "X1_1", "X2_1",
                              "A_1", "X1_0", "X2_0")]),
        treat.list = msmdata[c("A_1", "A_2", "A_3")],
        weights = w,
        distance = list(~ps1, ~ps2, ~ps3),
        un = TRUE)

```

bal.tab.weightit	<i>Balance Statistics for WeightIt Objects</i>
------------------	--

Description

Generates balance statistics for `weightit` and `weightitMSM` objects from **WeightIt**.

Usage

```

## S3 method for class 'weightit'
bal.tab(
  x,
  stats,
  int = FALSE,
  poly = 1,
  distance = NULL,
  add1 = NULL,
  data = NULL,
  continuous,
  binary,
  s.d.denom,
  thresholds = NULL,
  weights = NULL,
  cluster = NULL,
  imp = NULL,
  pairwise = TRUE,
  s.weights = NULL,
  abs = FALSE,
  subset = NULL,
  quick = TRUE,
  ...
)

```

Arguments

<code>x</code>	a <code>weightit</code> or <code>weightitMSM</code> object; the output of a call to <code>WeightIt::weightit()</code> or <code>WeightIt::weightitMSM()</code> .
<code>stats</code>	character; which statistic(s) should be reported. See <code>stats</code> for allowable options. For binary and multi-category treatments, "mean.diffs" (i.e., mean differences) is the default. For continuous treatments, "correlations" (i.e., treatment-covariate Pearson correlations) is the default. Multiple options are allowed.
<code>int</code>	logical or numeric; whether or not to include 2-way interactions of covariates included in <code>covs</code> and in <code>add1</code> . If numeric, will be passed to <code>poly</code> as well.
<code>poly</code>	numeric; the highest polynomial of each continuous covariate to display. For example, if 2, squares of each continuous covariate will be displayed (in addition to the covariate itself); if 3, squares and cubes of each continuous covariate will be displayed, etc. If 1, the default, only the base covariate will be displayed. If <code>int</code> is numeric, <code>poly</code> will take on the value of <code>int</code> .
<code>distance</code>	an optional formula or data frame containing distance values (e.g., propensity scores) or a character vector containing their names. If a formula or variable names are specified, <code>bal.tab()</code> will look in the argument to <code>data</code> , if specified. Propensity scores generated by <code>weightit()</code> and <code>weightitMSM()</code> are automatically included and named "prop.score".
<code>add1</code>	an optional formula or data frame containing additional covariates for which to present balance or a character vector containing their names. If a formula or variable names are specified, <code>bal.tab()</code> will look in the arguments to the input object, <code>covs</code> , and <code>data</code> , if specified. For longitudinal treatments, can be a list of allowable arguments, one for each time point.
<code>data</code>	an optional data frame containing variables named in other arguments. For some input object types, this is required.
<code>continuous</code>	whether mean differences for continuous variables should be standardized ("std") or raw ("raw"). Default "std". Abbreviations allowed. This option can be set globally using <code>set.cobalt.options()</code> .
<code>binary</code>	whether mean differences for binary variables (i.e., difference in proportion) should be standardized ("std") or raw ("raw"). Default "raw". Abbreviations allowed. This option can be set globally using <code>set.cobalt.options()</code> .
<code>s.d.denom</code>	character; how the denominator for standardized mean differences should be calculated, if requested. See <code>col_w_smd()</code> for allowable options. Abbreviations allowed. If not specified, <code>bal.tab()</code> will figure out which one is best based on the estimand of the <code>weightit</code> object: if ATT, "treated"; if ATC, "control"; otherwise "pooled".
<code>thresholds</code>	a named vector of balance thresholds, where the name corresponds to the statistic (i.e., in <code>stats</code>) that the threshold applies to. For example, to request thresholds on mean differences and variance ratios, one can set <code>thresholds = c(m = .05, v = 2)</code> . Requesting a threshold automatically requests the display of that statistic. When specified, extra columns are inserted into the Balance table describing whether the requested balance statistics exceeded the threshold or not. Summary tables tallying the number of variables that exceeded and were within

	the threshold and displaying the variables with the greatest imbalance on that balance measure are added to the output.
weights	a vector, list, or data.frame containing weights for each unit, or a string containing the names of the weights variables in data, or an object with a <code>get.w()</code> method or a list thereof. The weights can be, e.g., inverse probability weights or matching weights resulting from a matching algorithm.
cluster	either a vector containing cluster membership for each unit or a string containing the name of the cluster membership variable in data or the input object. See class-bal.tab.cluster for details.
imp	either a vector containing imputation indices for each unit or a string containing the name of the imputation index variable in data or the input object. See class-bal.tab.imp for details. Not necessary if data is a mids object.
pairwise	whether balance should be computed for pairs of treatments or for each treatment against all groups combined. See bal.tab.multi() for details. This can also be used with a binary treatment to assess balance with respect to the full sample.
s.weights	Optional; either a vector containing sampling weights for each unit or a string containing the name of the sampling weight variable in data. These function like regular weights except that both the adjusted and unadjusted samples will be weighted according to these weights if weights are used. If s.weights was supplied in the call to <code>weightit()</code> or <code>weightitMSM()</code> , they will automatically be included and do not need be specified again (though there is no harm if they are).
abs	logical; whether displayed balance statistics should be in absolute value or not.
subset	a logical or numeric vector denoting whether each observation should be included or which observations should be included. If logical, it should have length equal to the number of units. NAs will be treated as FALSE. This can be used as an alternative to cluster to examine balance on subsets of the data.
quick	logical; if TRUE, will not compute any values that will not be displayed. Set to FALSE if computed values not displayed will be used later.
...	for some input types, other arguments that are required or allowed. Otherwise, further arguments to control display of output. See display options for details.

Details

`bal.tab.weightit()` generates a list of balance summaries for the `weightit` object given.

Value

For point treatments, if clusters and imputations are not specified, an object of class "bal.tab" containing balance summaries for the `weightit` object. See [bal.tab\(\)](#) for details.

If imputations are specified, an object of class "bal.tab.imp" containing balance summaries for each imputation and a summary of balance across imputations. See [class-bal.tab.imp](#) for details.

If `weightit()` is used with multi-category treatments, an object of class "bal.tab.multi" containing balance summaries for each pairwise treatment comparison. See [bal.tab.multi\(\)](#) for details.

If `weightitMSM()` is used for longitudinal treatments, an object of class `"bal.tab.msm"` containing balance summaries for each time period. See [class-bal.tab.msm](#) for details.

If clusters are specified, an object of class `"bal.tab.cluster"` containing balance summaries within each cluster and a summary of balance across clusters. See [class-bal.tab.cluster](#) for details.

See Also

- [bal.tab\(\)](#) for details of calculations.

Examples

```
library(WeightIt)
data("lalonge", package = "cobalt")

## Basic propensity score weighting
w.out1 <- weightit(treat ~ age + educ + race +
                  married + nodegree + re74 + re75,
                  data = lalonge)

bal.tab(w.out1, un = TRUE,
        thresholds = c(m = .1, v = 2))

## Weighting with a multi-category treatment
w.out2 <- weightit(race ~ age + educ + married +
                  nodegree + re74 + re75,
                  data = lalonge)

bal.tab(w.out2, un = TRUE)
bal.tab(w.out2, un = TRUE, pairwise = FALSE)

## IPW for longitudinal treatments
data("msmdata", package = "WeightIt")

wmsm.out <- weightitMSM(list(A_1 ~ X1_0 + X2_0,
                             A_2 ~ X1_1 + X2_1 +
                             A_1 + X1_0 + X2_0,
                             A_3 ~ X1_2 + X2_2 +
                             A_2 + X1_1 + X2_1 +
                             A_1 + X1_0 + X2_0),
                        data = msmdata)

bal.tab(wmsm.out)
```

Description

`bal.tab()` and `love.plot()` display balance statistics for the included covariates. The `stats` argument in each of these functions controls which balance statistics are to be displayed. The argument to `stats` should be a character vector with the names of the desired balance statistics.

This page describes all of the available balance statistics and how to request them. Abbreviations are allowed, so you can use the first few letters of each balance statistics to request it instead of typing out its whole name. That convention is used throughout the documentation. For example, to request mean differences and variance ratios in `bal.tab()` or `love.plot()`, you could include `stats = c("m", "v")`. In addition, the `thresholds` argument uses the same naming conventions and can be used to request balance thresholds on each statistic. For example, to request a balance threshold of .1 for mean differences, you could include `thresholds = c(m = .1)`.

Below, each allowable entry to `stats` and `thresholds` are described, along with other details or option that accompany them.

Binary/Multi-Category Treatments:

"mean.diffs" Mean differences as computed by `col_w_smd()`. Can be abbreviated as "m". Setting the arguments `continuous` and `binary` to either "std" or "raw" will determine whether standardized mean differences or raw mean differences are calculated for continuous and categorical variables, respectively. When standardized mean differences are requested, the `s.d.denom` argument controls how the standardization occurs. When `abs = TRUE`, negative values become positive. Mean differences are requested by default when no entry to `stats` is provided.

"variance.ratios" Variance ratios as computed by `col_w_vr()`. Can be abbreviated as "v". Will not be computed for binary variables. When `abs = TRUE`, values less than 1 will have their inverse taken. When used with `love.plot`, the x-axis scaled will be logged so that, e.g., .5 is as far away from 1 as 2 is.

"ks.statistics" Kolmogorov-Smirnov (KS) statistics as computed by `col_w_ks()`.

"ovl.coefficients" Overlapping (OVL) statistics as computed by `col_w_ovl()`. Can be abbreviated as "ovl". Additional arguments passed to `col_w_ovl()`, such as `integrate` or `bw`, can be supplied to `bal.tab()` or `love.plot()`.

Continuous Treatments:

"correlations" Pearson correlations as computed by `col_w_cov()`. Can be abbreviated as "cor". Setting the arguments `continuous` and `binary` to either "std" or "raw" will determine whether correlations or covariances are calculated for continuous and categorical variables, respectively (they are both "std" by default). When correlations are requested, the `s.d.denom` argument controls how the standardization occurs. When `abs = TRUE`, negative values become positive. Pearson correlations are requested by default when no entry to `stats` is provided.

"spearman.correlations" Spearman correlations as computed by `col_w_cov()`. Can be abbreviated as "sp". All arguments are the same as those for "correlations". When `abs = TRUE`, negative values become positive.

"distance.correlations" Distance correlations as computed by `col_w_dcov()`. Can be abbreviated as "dis". Setting the arguments `continuous` and `binary` to either "std" or "raw" will determine whether distance correlations or distance covariances are calculated for continuous and categorical variables, respectively (they are both "std" by default). When distance correlations are requested, the `s.d.denom` argument controls how the standardization occurs.

"mean.diffs.target" Mean differences computed between the weighted and unweighted sample to ensure the weighted sample is representative of the original population. Can be abbreviated as "m". Setting the arguments continuous and binary to either "std" or "raw" will determine whether standardized mean differences or raw mean differences are calculated for continuous and categorical variables, respectively. The standardization factor will be computed in the unweighted sample. When abs = TRUE, negative values become positive. This statistic is only computed for the adjusted samples.

"ks.statistics.target" KS statistics computed between the weighted and unweighted sample to ensure the weighted sample is representative of the original population. Can be abbreviated as "ks". This statistic is only computed for the adjusted samples.

"ovl.coefficients.target" Overlapping coefficients computed between the weighted and unweighted sample to ensure the weighted sample is representative of the original population. Can be abbreviated as "ovl". This statistic is only computed for the adjusted samples.

If a statistic is requested in thresholds, it will automatically be placed in stats. For example, `bal.tab(..., stats = "m", thresholds = c(v = 2))` will display both mean differences and variance ratios, and the variance ratios will have a balance threshold set to 2.

Examples

```
data(lalonde)

#Binary treatments
bal.tab(treat ~ age + educ + married + re74, data = lalonde,
       stats = c("m", "v", "ks"))
love.plot(treat ~ age + educ + married + re74, data = lalonde,
         stats = c("m", "v", "ks"), binary = "std",
         thresholds = c(m = .1, v = 2))

#Continuous treatments
bal.tab(re75 ~ age + educ + married + re74, data = lalonde,
       stats = c("cor", "sp"))
love.plot(re75 ~ age + educ + married + re74, data = lalonde,
         thresholds = c(cor = .1, sp = .1))
```

balance-summary

Compute Balance and Summary Statistics for Covariates

Description

These functions quickly compute balance statistics for the given covariates. These functions are used in `bal.tab()`, but they are available for use in programming without having to call `bal.tab()` to get them.

- `col_w_mean()` computes the (weighted) means for a set of covariates and weights and is essentially a weighted version of `colMeans()`.
- `col_w_sd()` computes the (weighted) standard deviations for a set of covariates and weights.

- `col_w_smd()` computes the (weighted) (absolute) (standardized) difference in means for a set of covariates, a binary treatment, and weights.
- `col_w_vr()` computes the (weighted) variance ratio for a set of covariates, a binary treatment, and weights.
- `col_w_ks()` computes the (weighted) Kolmogorov-Smirnov (KS) statistic for a set of covariates, a binary treatment, and weights.
- `col_w_ovl()` computes the complement of the (weighted) overlapping coefficient compliment for a set of covariates, a binary treatment, and weights (based on Franklin et al, 2014).
- `col_w_cov()` and `col_w_corr()` compute the (weighted) (absolute) treatment-covariate covariance or correlation for a set of covariates, a continuous treatment, and weights.
- `col_w_dcov()` and `col_w_dcorr()` compute the (weighted) treatment-covariate distance covariance or distance correlation for a set of covariates, a continuous treatment, and weights.

Usage

```
col_w_mean(  
  mat,  
  weights = NULL,  
  s.weights = NULL,  
  subset = NULL,  
  na.rm = TRUE,  
  ...  
)  
  
col_w_sd(  
  mat,  
  weights = NULL,  
  s.weights = NULL,  
  bin.vars,  
  subset = NULL,  
  na.rm = TRUE,  
  ...  
)  
  
col_w_smd(  
  mat,  
  treat,  
  weights = NULL,  
  std = TRUE,  
  s.d.denom = "pooled",  
  abs = FALSE,  
  s.weights = NULL,  
  bin.vars,  
  subset = NULL,  
  weighted.weights = weights,  
  na.rm = TRUE,  
  ...  
)
```

```
)

col_w_vr(
  mat,
  treat,
  weights = NULL,
  abs = FALSE,
  s.weights = NULL,
  bin.vars,
  subset = NULL,
  na.rm = TRUE,
  ...
)

col_w_ks(
  mat,
  treat,
  weights = NULL,
  s.weights = NULL,
  bin.vars,
  subset = NULL,
  na.rm = TRUE,
  ...
)

col_w_ovl(
  mat,
  treat,
  weights = NULL,
  s.weights = NULL,
  bin.vars,
  subset = NULL,
  na.rm = TRUE,
  integrate = TRUE,
  steps = 1001L,
  ...
)

col_w_cov(
  mat,
  treat,
  weights = NULL,
  type = "pearson",
  std = FALSE,
  s.d.denom = "all",
  abs = FALSE,
  s.weights = NULL,
  bin.vars,
```

```
subset = NULL,
weighted.weights = weights,
na.rm = TRUE,
...
)

col_w_corr(
  mat,
  treat,
  weights = NULL,
  type = "pearson",
  s.d.denom = "all",
  abs = FALSE,
  s.weights = NULL,
  bin.vars,
  subset = NULL,
  weighted.weights = weights,
  na.rm = TRUE,
  ...
)

col_w_dcov(
  mat,
  treat,
  weights = NULL,
  std = FALSE,
  s.d.denom = "all",
  s.weights = NULL,
  subset = NULL,
  weighted.weights = weights,
  na.rm = TRUE,
  ...
)

col_w_dcorr(
  mat,
  treat,
  weights = NULL,
  s.d.denom = "all",
  s.weights = NULL,
  subset = NULL,
  weighted.weights = weights,
  na.rm = TRUE,
  ...
)
```

Arguments

<code>mat</code>	a numeric matrix or a data frame containing the covariates for which the statistic is to be computed. If a data frame, <code>splitfactor()</code> with <code>drop.first = "if2"</code> will be called if any character or factor variables are present. This can slow down the function, so it's generally best to supply a numeric matrix. If a numeric vector is supplied, it will be converted to a 1-column matrix first.
<code>weights</code>	numeric; an optional set of weights used to compute the weighted statistics. If sampling weights are supplied through <code>s.weights</code> , the weights should not incorporate these weights, as <code>weights</code> and <code>s.weights</code> will be multiplied together prior to computing the weighted statistics.
<code>s.weights</code>	numeric; an optional set of sampling weights used to compute the weighted statistics. If weights are supplied through <code>weights</code> , <code>weights</code> and <code>s.weights</code> will be multiplied together prior to computing the weighted statistics. Some functions use <code>s.weights</code> in a particular way; for others, supplying <code>weights</code> and <code>s.weights</code> is equivalent to supplying their product to either <code>weights</code> or <code>s.weights</code> . See Details.
<code>subset</code>	a logical vector with length equal to the number of rows of <code>mat</code> used to subset the data. See Details for notes on its use with <code>col_w_smd()</code> , <code>col_w_cov()</code> , and <code>col_w_corr()</code> .
<code>na.rm</code>	logical; whether NAs should be ignored or not. If FALSE, any variable with any NAs will have its corresponding statistic returned as NA. If TRUE, any variable with any NAs will have its corresponding statistic computed as if the missing value were not there.
<code>...</code>	for all functions, additional arguments supplied to <code>splitfactor()</code> when <code>mat</code> is a data frame. <code>data</code> , <code>var.name</code> , <code>drop.first</code> , and <code>drop.level</code> are ignored; <code>drop.first</code> is automatically set to "if2". For <code>col_w_ovl()</code> , other arguments passed to <code>density()</code> besides <code>x</code> and <code>weights</code> . Note that the default value for <code>bw</code> when unspecified is "nrd" rather than the default in <code>density()</code> , which is "nrd0".
<code>bin.vars</code>	a vector used to denote whether each variable is binary or not. Can be a logical vector with length equal to the number of columns of <code>mat</code> or a vector of numeric indices or character names of the binary variables. If missing (the default), the function will figure out which covariates are binary or not, which can increase computation time. If NULL, it will be assumed no variables are binary. All functions other than <code>col_w_mean()</code> treat binary variables different from continuous variables. If a factor or character variable is in <code>mat</code> , all the dummies created will automatically be marked as binary, but it should still receive an entry when <code>bin.vars</code> is supplied as logical.
<code>treat</code>	a vector of treatment status for each individual. For <code>col_w_smd()</code> , <code>col_w_vr()</code> , <code>col_w_ks()</code> , and <code>col_w_ovl()</code> , <code>treat</code> should have exactly two unique values. For <code>col_w_cov()</code> , <code>col_w_corr()</code> , <code>col_w_dcov()</code> , and <code>col_w_dcorr()</code> , <code>treat</code> should be a many-valued numeric vector.
<code>std</code>	logical; for <code>col_w_smd()</code> , whether the computed mean differences for each variable should be standardized; for <code>col_w_cov()</code> , whether treatment-covariate correlations should be computed (TRUE) rather than covariances (FALSE); for

	<code>col_w_dcov()</code>, whether treatment-covariate distance correlations should be computed (TRUE) rather than distance covariances (FALSE). Can have either length 1, whereby all variables will be standardized or not, or length equal to the number of columns of <code>mat</code>, whereby only variables with a value of TRUE will be standardized. See Details.
<code>s.d.denom</code>	for <code>col_w_smd()</code>, <code>col_w_cov()</code>, and <code>col_w_dcov()</code> when <code>std</code> is TRUE for some variables, and for <code>col_w_corr()</code> and <code>col_w_dcorr()</code>, how the standardization factor should be computed. For <code>col_w_smd()</code> (i.e., when computing standardized mean differences), allowable options include • "treated" - uses the standard deviation of the variable in the treated group • "control" - uses the standard deviation of the variable in the control group • "pooled" - uses the square root of the average of the variances of the variable in the treated and control groups • "all" - uses the standard deviation of the variable in the full sample • "weighted" - uses the standard deviation of the variable in the full sample weighted by <code>weighted.weights</code> • "hedges" - uses the small-sample corrected version of Hedge's G described in the WWC Procedures Handbook (see References) • the name of one of the treatment values - uses the standard deviation of the variable in that treatment group. For <code>col_w_cov()</code>, <code>col_w_corr()</code>, <code>col_w_dcov()</code>, and <code>col_w_dcorr()</code>, only "all" and "weighted" are allowed. Abbreviations allowed. This can also be supplied as a numeric vector of standard deviations with length equal to the number of columns of <code>mat</code>; the values will be used as the standardization factors.
<code>abs</code>	logical; for <code>col_w_smd()</code>, <code>col_w_cov()</code>, and <code>col_w_corr()</code>, whether the returned statistics should be in absolute value (TRUE) or not. For <code>col_w_vr()</code>, whether the ratio should always include the larger variance in the numerator, so that the ratio is always greater than or equal to 1. Default is FALSE.
<code>weighted.weights</code>	for <code>col_w_smd()</code>, <code>col_w_cov()</code>, <code>col_w_corr()</code>, <code>col_w_dcov()</code>, and <code>col_w_dcorr()</code>, when <code>std = TRUE</code> and <code>s.d.denom = "weighted"</code>, a vector of weights to be applied to the computation of the denominator standard deviation. If not specified, will use the argument to <code>weights</code>. When <code>s.d.denom</code> is not "weighted", this is ignored. The main purpose of this is to allow weights to be NULL while weighting the denominator standard deviations for assessing balance in the unweighted sample but using the standard deviations of the weighted sample.
<code>integrate</code>	logical; for <code>col_w_ovl()</code>, whether to use integrate() to calculate the area of overlap for continuous variables. If FALSE, a midpoint Riemann sum will be used instead. The Riemann sum is a little slower and very slightly imprecise (unnoticeably in most contexts). When TRUE, <code>integrate()</code> will be tried, and if it fails, the Riemann sum will be used as a fallback. The default (TRUE) is to use <code>integrate()</code> when possible.
<code>steps</code>	for <code>col_w_ovl()</code>, the number of points to use to compute the Riemann sum to approximate the integral. Used when <code>integrate = FALSE</code> and also when <code>integrate = TRUE</code> but integrate() fails and the Riemann sum is used as a fallback. Default is 1001 for 1000 partitions.

type for `col_w_cov()` and `col_w_corr()`, the type of covariance/correlation to be computed. Allowable options include "pearson" and "spearman". When "spearman" is requested, the covariates and treatment are first turned into ranks using `rank()` with `na.last = "keep"`.

Details

`col_w_mean()` computes column weighted means for a matrix of variables. It is similar to `colMeans()` but (optionally) incorporates weights. `weights` and `s.weights` are multiplied together prior to being used, and there is no distinction between them. This could be used to compute the weighted means of each covariate in the general population to examine the degree to which a weighting method has left the weighted samples resembling the original population.

`col_w_sd()` computes column weighted standard deviations for a matrix of variables. `weights` and `s.weights` are multiplied together prior to being used, and there is no distinction between them. The variance of binary variables is computed as $p(1 - p)$, where p is the (weighted) proportion of 1s, while the variance of continuous variables is computed using the standard formula; the standard deviation is the square root of this variance.

`col_w_smd()` computes the mean difference for each covariate between treatment groups defined by `treat`. These mean differences can optionally be weighted, standardized, and/or in absolute value. The standardization factor is computed using the unweighted standard deviation or variance when `s.weights` are absent, and is computed using the `s.weights`-weighted standard deviation or variance when `s.weights` are present, except when `s.d.denom = "weighted"`, in which case the product of `weighted.weights` and `s.weights` (if present) are used to weight the standardization factor. The standardization factor is computed using the whole sample even when `subset` is used. Note that unlike `bal.tab()`, `col_w_smd()` requires the user to specify whether each individual variable should be standardized using `std` rather than relying on continuous or binary. The weighted mean difference is computed using the product of `weights` and `s.weights`, if specified. The variance of binary variables is computed as $p(1 - p)$, where p is the (weighted) proportion of 1s, while the variance of continuous variables is computed using the standard formula.

`col_w_vr()` computes the variance ratio for each covariate between treatment groups defined by `treat`. When `abs = TRUE`, `pmax(out, 1/out)` is applied to the output so that the ratio is always greater than or equal to 1. For binary variables, the variance is computed as $p(1 - p)$, where p is the (weighted) proportion of 1s, while the variance of continuous variables is computed using the standard formula. Note that in `bal.tab()`, variance ratios are not computed for binary variables, while here, they are (but likely should not be interpreted). `weights` and `s.weights` are multiplied together prior to being used, and there is no distinction between them. Because of how the weighted variance is computed, exactly balanced groups may have variance ratios that differ slightly from 1.

`col_w_ks()` computes the KS statistic for each covariate using the method implemented in **twang**. The KS statistics can optionally be weighted. For binary variables, the KS statistic is just the difference in proportions. `weights` and `s.weights` are multiplied together prior to being used, and there is no distinction between them.

`col_w_ovl()` computes the complement of the overlapping coefficient as described by Franklin et al. (2014). It does so by computing the density of the covariate in the treated and control groups, then finding the area where those density overlap, and subtracting that number from 1, yielding a value between 0 and 1 where 1 indicates complete imbalance, and 0 indicates perfect balance. `density()` is used to model the density in each group. The bandwidth of the covariate in the smaller treatment group is used for both groups. The area of overlap can be computed using

`integrate`, which quickly and accurately computes the integral, or using a midpoint Riemann sum with 1000 partitions, which approximates the area more slowly. A reason to prefer the Riemann sum is that `integrate` can fail for unknown reasons, though Riemann sums will fail with some extreme distributions. When either method fails, the resulting value will be NA. For binary variables, the complement of the overlapping coefficient is just the difference in proportions. `weights` and `s.weights` are multiplied together prior to being used, and there is no distinction between them. The weights are used to compute the weighted density by supplying them to the `weights` argument of `density`.

`col_w_cov()` computes the covariances between a continuous treatment and the covariates to assess balance for a continuous treatment as recommended in Austin (2019). These covariances can optionally be weighted or in absolute value or can be requested as correlations (i.e., standardized covariances). Each correlation is computed as the covariance between the treatment and covariate divided by a standardization factor, which is equal to the square root of the product of the variance of treatment and the variance of the covariate. The standardization factor is computed using the unweighted variances when `s.weights` are absent, and is computed using the sampling weighted variances when `s.weights` are present, except when `s.d.denom = "weighted"`, in which case the product of `weighted.weights` and `s.weights` (if present) are used to weight the standardization factor. For this reason, the computed correlation can be greater than 1 or less than -1. The standardization factor is always computed using the whole sample even when `subset` is used. The covariance is computed using the product of `weights` and `s.weights`, if specified. The variance of binary variables is computed as $p(1 - p)$, where p is the (weighted) proportion of 1s, while the variance of continuous variables is computed using the standard formula.

`col_w_corr()` is a wrapper for `col_w_cov` with `std` set to `TRUE`.

`col_w_dcov()` computes the distance covariances between a continuous treatment and the covariates to assess balance for a continuous treatment. A multivariate version is described by Huling et al. (2023) for computing a scalar value that represents the balance for all covariates simultaneously; the statistic computed here is for one covariate at a time. The distance covariances can optionally be weighted or can be requested as distance correlations (i.e., standardized distance covariances). The distance correlations are computed as the distance covariance between the treatment and covariate divided by a standardization factor, which is equal to the square root of the product of the distance variance of treatment and the distance variance of the covariate, where the distance variance is the distance covariance of a variable with itself. The standardization factor is computed using the unweighted distance variances when `s.weights` are absent, and is computed using the sampling weighted distance variances when `s.weights` are present, except when `s.d.denom = "weighted"`, in which case the product of `weighted.weights` and `s.weights` (if present) are used to weight the standardization factor. For this reason, the computed distance correlation can be greater than 1. The standardization factor is always computed using the whole sample even when `subset` is used. The distance covariance is computed using the product of `weights` and `s.weights`, if specified.

`col_w_dcorr()` is a wrapper for `col_w_dcov` with `std` set to `TRUE`.

Value

A vector of balance statistics, one for each variable in `mat`. If `mat` has column names, the output will be named as well.

References

- Austin, P. C. (2019). Assessing covariate balance when using the generalized propensity score with quantitative or continuous exposures. *Statistical Methods in Medical Research*, 28(5), 1365–1377. doi:10.1177/0962280218756159
- Franklin, J. M., Rassen, J. A., Ackermann, D., Bartels, D. B., & Schneeweiss, S. (2014). Metrics for covariate balance in cohort studies of causal effects. *Statistics in Medicine*, 33(10), 1685–1699. doi:10.1002/sim.6058
- Huling, J. D., Greifer, N., & Chen, G. (2023). Independence Weights for Causal Inference with Continuous Treatments. *Journal of the American Statistical Association*, 0(0), 1–14. doi:10.1080/01621459.2023.2213485
- What Works Clearinghouse. (2020). WWC Procedures Handbook (Version 4.1). Retrieved from <https://ies.ed.gov/ncee/wwc/Handbooks>

See Also

- [bal.tab\(\)](#)
- [bal.compute\(\)](#)
- [balance-statistics](#)

Examples

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

treat <- lalonde$treat
covs <- subset(lalonde, select = -c(treat, re78))
covs0 <- splitfactor(covs, drop.first = "if2")
bin.vars <- c(FALSE, FALSE, TRUE, TRUE, TRUE,
              TRUE, TRUE, FALSE, FALSE)

W <- WeightIt::weightit(treat ~ covs, method = "glm",
                       estimand = "ATE")

weights <- W$weights

round(data.frame(
  m0 = col_w_mean(covs0, weights = weights, subset = treat == 0),
  sd0 = col_w_sd(covs0, weights = weights,
                bin.vars = bin.vars, subset = treat == 0),
  m1 = col_w_mean(covs0, weights = weights, subset = treat == 1),
  sd1 = col_w_sd(covs0, weights = weights,
                bin.vars = bin.vars, subset = treat == 1),
  smd = col_w_smd(covs0, treat = treat, weights = weights,
                 std = TRUE, bin.vars = bin.vars),
  vr = col_w_vr(covs0, treat = treat, weights = weights,
               bin.vars = bin.vars),
  ks = col_w_ks(covs0, treat = treat, weights = weights,
               bin.vars = bin.vars),
  ovl = col_w_ovl(covs0, treat = treat, weights = weights,
                 bin.vars = bin.vars),
  row.names = colnames(covs0)
), 4)
```

```
# Compare to bal.tab():
bal.tab(covs, treat = treat, weights = weights,
        disp = c("m", "sd"),
        stats = c("m", "v", "ks", "ovl"),
        estimand = "ATE", method = "weighting",
        binary = "std")
```

class-bal.tab.cens *Using bal.tab() with a Censoring Indicator*

Description

When `bal.tab()` is given a censoring indicator marked with `.cens()` rather than a treatment, its target is the full at-risk sample, and what matters is whether the units still under observation resemble that sample once weighted. This page outlines the output in this case.

Balance is therefore assessed between two samples built from the same units:

- **Uncensored:** the units with `C == 0`, carrying the censoring weights.
- **Full:** every at-risk unit, i.e., every unit with a non-missing `C`, carrying a weight of 1.

Setting `un = TRUE` adds the same comparison with the uncensored units unweighted, which is the imbalance the weights were estimated to remove.

The two samples are compared using the same statistics available for binary treatments (`mean.diffs`, `variance.ratios`, `ks.statistics`, and `ovl.coefficients`), because internally they are stacked into a binary comparison. The two samples name their own columns rather than taking a binary treatment's positional 0 and 1, so the balance table has `M.Uncensored` and `SD.Uncensored` for the uncensored sample and `M.Full` and `SD.Full` for the full one, and `Diff` is the difference between the full sample and the uncensored one.

`bal.plot()` works too, showing the weighted uncensored sample against the unweighted full sample.

Allowable arguments

Every argument to `bal.tab()` applies as it does with a binary treatment, with the following exceptions.

`s.d.denom` allowable values are "full" (the default), which uses the standard deviation of the covariate in the full at-risk sample; "uncensored", which uses that of the uncensored sample; and "pooled", "all", "weighted", and "hedges", which behave as they do for a binary treatment.

`estimand` **and** `focal` do not apply and are ignored with a warning. The target is the full at-risk sample.

`subclass` applies: subclassifying is itself a way of solving a censoring problem, in that within each subclass the uncensored units should resemble every at-risk unit in it. See "With subclasses" below.

`match.strata` does not apply.

`cluster` and `imp` apply as usual, and produce a `bal.tab.cluster` or `bal.tab.imp` object whose per-cluster or per-imputation components are the censoring balance tables described here.

Among longitudinal treatments

A censoring indicator can appear among a list of longitudinal treatments, as in `list(A1 ~ x, .cens(C1) ~ x, A2 ~ x)`, which is how a joint treatment-and-censoring model is written for `WeightIt::weightitMSM()`; `bal.tab()` also accepts such a `weightitMSM` object directly. Each entry of the list gets a table of its own kind, and each is assessed among the units still under observation entering it, so the full sample a censoring indicator is compared against is the risk set at that time point rather than the original cohort, and a treatment after it is assessed only among the units it did not censor. A list that mixes censoring with treatment gets no balance summary across time points. See [class-bal.tab.msm](#).

With subclasses

Subclassification is an alternative to weighting for solving a censoring problem: within each subclass, the units still under observation should resemble every at-risk unit in that subclass. Supplying subclass therefore produces a `bal.tab.cens` object that also inherits from `bal.tab.subclass`, with a balance table for each subclass and a summary across them, as described at [class-bal.tab.subclass](#).

The summary across subclasses is subclassification expressed as censoring weights: a unit still under observation in subclass k receives n_k/n_{k1} , where n_k is the number of at-risk units in the subclass and n_{k1} the number of them still under observation, and the full sample is left unweighted. This is the same summary one would get from supplying those weights to a censoring model directly.

The sample sizes have one column per subclass, with `Full`, `Uncensored`, and `Censored` rows.

Output

The output is a `bal.tab.cens` object, which inherits from `bal.tab.bin` and `bal.tab` (or from `bal.tab.subclass` when subclass is supplied), and has the same elements as an ordinary binary-treatment `bal.tab` object. Its `Observations` component differs: it has a single `Total` column, because the target sample is the same in every row, and the rows

- `Full`: the number (or effective number) of at-risk units,
- `Uncensored`: the number of units still under observation,
- `Adjusted` (or one row per set of weights): their effective number once weighted,
- `Censored`: the number of units censored, omitted when there are none.

`Uncensored` and `Censored` sum to `Full`.

See Also

- [bal.tab\(\)](#)
- [.cens\(\)](#) for marking an indicator as censoring
- [class-bal.tab.cluster](#), [class-bal.tab.imp](#), [class-bal.tab.subclass](#), and [class-bal.tab.msm](#) for the segmented and longitudinal cases
- [treat](#) for the attributes that decide how a treatment is compared and what its groups are called

`class-bal.tab.cluster` *Using bal.tab() with Clustered Data*

Description

When using `bal.tab()` with clustered or subgrouped data, the output will be different from the case with single-level data, and there are some options that are common across all `bal.tab()` methods. This page outlines the outputs and options in this case.

There are two main components of the output of `bal.tab()` with clustered data: the within-cluster balance summaries and the across-cluster balance summary. The within-cluster balance summaries display balance for units within each cluster separately.

The across-cluster balance summary pools information across the within-cluster balance summaries to simplify balance assessment. It provides a combination (e.g., mean or maximum) of each balance statistic for each covariate across all clusters. This allows you to see how bad the worst imbalance is and what balance looks like on average. The balance summary will not be computed if longitudinal treatments, multi-category treatments, or multiply imputed data are used.

In order to use the `thresholds` argument with `bal.tab()` with clustered data and the balance summary across clusters displayed, `cluster.fun` must be supplied and set to a single string, which is not the default.

Allowable arguments

There are four arguments for each `bal.tab()` method that can handle clustered data: `cluster`, `which.cluster`, `cluster.summary`, and `cluster.fun`.

`cluster` A vector of cluster membership. This can be factor, character, or numeric vector. This argument is required to let `bal.tab()` know that the data is clustered. If a `data` argument is specified, this can also be the name of a variable in `data` that contains cluster membership.

`which.cluster` This is a display option that does not affect computation. If `.all` (the default), all clusters in `cluster` will be displayed. If `.none`, no clusters will be displayed. Otherwise, can be a vector of cluster names or numerical indices for which to display balance. Indices correspond to the alphabetical order of cluster names (or the order of cluster levels if a factor).

`cluster.summary` This is a display option that does not affect computation. If `TRUE`, the balance summary across clusters will be displayed. The default is `TRUE`, and if `which.cluster` is `.none`, it will automatically be set to `TRUE`.

`cluster.fun` This is a display option that does not affect computation. Can be "min", "mean", or "max" and corresponds to which function is used in the across-cluster summary to combine results across clusters. For example, if `cluster.fun = "mean"` the mean balance statistic across clusters will be displayed. The default when `abs = FALSE` in the `bal.tab()` call is to display all three. The default when `abs = TRUE` in the `bal.tab()` call is to display just the mean and maximum absolute balance statistic.

Output

The output is a `bal.tab.cluster` object, which inherits from `bal.tab`. It has the following elements:

- `Cluster.Balance`: For each cluster, a regular `bal.tab` object containing a balance table, a sample size summary, and other balance assessment tools, depending on which options are specified.
- `Cluster.Summary`: The balance summary across clusters. This will include the combination of each balance statistic for each covariate across all clusters according to the value of `cluster.fun`.
- `Observations`: A table of sample sizes or effective sample sizes for each cluster before and after adjustment.

As with other methods, multiple weights can be specified, and values for all weights will appear in all tables.

See Also

- `bal.tab()`
- `bal.tab.data.frame()`
- `print.bal.tab()`
- `vignette("segmented-data")` for examples

class-bal.tab.imp

Using bal.tab() with Multiply Imputed Data

Description

When using `bal.tab()` with multiply imputed data, the output will be different from the case with a single data set. Multiply imputed data can be used with all `bal.tab()` methods, and the `mimids` and `wimids` methods for **MatchThem** objects automatically incorporate multiply imputed data. This page outlines the outputs and options available with multiply imputed data.

There are two main components of the output of `bal.tab()` with multiply imputed data: the within-imputation balance summaries and the across-imputation balance summary. The within-imputation balance summaries display balance for units within each imputed data set separately. In general, this will not be very useful because interest rarely lies in the qualities of any individual imputed data set.

The across-imputation balance summary pools information across the within-imputation balance summaries to simplify balance assessment. It provides the average, smallest, and largest balance statistic for each covariate across all imputations. This allows you to see how bad the worst imbalance is and what balance looks like on average across the imputations. The summary behaves differently depending on whether `abs` is specified as `TRUE` or `FALSE`. When `abs = TRUE`, the across-imputation balance summary will display the mean absolute balance statistics and the maximum absolute balance statistics. When `abs = FALSE`, the across-imputation balance summary will display the minimum, mean, and maximum of the balance statistic in its original form.

In order to use the `thresholds` argument with `bal.tab()` with multiply imputed data and the balance summary across imputations displayed, `imp.fun` must be supplied and set to a single string, which is not the default. See `vignette("segmented-data")` for details.

Allowable arguments

There are four arguments for each `bal.tab()` method that can handle multiply imputed data: `imp`, `which.imp`, `imp.summary`, and `imp.fun`.

imp A vector of imputation membership. This can be factor, character, or numeric vector. This argument is required to let `bal.tab()` know that the data is multiply imputed unless **MatchThem** objects are used. If a `data` argument is specified, this can also be the name of a variable in `data` that contains imputation membership. If the `data` argument is a `mids` object, the output of a call to `mice()`, `imp` does not need to be specified and will automatically be extracted from the `mids` object.

which.imp This is a display option that does not affect computation. If `.all`, all imputations in `imp` will be displayed. If `.none` (the default), no imputations will be displayed. Otherwise, can be a vector of imputation indices for which to display balance.

imp.summary This is a display option that does not affect computation. If `TRUE`, the balance summary across imputations will be displayed. The default is `TRUE`, and if `which.imp` is `.none`, it will automatically be set to `TRUE`.

imp.fun This is a display option that does not affect computation. Can be `"min"`, `"mean"`, or `"max"` and corresponds to which function is used in the across-imputation summary to combine results across imputations. For example, if `imp.fun = "mean"` the mean balance statistic across imputations will be displayed. The default when `abs = FALSE` in the `bal.tab()` call is to display all three. The default when `abs = TRUE` in the `bal.tab()` call is to display just the mean and maximum absolute balance statistic.

Output

The output is a `bal.tab.imp` object, which inherits from `bal.tab`. It has the following elements:

- **Imputation.Balance:** For each imputation, a regular `bal.tab` object containing a balance table, a sample size summary, and other balance assessment tools, depending on which options are specified.
- **Balance.Across.Imputations:** The balance summary across imputations. This will include the combination of each balance statistic for each covariate across all imputations according to the value of `imp.fun`.
- **Observations:** A table of sample sizes or effective sample sizes averaged across imputations before and after adjustment.

As with other methods, multiple weights can be specified, and values for all weights will appear in all tables.

See Also

- [bal.tab\(\)](#)
- [bal.tab.data.frame\(\)](#)

- `print.bal.tab()`
- `vignette("segmented-data")` for examples

class-bal.tab.msm

Using bal.tab() with Longitudinal Treatments

Description

When using `bal.tab()` with longitudinal treatments, the output will be different from the case with point treatments, and there are some options that are common across all `bal.tab()` methods for dealing with longitudinal data. This page outlines the outputs and options in this case.

There are two main components of the output of `bal.tab()` with longitudinal treatments: the time-point-specific balance summary and across-time-points balance summary. The time-point-specific balance summaries are standard point treatment balance summaries at each time point.

The across-time-points balance summary is, for each variable, the greatest imbalance across all time-point-specific balance summaries. If the greatest observed imbalance is tolerable, then all other imbalances for that variable will be tolerable too, so focusing on reducing the greatest imbalance is sufficient for reducing imbalance overall. The balance summary will not be computed if multi-category treatments or multiply imputed data are used, or if the time points are not all of the same kind (see below).

Mixing treatments and censoring

One entry of the list may be a censoring indicator marked with `.cens()` rather than a treatment, as in `list(A1 ~ x, .cens(C1) ~ x, A2 ~ x)`, which is how a joint treatment-and-censoring model is written for `WeightIt::weightitMSM()`. `bal.tab()` produces one table per entry, of whichever kind that entry is: an ordinary balance table for a treatment, and the censoring balance table described at `class-bal.tab.cens` for an indicator.

Each entry is assessed among the units still under observation entering it. A unit is under observation until a censoring indicator earlier in the list marks it censored; the risk set is accumulated from the indicators themselves rather than read off the treatments, so it makes no difference whether the data records a treatment for a unit that has already dropped out or leaves it missing. A missing treatment for a unit that is still under observation is an error naming the time point it appeared in. A censoring entry's own comparison uses the risk set as it stands entering it, so it includes the units it is about to remove; those are the full sample it compares against.

A censoring balance table and a treatment balance table say different things about different samples, so a list that mixes them gets no balance summary across time points, exactly as a list mixing continuous and binary treatments does not. Each time point's own table still reports its sample sizes. A list in which every entry is a censoring indicator is not a mixture and is summarized as usual.

Each time point's default `s.d.denom` is the one its own kind of model implies – "pooled" for a binary or multi-category treatment, "all" for a continuous one, and "full" for a censoring indicator – so that a model gives the same numbers in a list as it would on its own. A value supplied to `s.d.denom` is shared by every time point, which is one reason it is recommended not to set it for longitudinal treatments.

Naming the time points

Each time point is named for its position in the list, whether the model there is a treatment or a censoring model, and the variable it is about, as in 1. Treatment: A_1 or 2. Censoring: C_1. The number is the position in the list rather than the treatment period, so that it is the number which.time takes and a censoring model has one too; which.time also accepts the variable name on its own (e.g., "C_1").

Allowable arguments

There are two additional arguments for each bal.tab() method that can handle longitudinal treatments: which.time and msm.summary.

which.time This is a display option that does not affect computation. If .all (the default), all time points will be displayed. If .none, no time points will be displayed. Otherwise, can be a vector of treatment names or indices for which to display balance.

msm.summary This is a display option that does not affect computation. If TRUE, the balance summary across time points will be displayed. The default is TRUE, and if which.time is .none, it will automatically be set to TRUE.

Output

The output is a bal.tab.msm object, which inherits from bal.tab. It has the following elements:

- **Time.Balance:** For each time point, a regular bal.tab object containing a balance table, a sample size summary, and other balance assessment tools, depending on which options are specified.
- **Balance.Across.Times:** The balance summary across time points. This will include the maximum balance statistic(s) for each covariate across all time points. Absent when the time points are not all of the same kind.
- **Observations:** A list with a table of sample sizes or effective sample sizes for each time point, before and after adjustment. Always present, but displayed only alongside the balance summary across time points: it gathers in one place what each time point's own table has already reported, so without the summary there is nothing it adds.

As with other methods, multiple weights can be specified, and values for all weights will appear in all tables.

Note

The balance tables presented here are not the same as those recommended by Jackson (2016) and computed in his R package, **confoundr**, as these do not take into account treatment history. The balance statistics presented here should be used with caution and may not reflect balance in an accurate way.

References

Jackson, J. W. (2016). Diagnostics for Confounding of Time-varying and Other Joint Exposures: *Epidemiology*, 27(6), 859–869. doi:10.1097/EDE.0000000000000547

See Also

- `bal.tab()`
- `bal.tab.time.list()`
- `print.bal.tab()`
- `.cens()` and `class-bal.tab.cens` for a censoring indicator among the time points
- `vignette("longitudinal-treat")` for examples

class-bal.tab.multi *Using bal.tab() with Multi-Category Treatments*

Description

When using `bal.tab()` with multi-category treatments, the output will be different from the case with binary or continuous treatments, and there are some options that are common across all `bal.tab()` methods. This page outlines the outputs and options in this case.

There are two main components of the output of `bal.tab()` with multi-category treatments: the two-group treatment comparisons and the balance summary. The two-group treatment comparisons are standard binary treatment comparison either for pairs of groups (e.g., for treatments A, B, and C, "A vs. B", "A vs. C", and "B vs. C") or each group against all the groups (i.e., the entire sample).

The balance summary is, for each variable, the greatest imbalance across all two-group comparisons. So, for variable X1, if "A vs. B" had a standardized mean difference of 0.52, "A vs. C" had a standardized mean difference of .17, and "B vs. C" had a standardized mean difference of .35, the balance summary would have 0.52 for the value of the standardized mean difference for X1. The same goes for other variables and other measures of balance. If the greatest observed imbalance is tolerable, then all other imbalances for that variable will be tolerable too, so focusing on reducing the greatest imbalance is sufficient for reducing imbalance overall. (Note that when `s.d.denom = "pooled"`, i.e., when the estimand is the ATE, the pooled standard deviation in the denominator will be the average of the standard deviations across all treatment groups, not just those used in the pairwise comparison.) The balance summary will not be computed if multiply imputed data are used.

Allowable arguments

There are four arguments for each `bal.tab()` method that can handle multi-category treatments: `pairwise`, `focal`, `which.treat`, and `multi.summary`.

pairwise Whether to compute the two-group comparisons pairwise or not. If `TRUE`, `bal.tab()` will compute comparisons for each pair of treatments. This can be valuable if treatments are to be compared with one another (which is often the case). If `FALSE`, `bal.tab()` will compute balance for each treatment group against the full unadjusted sample when `focal` is `NULL` and for each non-focal group against the focal group otherwise.

focal When one group is to be compared to multiple control groups in an ATT analysis, the group considered "treated" is the focal group. By specifying the name or index of the treatment condition considered focal, `bal.tab()` will only compute and display pairwise balance for treatment comparisons that include the focal group when `pairwise = FALSE`.

`which.treat` This is a display option that does not affect computation. When displaying the `bal.tab` output, which treatments should be displayed? If a vector of length 1 is entered, all comparisons involving that treatment group will be displayed. If a vector of length 2 or more is entered, all comparisons involving treatments that both appear in the input will be displayed. For example, inputting "A" will display "A vs. B" and "A vs. C", while entering `c("A", "B")` will only display "A vs. B". `.none` indicates no treatment comparisons will be displayed, and `.all` indicates all treatment comparisons will be displayed. `.none` is the default.

`multi.summary` If TRUE, the balance summary across all comparisons will be computed and displayed. This includes one row for each covariate with maximum balance statistic across all pairwise comparisons. Note that, if variance ratios or KS statistics are requested in addition to mean differences, the displayed values may not come from the same pairwise comparisons; that is, the greatest standardized mean difference and the greatest variance ratio may not come from the same comparison. The default is TRUE, and if `which.treat` is `.none`, it will automatically be set to TRUE.

Output

The output is a `bal.tab.multi` object, which inherits from `bal.tab`. It has the following elements:

- `Pair.Balance`: For each pair of treatment groups, a regular `bal.tab` object containing a balance table, a sample size summary, and other balance assessment tools, depending on which options are specified. If `pairwise` is FALSE, the comparisons will be between each group and the groups combined (labeled "All") when `focal` is NULL and between each non-focal group and the focal group otherwise.
- `Balance.Across.Pairs`: The balance summary across two-group comparisons. This will include the greatest (i.e., maximum) absolute balance statistics(s) for each covariate across all comparisons computed. Thresholds can be requested for each balance measure as with binary treatments.
- `Observations`: A table of sample sizes or effective sample sizes for each treatment group before and after adjustment.

As with other methods, multiple weights can be specified, and values for all weights will appear in all tables.

Note

In versions 4.3.1 and earlier, setting `pairwise = FALSE` would compare each group to the full adjusted sample. Now, each group is compared to the full *unadjusted* sample (unadjusted except for `s.weights`, if supplied).

In versions 4.3.1 and earlier, `pairwise` was ignored with non-NULL `focal` and was automatically set to FALSE. `pairwise` can be specified and its default is now TRUE, so balance between all treatment groups will be computed by default rather than only between each non-group and the focal group. To recover previous functionality, set `pairwise = FALSE` with non-NULL `focal`.

See Also

- [bal.tab\(\)](#)

- `bal.tab.data.frame()`
- `print.bal.tab()`
- `vignette("segmented-data")` for examples

class-bal.tab.subclass

Using bal.tab() with Subclassified Data

Description

When using `bal.tab()` with subclassified data, i.e., data split into subclasses where balance may hold, the output will be different from the standard, non-subclassified case, and there is an additional option for controlling display. This page outlines the outputs and options in this case.

There are two main components of the output of `bal.tab()` with subclassified data: the balance within subclasses and the balance summary across subclasses. The within-subclass balance displays essentially are standard balance displays for each subclass, except that only "adjusted" values are available, because the subclassification itself is the adjustment.

The balance summary is, for each variable, like a weighted average of the balance statistics across subclasses. With a binary treatment, this is computed internally by assigning each individual a weight based on their subclass and treatment group membership and then computing weighted balance statistics as usual with these weights. This summary is the same one would get if subclasses were supplied to the `match.strata` argument rather than to `subclass`. Because the means and mean differences are additive, their computed values will be weighted averages of the subclass-specific values, but for other statistics, the computed values will not be.

With a censoring indicator, subclassification is itself the solution: within each subclass the units still under observation should resemble every at-risk unit in it, and the summary across subclasses is that subclassification expressed as censoring weights. See `class-bal.tab.cens`.

With a continuous treatment, subclassification cannot be expressed as weights, because no set of weights makes a continuous treatment independent of the covariates within a subclass. The balance summary is therefore computed by combining the subclass-specific statistics directly, weighting each subclass by its share of the subclassified units (using `s.weights`, if supplied). Every statistic is a weighted average of the subclass-specific values, except the standard deviations, which are combined in quadrature so that the summary value is the pooled within-subclass standard deviation. The summary means are consequently the same as the means in the original sample, since subclassification does not change the distribution of the covariates.

Allowable arguments

There are three arguments for `bal.tab()` that relate to subclasses: `subclass`, `which.subclass`, and `subclass.summary`.

subclass For the `data.frame` and `formula` methods of `bal.tab()`, a vector of subclass membership or the name of the variable in `data` containing subclass membership. When using subclassification with a function compatible with **cobalt**, such as `matchit()` in **MatchIt**, this argument can be omitted because the subclasses are in the output object.

`which.subclass` This is a display option that does not affect computation. If `.all`, all subclasses in `subclass` will be displayed. If `.none` (the default), no subclasses will be displayed. Otherwise, can be a vector of subclass indices for which to display balance.

`subclass.summary` This is a display option that does not affect computation. If `TRUE`, the balance summary across subclasses will be displayed. The default is `TRUE`, and if `which.subclass` is `.none`, it will automatically be set to `TRUE`.

Output

The output is a `bal.tab.subclass` object, which inherits from `bal.tab`. It has the following elements:

- `Subclass.Balance`: A list of data frames containing balance information for each covariate in each subclass.
- `Balance.Across.Subclass`: A data frame containing balance statistics for each covariate aggregated across subclasses and for the original sample (i.e., unadjusted). See [bal.tab\(\)](#) for details on what this includes.
- `Observations`: A table of sample sizes in each subclass and overall.

See Also

- [bal.tab\(\)](#)
- [bal.tab.data.frame\(\)](#)
- [print.bal.tab\(\)](#)

display-options

Options for Displaying bal.tab() Output

Description

Several additional arguments can be passed to [bal.tab\(\)](#) that control the display of the output; these arguments are documented here. Not all arguments are applicable to all uses of `bal.tab()`; for example, `which.subclass`, which controls which subclasses are displayed when subclassification is used, won't do anything when subclassification is not used. Note that when `quick = TRUE` is set in the call to `bal.tab()` (which is the default), setting any of these arguments to `FALSE` can prevent some values from being computed, which can have unintended effects.

Allowed arguments

`disp.bal.tab` logical; whether to display the table of balance statistics. Default is `TRUE`, so the balance table is displayed.

`imbalanced.only` logical; whether to display only the covariates that failed to meet at least one of balance thresholds. Default is `FALSE`, so all covariates are displayed.

`un` logical; whether to print statistics for the unadjusted sample as well as for the adjusted sample. Default is `FALSE`, so only the statistics for the adjusted sample are displayed.

`disp` character; which distribution summary statistic(s) should be reported. Allowable options include "means" and "sds". Multiple options are allowed. Abbreviations allowed.

`stats` character; which statistic(s) should be reported. See [stats](#) to see which options are available. Multiple options are allowed. Abbreviations allowed. For binary and multi-category treatments, the default is "mean.diff" (i.e., [standardized] mean differences), and for continuous treatments, the default is "correlations" (i.e., treatment-covariate Pearson correlations).

`factor_sep` character; the string used to separate factor variables from their levels when variable names are printed. Default is "_". See the section *Variable names using var.names* below, which covers the separators too.

`int_sep` character; the string used to separate two variables involved in an interaction when variable names are printed. Default is "*". Older versions of **cobalt** used "_".

`disp.call` logical; whether to display the function call from the original input object, if present. Default is FALSE, so the function call is not displayed.

`var.names` an optional object providing alternate names for the variables, which will otherwise be displayed as they are stored. See the section *Variable names using var.names* below.

When subclassification is used:

`which.subclass` Which subclasses (if any) should be displayed. If `.all`, all subclasses will be displayed. If `.none` (the default), no subclasses will be displayed. Otherwise, can be a vector of subclass indices for which to display balance.

`subclass.summary` logical; whether to display the balance summary across subclasses. If TRUE, the balance summary across subclasses will be displayed. The default is TRUE, and if `which.subclass` is `.none`, it will automatically be set to TRUE.

When the treatment is multi-category:

`which.treat` For which treatments or treatment combinations balance tables should be displayed. If a vector of length 1 is entered, all comparisons involving that treatment group will be displayed. If a vector of length 2 or more is entered, all comparisons involving treatments that both appear in the input will be displayed. For example, setting `which.treat = "A"` will display "A vs. B" and "A vs. C", while setting `which.treat = c("A", "B")` will only display "A vs. B". `.none` indicates no treatment comparisons will be displayed, and `.all` indicates all treatment comparisons will be displayed. Default is `.none`. See [bal.tab.multi\(\)](#).

`multi.summary` logical; whether to display the balance summary across all treatment pairs. This includes one row for each covariate with maximum balance statistic across all pairwise comparisons. Note that, if variance ratios or KS statistics are requested, the displayed values may not come from the same pairwise comparisons; that is, the greatest standardized mean difference and the greatest variance ratio may not come from the same comparison. Default is TRUE when `which.treat` is `.none` and FALSE otherwise. See [bal.tab.multi\(\)](#).

When clusters are present:

`which.cluster` For which clusters balance tables should be displayed. If `.all`, all clusters in cluster will be displayed. If `.none`, no clusters will be displayed. Otherwise, can be a vector of cluster names or numerical indices for which to display balance. Indices correspond to the alphabetical order of cluster names (or the order of cluster levels if a factor). Default is `.all`. See [class-bal.tab.cluster](#).

`cluster.summary` logical; whether to display the balance summary across clusters. Default is TRUE when `which.cluster` is `.none` and FALSE otherwise (note the default for `which.cluster` is `.all`). See [class-bal.tab.cluster](#).

`cluster.fun` Which function is used in the across-cluster summary to combine results across clusters. Can be "min", "mean", or "max". For example, if `cluster.fun = "mean"` the mean balance statistic across clusters will be displayed. The default when `abs = FALSE` in the `bal.tab()` call is to display all three. The default when `abs = TRUE` in the `bal.tab()` call is to display just the mean and maximum absolute balance statistic. See [class-bal.tab.cluster](#).

When multiple imputations are present:

`which.imp` For which imputations balance tables should be displayed. If `.all`, all imputations in `imp` will be displayed. If `.none`, no imputations will be displayed. Otherwise, can be a vector of imputation indices for which to display balance. Default is `.none`. See [class-bal.tab.imp](#).

`imp.summary` logical; whether to display the balance summary across imputations. Default is TRUE when `which.imp` is `.none` and FALSE otherwise. See [class-bal.tab.imp](#).

`imp.fun` Which function is used in the across-imputation summary to combine results across imputations. Can be "min", "mean", or "max". For example, if `imp.fun = "mean"` the mean balance statistic across imputations will be displayed. The default when `abs = FALSE` in the `bal.tab()` call is to display all three. The default when `abs = TRUE` in the `bal.tab()` call is to display just the mean and maximum absolute balance statistic. See [class-bal.tab.imp](#).

When the treatment is longitudinal:

`which.time` For which time points balance tables should be displayed. If `.all`, all time points will be displayed. If `.none`, no time points will be displayed. Otherwise, can be a vector of treatment names or indices for which to display balance. Default is `.none`. See [class-bal.tab.msm](#).

`msm.summary` logical; whether to display the balance summary across time points. Default is TRUE when `which.time` is `.none` and FALSE otherwise. See [class-bal.tab.msm](#).

Variable names using `var.names`

Variables are displayed as they are named in the output of `bal.tab()`, which may not be how they should be named in a report; `var.names` supplies alternate names for them. It is taken by `bal.tab()` and by everything that displays a `bal.tab` object (i.e., `print()`, `format()`, `as.data.frame()`, and `love.plot()`) and means the same thing in all of them. What is given to `bal.tab()` applies to everything that displays the object afterwards, so a set of names need be settled on only once; what is given to one of the others adds to that and replaces any entry it names.

Only the names on display change. The variables are still stored, and still selected, under their own names, so `var.names` never has to be undone to work with the object. Because the names on display are what is being changed, it is important to know them before specifying alternate ones: they may differ from the names in the original data. `var.names()` extracts them, optionally to a CSV file to edit and read back in; when a `var.names` has already been applied, it reports the names that produced, so a set arrived at once can be edited rather than written out again.

`var.names` pairs each old name with the new name to display in its place, in any of three structures:

1. a vector or list of new variable names, with the names of the values the old variable names

2. a data frame with exactly one column containing the new variable names and the row names containing the old variable names
3. a data frame with two columns, the first (or the one named "old") containing the old variable names and the second (or the one named "new") containing the new variable names.

A variable not named in it keeps the name it has.

Names are matched either against the whole name string or against the components a name is built from. For example, if a factor variable "X" with levels "a", "b", and "c" is displayed, the variables "X_a", "X_b", and "X_c" will be displayed. You can enter replacement names for all three variables individually, or you can simply specify a replacement name for "X", and "X" will be replaced by the given name everywhere it appears, including not just factor expansions but also polynomials and the interactions produced by `int = TRUE`. In an interaction with another variable, say "Y", there are several ways to replace the name of the interaction term "X_a * Y". If the entire string ("X_a * Y") is included in `var.names`, the entire string will be replaced. If "X_a" is included, only it will be replaced (and it will be replaced everywhere else it appears). If "X" is included, only it will be replaced (and it will be replaced everywhere else it appears). See the example at [var.names\(\)](#).

Two variables cannot be given the same name, which would leave them indistinguishable in a table and sitting on top of each other in a plot; doing so is an error.

The separators:

`factor_sep` and `int_sep` are taken in the same places and honored the same way: given to `bal.tab()` they apply to everything that displays the object afterwards, and given to one of the display functions they apply there. Each is a single string rather than a set of entries, so one given later simply replaces the one in force, as in `love.plot(b, factor_sep = ":")`.

A `bal.tab` object records what each name is made of, so the separators can be changed after the fact without the names having to be parsed apart, and a replacement given for a variable applies whichever separators are on display. The names a covariate is stored under, and by which `var.names` keys its replacements, are those the object was built with, and are not affected.

Setting options globally

In addition to being able to be specified as arguments, if you find you frequently set a display option to something other than its default, you can set that as a global option (for the present R session) using [set.cobalt.options\(\)](#) and retrieve it using [get.cobalt.options\(\)](#). Note that global options cannot be set for `which.subclass`, `which.cluster`, `which.imp`, `which.treat`, `which.time`, or `var.names`.

Note

When calling `bal.tab()` using [do.call\(\)](#), if you are using `.all` or `.none` as inputs to arguments, you need to use [alist\(\)](#) rather than [list\(\)](#) to group the arguments. For example, `do.call(bal.tab, list(., which.cluster = .none))` will produce an error, but `do.call(bal.tab, alist(., which.cluster = .none))` should work correctly.

See Also

[bal.tab\(\)](#), [print.bal.tab\(\)](#)

extract.bal.tab	<i>Extract a Balance Table for Further Use</i>
-----------------	--

Description

`as.data.frame()` extracts the balance statistics computed by `bal.tab()` as a tidy data frame, one row per covariate, sample, and statistic. `format()` returns the balance table exactly as `print.bal.tab()` displays it, as a data frame of formatted strings, ready to be passed to a table-rendering function.

Together they are meant to remove the need to pick apart a `bal.tab` object by hand when reporting balance in a document.

Usage

```
## S3 method for class 'bal.tab'
as.data.frame(
  x,
  row.names = NULL,
  optional = FALSE,
  ...,
  var.names = NULL,
  wide = FALSE
)

## S3 method for class 'bal.tab'
format(
  x,
  ...,
  var.names = NULL,
  digits = max(3L, getOption("digits") - 3L),
  component = "balance"
)
```

Arguments

<code>x</code>	a <code>bal.tab</code> object; the output of a call to <code>bal.tab()</code> .
<code>row.names, optional</code>	ignored; present for consistency with the <code>as.data.frame()</code> generic.
<code>...</code>	arguments passed to <code>print.bal.tab()</code> to control which statistics, samples, and covariates are included, e.g. <code>stats</code> , <code>disp</code> , <code>un</code> , <code>imbalanced.only</code> , or <code>disp.thresholds</code> . The <code>.all</code> and <code>.none</code> shorthands are accepted as they are by <code>print()</code> . Arguments that were not computed in the original call to <code>bal.tab()</code> cannot be requested here, for the same reason they cannot be requested in <code>print()</code> .
<code>var.names</code>	an optional object providing alternate names for the variables, which will otherwise be returned as they are stored. Entries given here add to those given in the original call to <code>bal.tab()</code> and replace any entry they name. See display-options for how to specify it.

wide	logical; for <code>as.data.frame()</code> , whether to return the table in the layout <code>print()</code> uses, with one column per sample and statistic, rather than the default tidy layout. Default is <code>FALSE</code> .
digits	for <code>format()</code> , the number of significant digits to display. Default is the same as for <code>print()</code> .
component	for <code>format()</code> , which table to return: "balance" (the default) for the balance table, or "observations" for the sample size table.

Details

Both functions accept every argument `print()` accepts, and resolve them the same way, so the same warnings are raised when a requested value was not computed because `quick = TRUE` in the original call to `bal.tab()`.

The variable column and the row names carry the covariates' display names, so `var.names` – given here or in the original call to `bal.tab()` – changes them. What the covariates are stored under is unaffected. See [display-options](#).

`as.data.frame()` returns the balance statistics themselves, from each innermost balance table. It does not return the summaries across clusters, imputations, treatment pairs, or time points, which are aggregates of those statistics; `format()` returns the summary when that is what `print()` displays.

Value

`as.data.frame()` returns a data frame with one row per covariate, sample, and statistic, and these columns:

`variable` the covariate name, as it appears in the row names of the balance table.

`type` the covariate type: "Binary", "Contin.", or "Distance".

`sample` "Unadjusted", or the name of the set of weights.

`stat` the name of the statistic, using the same names as `bal.tab()`'s `stats` argument (e.g., "mean.diffs"), with "mean" and "sd" for the distribution summary statistics.

`group` for "mean" and "sd", the name of the treatment group the value describes, as `bal.tab()` names it: the two group names for a binary treatment, the level for a multi-category one, "Uncensored" or "Full" for a censoring indicator, and "All" for a continuous treatment, which has no groups, or for the full sample when `pairwise = FALSE`. NA for a statistic that contrasts two groups, which belongs to neither of them.

`estimate` the value of the statistic.

`threshold` the balance verdict, e.g. "Balanced, <0.1", when a threshold was requested for that statistic; NA for a row it does not cover, such as a mean.

`threshold.value` the numeric threshold, on the same rows.

The two threshold columns are present only when a threshold is on display for at least one statistic; when none is, they would be empty throughout and are omitted. The rest of the columns are always present.

When the data are segmented – by cluster, imputation, treatment pair, time point, or subclass – one further column per level of segmentation identifies it. Segmentation is always a column, never a nested list, so the result is a single rectangle whatever the shape of the input.

A multi-category treatment is reported one pair of groups at a time, but a mean or a standard deviation belongs to a group rather than to a comparison, and is the same in every pair that group appears in. Such a row therefore appears once, with pair set to NA; only the statistics that contrast two groups carry a pair. The same applies to the full sample's own means when pairwise = FALSE, which would otherwise be repeated against every group.

With wide = TRUE, the columns are those print() displays, with the covariate names moved from the row names into a variable column and any segmentation columns retained.

format() returns a data frame of character vectors with the covariate names as row names, formatted exactly as print() displays them: rounded to digits, padded to a common number of decimal places, with NA shown as ".".

See Also

- [bal.tab\(\)](#)
- [print.bal.tab\(\)](#)
- [love.plot\(\)](#) for a graphical alternative
- [display-options](#) for var.names

Examples

```
data("lalonge", package = "cobalt")

b <- bal.tab(treat ~ age + educ + race + re74, data = lalonge,
             s.d.denom = "pooled", stats = c("m", "ks"),
             thresholds = c(m = .1), un = TRUE)

#Tidy: one row per covariate, sample, and statistic
head(as.data.frame(b))

#The layout print() shows
as.data.frame(b, wide = TRUE)

#Ready for knitr::kable() or any other table renderer
format(b)

format(b, component = "observations")

#print()'s arguments work here too
as.data.frame(b, stats = "ks", un = FALSE)
```

Description

`f.build()` returns a [formula](#) of the form $y \sim x_1 + x_2 + \dots$ from a data frame input. It can be much quicker to use `f.build()` than to hand-write the precise formula, which may contain errors. It can be used in place of a formula in, for example, `glm()`, `matchit()`, or `bal.tab()`. It provides similar functionality to [reformulate\(\)](#).

Usage

```
f.build(y = NULL, rhs = NULL)
```

Arguments

<code>y</code>	the quoted name of the response (left hand side) variable in the formula. Only one variable is supported. If missing, <code>NULL</code> , or the empty string (<code>""</code>), the formula will have no response variable. If <code>rhs</code> is not supplied, <code>y</code> will replace <code>rhs</code> and <code>y</code> will be set to <code>""</code> .
<code>rhs</code>	a data frame whose variable names will be the terms on the right hand side of the formula, or a character vector whose values will be the terms on the right hand side of the formula. If missing, the argument to <code>y</code> will replace <code>rhs</code> and <code>y</code> will be set to <code>""</code> ; in essence, <code>f.build("x")</code> is the same as <code>f.build("", "x")</code> , both producing $\sim x$.

Value

a formula object.

See Also

[reformulate\(\)](#)

Examples

```
data(lalonde)
covs <- subset(lalonde, select = -c(treat, re78))
lm(f.build("treat", covs), data = lalonde)
```

get.w

Extract Weights from Preprocessing Objects

Description

Extracts weights from the outputs of preprocessing functions.

Usage

```
get.w(x, ...)  
  
## S3 method for class 'matchit'  
get.w(x, ...)  
  
## S3 method for class 'ps'  
get.w(x, stop.method = NULL, estimand, s.weights = FALSE, ...)  
  
## S3 method for class 'mnp'  
get.w(x, stop.method = NULL, s.weights = FALSE, ...)  
  
## S3 method for class 'ps.cont'  
get.w(x, s.weights = FALSE, ...)  
  
## S3 method for class 'iptw'  
get.w(x, stop.method = NULL, s.weights = FALSE, ...)  
  
## S3 method for class 'Match'  
get.w(x, ...)  
  
## S3 method for class 'CBPS'  
get.w(x, estimand, ...)  
  
## S3 method for class 'CBMSM'  
get.w(x, ...)  
  
## S3 method for class 'ebalance'  
get.w(x, treat, ...)  
  
## S3 method for class 'optmatch'  
get.w(x, estimand, ...)  
  
## S3 method for class 'cem.match'  
get.w(x, estimand, ...)  
  
## S3 method for class 'weightit'  
get.w(x, s.weights = FALSE, ...)  
  
## S3 method for class 'designmatch'  
get.w(x, treat, estimand, ...)  
  
## S3 method for class 'mimids'  
get.w(x, ...)  
  
## S3 method for class 'wimids'  
get.w(x, ...)
```

```
## S3 method for class 'sbwcau'
get.w(x, ...)
```

Arguments

<code>x</code>	output from the corresponding preprocessing packages.
<code>...</code>	arguments passed to other methods.
<code>stop.method</code>	the name of the stop method used in the original call to <code>ps()</code> or <code>mnps()</code> in twang , e.g., "es.mean". If empty, will return weights from all stop method available into a data frame. Abbreviations allowed.
<code>estimand</code>	if weights are computed using the propensity score (i.e., for the <code>ps</code> and <code>CBPS</code> methods), which estimand to use to compute the weights. If "ATE", weights will be computed as $1/ps$ for the treated group and $1/(1-ps)$ for the control group. If "ATT", weights will be computed as 1 for the treated group and $ps/(1-ps)$ for the control group. If not specified, <code>get.w()</code> will try to figure out which estimand is desired based on the object. If weights are computed using subclasses/matching strata (i.e., for the <code>cem</code> and <code>designmatch</code> methods), which estimand to use to compute the weights. First, a subclass propensity score is computed as the proportion of treated units in each subclass, and the one of the formulas above will be used based on the estimand requested. If not specified, "ATT" is assumed.
<code>s.weights</code>	whether the sampling weights included in the original call to the fitting function should be included in the weights. If TRUE, the returned weights will be the product of the balancing weights estimated by the fitting function and the sampling weights. If FALSE, only the balancing weights will be returned.
<code>treat</code>	a vector of treatment status for each unit. This is required for methods that include <code>treat</code> as an argument. The treatment variable that was used in the original preprocessing function call should be used.

Details

The output of `get.w()` can be used in calls to the formula and data frame methods of `bal.tab()` (see example below). In this way, the output of multiple preprocessing packages can be viewed simultaneously and compared. The weights can also be used in `weights` statements in regression methods to compute weighted effects.

twang has a function called `get.weights()` that performs the same function on `ps` objects but offers slightly finer control. Note that the weights generated by `get.w()` for `ps` objects do not include sampling weights by default.

When sampling weights are used with `CBPS()` in **CBPS**, the returned weights will already have the sampling weights incorporated. To retrieve the balancing weights on their own, divide the returned weights by the original sampling weights. For other packages, the balancing weights are returned separately unless `s.weights = TRUE`, which means they must be multiplied by the sampling weights for effect estimation.

When `Match()` in **Matching** is used with `CommonSupport = TRUE`, the returned weights will be incorrect. This option is not recommended by the package authors.

Value

A vector or data frame of weights for each unit. These may be matching weights or balancing weights.

Examples

```
data("lalonge", package = "cobalt")

m.out <- MatchIt::matchit(treat ~ age + educ + race,
  data = lalonge,
  estimand = "ATT")

w.out <- WeightIt::weightit(treat ~ age + educ + race,
  data = lalonge,
  estimand = "ATT")

bal.tab(treat ~ age + educ + race, data = lalonge,
  weights = data.frame(matched = get.w(m.out),
    weighted = get.w(w.out)),
  method = c("matching", "weighting"),
  estimand = "ATT")
```

lalonge

Data from National Supported Work Demonstration and PSID, as analyzed by Dehejia and Wahba (1999).

Description

This is a subsample of the data from the treated group in the National Supported Work Demonstration (NSW) and the comparison sample from the Population Survey of Income Dynamics (PSID). This data was previously analyzed extensively by Lalonde (1986) and Dehejia and Wahba (1999).

Usage

```
lalonge

lalonge_mis
```

Format

A data frame with 614 observations (185 treated, 429 control). There are 9 variables measured for each individual.

- "treat" is the treatment assignment (1=treated, 0=control).
- "age" is age in years.
- "educ" is education in number of years of schooling.

- "race" is the individual's race/ethnicity, (Black, Hispanic, or White). Note some other versions of this dataset use indicator variables black and hispan instead of a single race variable.
- "married" is an indicator for married (1=married, 0=not married).
- "nodegree" is an indicator for whether the individual has a high school degree (1=no degree, 0=degree).
- "re74" is income in 1974, in U.S. dollars.
- "re75" is income in 1975, in U.S. dollars.
- "re78" is income in 1978, in U.S. dollars.

"treat" is the treatment variable, "re78" is the outcome, and the others are pre-treatment covariates. An object of class `data.frame` with 614 rows and 9 columns.

Details

`lalonds_mis` is the same dataset with missing values in three of the covariates.

References

Lalonde, R. (1986). Evaluating the econometric evaluations of training programs with experimental data. *American Economic Review* 76: 604-620.

Dehejia, R.H. and Wahba, S. (1999). Causal Effects in Nonexperimental Studies: Re-Evaluating the Evaluation of Training Programs. *Journal of the American Statistical Association* 94: 1053-1062.

love.plot

Display Balance Statistics in a Love Plot

Description

Generates a "Love" plot graphically displaying covariate balance before and after adjusting. Options are available for producing publication-ready plots. Detailed examples are available in `vignette("love.plot")`.

Usage

```
love.plot(
  x,
  stats,
  abs,
  agg.fun = NULL,
  var.order = NULL,
  drop.missing = TRUE,
  drop.distance = FALSE,
  thresholds = NULL,
  line = FALSE,
  stars = "none",
  grid = FALSE,
  limits = NULL,
```

```

    colors = NULL,
    shapes = NULL,
    alpha = 1,
    size = 3,
    wrap = 30,
    var.names = NULL,
    title,
    sample.names,
    labels = FALSE,
    position = "right",
    themes = NULL,
    ...
)

```

Arguments

<code>x</code>	the valid input to a call to <code>bal.tab()</code> (e.g., the output of a preprocessing function). Other arguments that would be supplied to <code>bal.tab()</code> can be entered with <code>...</code> . Can also be a <code>bal.tab</code> object, i.e., the output of a call to <code>bal.tab()</code> . See Examples. If <code>x</code> is not a <code>bal.tab</code> object, <code>love.plot()</code> calls <code>bal.tab()</code> with the arguments supplied.
<code>stats</code>	character; which statistic(s) should be reported. See <code>stats</code> for allowable options. For binary and multi-category treatments, "mean.diffs" (i.e., mean differences) is the default. For continuous treatments, "correlations" (i.e., treatment-covariate Pearson correlations) is the default. Multiple options are allowed.
<code>abs</code>	logical; whether to present the statistic in absolute value or not. For variance ratios, this will force all ratios to be greater than or equal to 1. If <code>x</code> is a <code>bal.tab</code> object, <code>love.plot()</code> might ignore <code>abs</code> depending on the original <code>bal.tab()</code> call. If unspecified, uses whatever was used in the call to <code>bal.tab()</code> .
<code>agg.fun</code>	if balance is to be displayed across clusters or imputations rather than within a single cluster or imputation, which summarizing function ("mean", "max", or "range") of the balance statistics should be used. If "range" is entered, <code>love.plot()</code> will display a line from the min to the max with a point at the mean for each covariate. Abbreviations allowed; "range" is default. Remember to set <code>which.<ARG> = .none</code> (where <code><ARG></code> is the grouping argument, such as <code>cluster</code> or <code>imp</code>) to use <code>agg.fun</code> . See Details.
<code>var.order</code>	a character or <code>love.plot</code> object; how to order the variables in the plot. See Details.
<code>drop.missing</code>	logical; whether to drop rows for variables for which the statistic has a value of NA, for example, variance ratios for binary variables. If FALSE, there will be rows for these variables but no points representing their value. Default is TRUE, so that variables with missing balance statistics are absent. When multiple stats are requested, only variables with NAs for all stats will be dropped if <code>drop.missing = TRUE</code> . This argument used to be called <code>no.missing</code> , and that name still works (but has been deprecated).
<code>drop.distance</code>	logical; whether to ignore the distance measure (if there are any) in plotting.

thresholds	numeric; an optional value to be used as a threshold marker in the plot. Should be a named vector where each name corresponds to the statistic for which the threshold is to be applied. See example at stats . If <code>x</code> is a <code>bal.tab</code> object and a threshold was set in it (e.g., with <code>thresholds</code>), its threshold will be used unless overridden using the <code>threshold</code> argument in <code>love.plot()</code> .
line	logical; whether to display a line connecting the points for each sample.
stars	when mean differences are to be displayed, which variable names should have a star (i.e., an asterisk) next to them. Allowable values are "none", "std" (for variables with mean differences that have been standardized), or "raw" (for variables with mean differences that have not been standardized). If "raw", the x-axis title will be "Standardized Mean Differences". Otherwise, it will be "Mean Differences". Ignored when mean difference are not displayed. See Details for an explanation of the purpose of this option.
grid	logical; whether gridlines should be shown on the plot. Default is FALSE.
limits	numeric; the bounds for the x-axis of the plot. Must a (named) list of vectors of length 2 in ascending order, one for each value of <code>stats</code> that is to have limits; e.g., <code>list(m = c(-.2, .2))</code> . If values exceed the limits, they will be plotted at the edge.
colors	the colors of the points on the plot. See 'Color Specification' at graphics::par() or the ggplot2 aesthetic specifications vignette (<code>vignette("ggplot2-specs")</code>). The first value corresponds to the color for the unadjusted sample, and the second color to the adjusted sample. If only one is specified, it will apply to both. Defaults to the default ggplot2 colors.
shapes	the shapes of the points on the plot. Must be one or two numbers between 1 and 25 or the name of a valid shape. See the ggplot2 aesthetic specifications vignette (<code>vignette("ggplot2-specs")</code>) for valid options. Values 15 to 25 are recommended. The first value corresponds to the shape for the unadjusted sample, and the second color to the adjusted sample. If only one is specified, it will apply to both. Defaults to 19 ("circle filled").
alpha	numeric; the transparency of the points. See ggplot2::scale_alpha() .
size	numeric; the size of the points on the plot. Defaults to 3. In previous versions, the size was scaled by a factor of 3. Now size corresponds directly to the size aesthetic in ggplot2::geom_point() .
wrap	numeric; the number of characters at which to wrap axis labels to the next line. Defaults to 30. Decrease this if the axis labels are excessively long.
var.names	an optional object providing alternate names for the variables in the plot, which will otherwise be the variable names as they are stored. This may be useful when variables have ugly names. See display-options for how to specify <code>var.names</code> . Names given in the original call to <code>bal.tab()</code> are used as well; those given here add to them and replace any entry they name. <code>var.names()</code> can be a useful tool for extracting and editing the names from the <code>bal.tab</code> object.
title	character; the title of the plot.
sample.names	character; new names to be given to the samples (i.e., in place of "Unadjusted" and "Adjusted"). For example, when matching it used, it may be useful to enter <code>c("Unmatched", "Matched")</code> .

labels	logical or character; labels to give the plots when multiple stats are requested. If TRUE, the labels will be capital letters. Otherwise, must be a string with the same length as stats. This can be useful when the plots are to be used in an article.
position	the position of the legend. When stats has length 1, this can be any value that would be appropriate as an argument to <code>legend.position</code> in <code>ggplot2::theme()</code> . When stat has length greater than 1, can be one of "none", "left", "right", "bottom", or "top".
themes	an optional list of theme objects to append to each individual plot. Each entry should be the output of a call to <code>ggplot2::theme()</code> in <code>ggplot2</code> . This is a way to customize the individual plots when multiple stats are requested since the final output is not a manipulable ggplot object. It can be used with length-1 stats, but it probably makes more sense to just add the <code>theme()</code> call after <code>love.plot()</code> .
...	additional arguments passed to <code>bal.tab()</code> or options for display of the plot. The following related arguments are currently accepted: <code>use.grid</code> whether to use <code>gridExtra::arrangeGrob()</code> in <code>gridExtra</code> to make the plot when stats has length 1. See section Value. <code>disp.subclass</code> whether to display individual subclasses if subclassification is used. Overrides the <code>disp.subclass</code> option in the original <code>bal.tab()</code> call if <code>x</code> is a <code>bal.tab</code> object. <code>star_char</code> character; when stars are used, the character that should be the "star" next to the starred variables. The default is <code>"*"</code>. <code>"†"</code> or <code>"\u2020"</code> (i.e., dagger) might be appealing as well. Additionally, any of the <code>which.</code> arguments used with clustered or multiply imputed data or longitudinal or multi-category treatments can be specified to display balance on selected groupings. Set to <code>.none</code> to aggregate across groups (in which <code>agg.fun</code> comes into effect) and set to <code>.all</code> to view all groups. See display-options for options, and see <code>vignette("segmented-data")</code> for details and examples.

Details

`love.plot()` can be used with clusters, imputations, and multi-category and longitudinal treatments in addition to the standard case. Setting the corresponding `which.` argument to `.none` will aggregate across that dimension. When aggregating, an argument should be specified to `agg.fun` referring to whether the mean, minimum ("min"), or maximum ("max") balance statistic or range ("range", the default) of balance statistics for each covariate should be presented in the plot. See `vignette("segmented-data")` for examples.

With subclasses, balance will be displayed for the unadjusted sample and the aggregated subclassified sample. If `disp.subclass` is TRUE, each subclass will be displayed additionally as a number on the plot.

Variable order using `var.order`:

The order that the variables are presented in depends on the argument to `var.order`. If NULL, the default, they will be displayed in the same order as in the call to `bal.tab()`, which is the order of the underlying data set. If "alphabetical", they will be displayed in alphabetical order. If

"unadjusted", they will be ordered by the balance statistic of the unadjusted sample. To order by the values of the adjusted sample, "adjusted" can be supplied if only one set of weights (or subclasses) are specified; otherwise, the name of the set of weights should be specified.

If multiple stats are requested, the order will be determined by the first entry to stats; for example, if both "mean.diffs" and "ks.statistics" are requested and var.order = "unadjusted", the variables will be displayed in order of the unadjusted mean differences for both plots. If multiple plots are produced simultaneously (i.e., for individual clusters or imputations), var.order can only be NULL or "alphabetical".

If a love.plot object is supplied, the plot being drawn will use the variable order in the supplied love.plot object. This can be useful when making more than one plot and the variable order should be the same across plots.

Variable names using var.names:

The variables are presented as they are named in the output of the call to bal.tab(), which may not be how they should be named in a report. var.names supplies alternate names for them; it is documented at [display-options](#), along with the several structures it accepts and how a name given for a base variable reaches the factor levels, polynomials, and interactions it appears in. It is one of the display options, so it can be given to bal.tab() instead, in which case the plot uses those names without being told them again.

The factor_sep and int_sep display options are taken here too, so the separators a plot displays can be chosen without recomputing the object; see [display-options](#).

Stars and the x-axis label with mean differences:

When mean differences are to be displayed, love.plot() attempts to figure out the appropriate label for the x-axis. If all mean differences are standardized, the x-axis label will be "Standardized Mean Differences". If all mean differences are raw (i.e., unstandardized), the x-axis label will be "Mean Differences". Otherwise, love.plot() turns to the stars argument. If "raw", the x-axis label will be "Standardized Mean Differences" (i.e., because un-starred variables have standardized mean differences displayed). If "std", the x-axis label will be "Mean Differences" (i.e., because un-starred variables have raw mean differences displayed). If "none", the x-axis label will be "Mean Differences" and a warning will be issued recommending the use of stars.

The default is to display standardized mean differences for continuous variables, raw mean differences for binary variables, and no stars, so this warning will be issued in most default uses of love.plot(). The purpose of this is to correct behavior of previous versions of **cobalt** in which the default x-axis label was "Mean Differences", even when standardized mean differences were displayed, yielding a potentially misleading plot. This warning requires the user to think about what values are being displayed. The idea of using stars is that the user can, in a caption for the plot, explain that variables with an asterisk have standardized (or raw) mean differences display, in contrast to un-starred variables.

Value

When only one type of balance statistic is requested, the returned object is a standard ggplot object that can be manipulated using **ggplot2** syntax. This facilitates changing fonts, background colors, and features of the legend outside of what love.plot() provides automatically.

When more than one type of balance statistic is requested, the plot is constructed using [gridExtra::arrangeGrob\(\)](#), which arranges multiple plots and their shared legend into one plot. Because the output of arrangeGrob()

is a `gtable` object, its features cannot be manipulated in the standard way. Use the `themes` argument to change theme elements of the component plots. The original plots are stored in the `"plots"` attribute of the output object.

Note

`love.plot` can also be called by using `plot()` or `autoplot()` on a `bal.tab` object. If used in this way, some messages may appear twice. It is recommended that you just use `love.plot()` instead.

See Also

`bal.tab()`, `vignette("love.plot")`
`display-options` for `var.names`

Examples

```
data("lalonge", package = "cobalt")

## Propensity score weighting
library(WeightIt)
w.out1 <- weightit(treat ~ age + educ + race + married +
                  nodegree + re74 + re75,
                  data = lalonge)

love.plot(w.out1, thresholds = c(m = .1),
          var.order = "unadjusted")

## Using alternate variable names
v <- data.frame(old = c("age", "educ", "race_black", "race_hispan",
                       "race_white", "married", "nodegree", "re74",
                       "re75", "distance"),
               new = c("Age", "Years of Education", "Black",
                      "Hispanic", "White", "Married", "No Degree",
                      "Earnings 1974", "Earnings 1975",
                      "Propensity Score"))

love.plot(w.out1, stats = "m", threshold = .1,
          var.order = "unadjusted", var.names = v)

#Using multiple stats
love.plot(w.out1, stats = c("m", "ks"),
          thresholds = c(m = .1, ks = .05),
          var.order = "unadjusted", var.names = v, stars = "raw",
          position = "bottom", wrap = 20)

#Changing visual elements
love.plot(w.out1, thresholds = c(m = .1),
          var.order = "unadjusted", var.names = v, abs = TRUE,
          shapes = c("triangle filled", "circle"),
          colors = c("red", "blue"), line = TRUE,
          grid = FALSE, sample.names = c("Original", "Weighted"),
          stars = "raw", position = "top")
```

print.bal.tab

Print Results of a Call to bal.tab()

Description

Prints bal.tab() output in a clean way. Provides options for printing.

Usage

```
## S3 method for class 'bal.tab'
print(
  x,
  imbalanced.only,
  un,
  disp.bal.tab,
  disp.call,
  stats,
  disp.thresholds,
  disp,
  which.subclass,
  subclass.summary,
  which.imp,
  imp.summary,
  imp.fun,
  which.treat,
  multi.summary,
  which.time,
  msm.summary,
  which.cluster,
  cluster.summary,
  cluster.fun,
  var.names = NULL,
  digits = max(3L, getOption("digits") - 3),
  ...
)
```

Arguments

x a bal.tab object; the output of a call to [bal.tab\(\)](#).

imbalanced.only logical; whether to display only the covariates that failed to meet at least one of balance thresholds. Depends only on whether threshold were initial set in the call to bal.tab() and not on any arguments to print() (except disp.bal.tab).

un	logical; whether to display balance values for the unadjusted sample. Ignored (and set to TRUE) if no conditioning was performed.
disp.bal.tab	logical; whether to display the table of balance statistics. If FALSE, only other values (e.g., the call, sample sizes, balance tallies, and maximum imbalances) will be presented.
disp.call	logical; whether to display the function call for the input object, if any.
stats	character; which statistic(s) should be reported. For binary or multi-category treatments, the options are "mean.diffs" for mean differences (standardized or not according the selected bal.tab() options), "variance.ratios" for variance ratios, and "ks.statistics" for Kolmogorov-Smirnov statistics. "mean.diffs" is the default. For continuous treatments, the only option is "correlations" for treatment-covariate correlations. Multiple options are allowed. Abbreviations allowed. Statistics that weren't requested in the original call to bal.tab() cannot be requested with print() unless quick = FALSE in the original call.
disp.thresholds	logical; whether to display thresholds for each statistic for which thresholds were originally requested in the call to bal.tab(). Should be a named logical vector with names corresponding to the thresholds. For example, if thresholds for mean differences were requested in bal.tab(), set disp.thresholds = c(m = FALSE) to prevent them from being printed. If a statistic was prevented from being displayed by another argument to print(), the thresholds will not be displayed.
disp	character; which distribution summary statistics to display. Allowable options include "means" and "sds". Statistics that weren't requested in the original call to bal.tab() cannot be requested with print() unless quick = FALSE in the original call.
which.subclass	when used with subclassification, which subclass(es) to display. If NULL, all subclasses will be displayed. If NA, no subclasses will be displayed. Otherwise, can be a vector of subclass indices for which to display balance. To display the subclasses requested in the original call to bal.tab(), omit this argument. See class-bal.tab.subclass for details.
subclass.summary	logical; when used with subclassification, whether to display the subclass balance summary table. If which.subclass is NA, subclass.summary will be set to TRUE. See class-bal.tab.subclass for details.
which.imp	when used with multiply imputed data, which imputation(s) to display. If NULL, all imputations will be displayed. If NA, no imputations will be displayed. Otherwise, can be a vector of imputations numbers for which to display balance. To display the imputations requested in the original call to bal.tab(), omit this argument. See class-bal.tab.imp for details.
imp.summary	logical; when used with multiply imputed data, whether to display the imputation summary table. If which.imp is NA, imp.summary will be set to TRUE. See class-bal.tab.imp for details.
imp.fun	character; when used with multiply imputed data, a character vector of functions of balance statistics to display when displaying balance across imputations. Can be "mean", "min", or "max". More than one are allowed. See class-bal.tab.imp for details.

<code>which.treat</code>	when used with multi-category treatments, which treatments to display. See bal.tab.multi() for details.
<code>multi.summary</code>	logical; when used with multi-category treatments, whether to display the balance summary table across pairwise comparisons. See bal.tab.multi() for details.
<code>which.time</code>	when used with longitudinal treatments, which time periods to display if longitudinal treatments are used. See class-bal.tab.msm for details.
<code>msm.summary</code>	logical; when used with longitudinal treatments, whether to display the balance summary table across time periods. See class-bal.tab.msm for details.
<code>which.cluster</code>	when used with clustered data, which cluster(s) to display. If NULL, all clusters will be displayed. If NA, no clusters will be displayed. Otherwise, can be a vector of cluster names or numerical indices for which to display balance. Indices correspond to the alphabetical order of cluster names. To display the clusters requested in the original call to <code>bal.tab()</code> , omit this argument. See class-bal.tab.cluster for details.
<code>cluster.summary</code>	logical; when used with clustered data, whether to display the cluster summary table. If <code>which.cluster</code> is NA, <code>cluster.summary</code> will be set to TRUE. See class-bal.tab.cluster for details.
<code>cluster.fun</code>	character; when used with clustered data, a character vector of functions of balance statistics to display when displaying balance across clusters. Can be "mean", "min", or "max". More than one are allowed. See class-bal.tab.cluster for details.
<code>var.names</code>	an optional object providing alternate names for the variables, which will otherwise be displayed as they are stored. Entries given here add to those given in the original call to <code>bal.tab()</code> and replace any entry they name. See display-options for how to specify it.
<code>digits</code>	the number of digits to display.
<code>...</code>	further arguments passed to or from other methods.

Details

Simply calling `bal.tab()` will print its results, but it can be useful to store the results into an object and print them again later, possibly with different print options specified. The `print()` function automatically dispatches the correct method for the `bal.tab` object given.

Any parameter used in `bal.tab()` for calculations, such as `int`, `add1`, or `distance`, cannot be used with `print()`; only those parameters listed above, those that solely determine printing options, can be used. To change computation options, a new call to `bal.tab()` must be performed.

Prior versions of `print()` had separate methods for each `bal.tab` class. Now they are dispatched internally.

Value

`x`, invisibly. `print()` is called for the side effect of displaying the balance table and leaves the object as it was; the arguments above decide what is shown, not what is returned.

Note

Unless `quick = FALSE` in the original call to `bal.tab()` (which is not the default), some values may not be calculated, in which case using `print()` will not display these values even when requested. For example, if `stats = "m"` and `quick = TRUE` in the original call to `bal.tab()` (the default for both), setting `stats = "ks"` in `print()` will not print the KS statistics because they were not calculated.

See Also

[print\(\)](#), [bal.tab\(\)](#)

[display-options](#) for further information on some of these options.

Examples

```
data("lalonge", package = "cobalt")
library(WeightIt)

w.out <- weightit(treat ~ age + educ + married +
                 race + re74 + re75,
                 data = lalonge)

b <- bal.tab(w.out, stats = c("m", "v", "ks"),
             un = TRUE, v.threshold = 2)

print(b, un = FALSE, stats = c("m", "v"),
      disp.thresholds = c(v = FALSE))
```

set.cobalt.options	<i>Set and Get Options in cobalt</i>
--------------------	--------------------------------------

Description

Makes it easier to set **cobalt** options. `set.cobalt.options()` is essentially a wrapper for [options\(\)](#) but performs several checks, and `get.cobalt.options()` is essentially a wrapper for [getOption\(\)](#).

Usage

```
set.cobalt.options(..., default = FALSE)

get.cobalt.options(...)
```

Arguments

... For `set.cobalt.options()`, `bal.tab()` parameters and the values they should take. These should be the name of the parameter in `bal.tab()` without `"cobalt_"` preceding them. See examples. If any values are `NULL`, the corresponding options will be set back to their defaults.

For `get.cobalt.options()`, one or more strings containing the name of a parameter option to be retrieved. See examples. If empty, all available options and their values will be returned.

`default` if TRUE, sets all **cobalt** options not named in ... to their default values.

Details

When an option is set to NULL, it is set to its default value. The defaults are not displayed but are listed on the help pages where they appear. Most options correspond to display options, which can be accessed [here](#). Some others (e.g., continuous and binary) are described on the [bal.tab\(\)](#) help page.

Value

`set.cobalt.options()` is called for its side effect of setting options, and returns invisibly: the previous values of any options it returned to their defaults, as `options()` gives them, or NULL if it returned none.

`get.cobalt.options()` returns a named list with one entry per requested option, holding its current value, or NULL for an option that has not been set. Called with no arguments, it returns every available option.

See Also

- [options\(\)](#)
- [display-options](#) for some arguments that can be set via options.

Examples

```
# Set un to be TRUE to always display unadjusted
# balance measures and set binary to "std" to
# produce standardized mean differences for
# binary variables.

set.cobalt.options(un = TRUE, binary = "std")

# Note: the above is equivalent to:
# options(cobalt_un = TRUE, cobalt_binary = "std")
# but performs some additional checks

get.cobalt.options("un", "binary")

# Note: the above is equivalent to:
# getOption("cobalt_un")
# getOption("cobalt_binary")

# Return all cobalt options to their defaults

set.cobalt.options(default = TRUE)

# View all available options
```

```
get.cobalt.options()
```

splitfactor

Split and Unsplit Factors into Dummy Variables

Description

`splitfactor()` splits factor variables into dummy (0/1) variables. This can be useful when functions do not process factor variables well or require numeric matrices to operate. `unsplitfactor()` combines dummy variables into factor variables, undoing the operation of `splitfactor()`.

Usage

```
splitfactor(  
  data,  
  var.name,  
  drop.level = NULL,  
  drop.first = TRUE,  
  drop.singleton = FALSE,  
  drop.na = TRUE,  
  sep = "_",  
  replace = TRUE,  
  split.with = NULL,  
  check = TRUE  
)
```

```
unsplitfactor(  
  data,  
  var.name,  
  dropped.level = NULL,  
  dropped.na = TRUE,  
  sep = "_",  
  replace = TRUE  
)
```

Arguments

<code>data</code>	A data frame containing the variables to be split or unsplit. In <code>splitfactor()</code> , can be a factor variable to be split.
<code>var.name</code>	For <code>splitfactor()</code> , the names of the factor variables to split. If not specified, will split all factor variables in <code>data</code> . If <code>data</code> is a factor, the stem for each of the new variables to be created. For <code>unsplitfactor()</code> , the name of the previously split factor. If not specified and <code>data</code> is the output of a call to <code>splitfactor()</code> , all previously split variables will be unsplit.
<code>drop.level</code>	The name of a level of <code>var.name</code> for which to drop the dummy variable. Only works if there is only one variable to be split.

<code>drop.first</code>	Whether to drop the first dummy created for each factor. If "if2", will only drop the first category if the factor has exactly two levels. The default is to always drop the first dummy (TRUE).
<code>drop.singleton</code>	Whether to drop a factor variable if it only has one level.
<code>drop.na</code>	If NAs are present in the variable, how to handle them. If TRUE, no new dummy will be created for NA values, but all created dummies will have NA where the original variable was NA. If FALSE, NA will be treated like any other factor level, given its own column, and the other dummies will have a value of 0 where the original variable is NA.
<code>sep</code>	A character separating the the stem from the value of the variable for each dummy. For example, for "race_black", <code>sep = "_"</code> .
<code>replace</code>	Whether to replace the original variable(s) with the new variable(s) (TRUE) or the append the newly created variable(s) to the end of the data set (FALSE).
<code>split.with</code>	A list of vectors or factors with lengths equal to the number of columns of data that are to be split in the same way data is. See Details.
<code>check</code>	Whether to make sure the variables specified in <code>var.name</code> are actually factor (or character) variables. If splitting non-factor (or non-character) variables into dummies, set <code>check = FALSE</code> . If <code>check = FALSE</code> and <code>data</code> is a <code>data.frame</code> , an argument to <code>var.name</code> must be specified.
<code>dropped.level</code>	The value of each original factor variable whose dummy was dropped when the variable was split. If left empty and a dummy was dropped, the resulting factor will have the value NA instead of the dropped value. There should be one entry per variable to unsplit. If no dummy was dropped for a variable, an entry is still required, but it will be ignored.
<code>dropped.na</code>	If TRUE, will assume that NAs in the variables to be unsplit correspond to NA in the unsplit factor (i.e., that <code>drop.na = TRUE</code> was specified in <code>split.factor()</code>). If FALSE, will assume there is a dummy called "var.name_stem_NA" (e.g., "x_NA") that contains 1s where the unsplit factor should be NA (i.e., that <code>drop.na = FALSE</code> was specified in <code>split.factor()</code>). If NAs are stored in a different column with the same stem, e.g., "x_miss", that name (e.g., "miss") can be entered instead.

Details

If there are NAs in the variable to be split, the new variables created by `splitfactor()` will have NA where the original variable is NA.

When using `unsplitfactor()` on a data frame that was generated with `splitfactor()`, the arguments `dropped.na`, and `sep` are unnecessary.

If `split.with` is supplied, the elements will be split in the same way data is. For example, if data contained a 4-level factor that was to be split, the entries of `split.with` at the same index as the factor and would be duplicated so that resulting entries will have the same length as the number of columns of data after being split. The resulting values are stored in the "split.with" attribute of the output object. See Examples.

Value

For `splitfactor()`, a data frame containing the original data set with the newly created dummies.
 For `unsplitfactor()`, a data frame containing the original data set with the newly created factor variables.

See Also

`model.matrix()`

Examples

```
data("lalonge", package = "cobalt")

lalonge.split <- splitfactor(lalonge, "race",
                           replace = TRUE,
                           drop.first = TRUE)
# A data set with "race_hispan" and "race_white" instead
# of "race".

lalonge.unsplit <- unsplitfactor(lalonge.split, "race",
                                replace = TRUE,
                                dropped.level = "black")

all.equal(lalonge, lalonge.unsplit) #TRUE

# Demonstrating the use of split.with:
to.split <- list(letters[1:ncol(lalonge)],
                1:ncol(lalonge))

lalonge.split <- splitfactor(lalonge, split.with = to.split,
                            drop.first = FALSE)
attr(lalonge.split, "split.with")
```

var.names

Extract Variable Names from bal.tab Objects

Description

This function extracts variable names from a `bal.tab` object for use in specifying alternate variable names in `bal.tab()`, `print()`, `format()`, `as.data.frame()`, and `love.plot()`. Optionally, a file can be written for easy editing of names.

Usage

```
var.names(b, type, file = NULL, minimal = FALSE)
```

Arguments

<code>b</code>	a <code>bal.tab</code> object; the output of a call to <code>bal.tab()</code> .
<code>type</code>	the type of output desired. Can either be "df" for a data frame or "vec" for a named vector. See "Value". The default is "vec" unless <code>file</code> is not NULL.
<code>file</code>	optional; a file name to save the output if <code>type = "df"</code> . See <code>utils::write.csv()</code> , which <code>var.name()</code> calls. Must end in <code>.csv</code> .
<code>minimal</code>	whether the output should contain all variable names (i.e., all rows that appear the output of <code>bal.tab()</code>) or just the unique base variables. See "Details".

Details

The purpose of the function is to make supplying new variable names to the `var.names` argument easier. Rather than manually creating a vector or data frame with all the variable names that one desires to change, one can use `var.names()` to extract variable names from a `bal.tab` object and edit the output. Importantly, the output can be saved to a CSV file, which can be easily edited and read back into R, as demonstrated in the Example. See [display-options](#) for what `var.names` does and for the structures it accepts.

When `minimal = TRUE`, only a minimal set of variables will be output. For example, if the variables analyzed in `bal.tab()` are `age`, `race`, and `married`, and `int = TRUE` in `bal.tab()`, many variables will appear in the output, including expansions of the factor variables, the polynomial terms, and the interactions. Rather than renaming all of these variables individually, one can rename just the three base variables, and all variables that arise from them will be accordingly renamed. Setting `minimal = TRUE` requests only these base variables.

If a `var.names` was given in the call to `bal.tab()`, the names it produced are what is returned, so that a set of names arrived at once can be edited rather than written out again. Passing the result back to any of the functions that take `var.names` reproduces the names it came from, because the old names it is keyed by are the ones the variables are stored under, which `var.names` never changes.

Which names an edit then reaches follows from `minimal`, as it does for any `var.names`: an edit to a base variable in the minimal output reaches every name that variable appears in, while an edit to an entry of the full output changes only the name it is an entry for.

Value

If `type = "vec"`, a character vector with the variable names as the names and the names they are displayed under as the entries.

If `type = "df"`, a data frame with two columns called "old" and "new", the first with the variable names and the second with the names they are displayed under.

When no `var.names` has been applied to the object, every variable is displayed as it is stored and the two agree. When one has, the names it produced are given, so that they can be edited rather than written out again; see Details.

If `file` is not NULL, the output will be returned invisibly.

Note

Not all programs can properly read the Unicode characters for the polynomial terms when requested. These may appear strange in, e.g., Excel, but R will process the characters correctly.

See Also

[display-options](#) for var.names, the argument this function supplies

Examples

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

b1 <- bal.tab(treat ~ age + race + married, data = lalonde,
             int = TRUE)
v1 <- var.names(b1, type = "vec", minimal = TRUE)
v1["age"] <- "Age (Years)"
v1["race"] <- "Race/Eth"
v1["married"] <- "Married"

love.plot(b1, var.names = v1, factor_sep = ": ")

#The same names in the table, set once in `bal.tab()`
b1 <- bal.tab(treat ~ age + race + married, data = lalonde,
             int = TRUE, var.names = v1)
b1

#They come back out to be edited rather than rewritten
v1 <- var.names(b1, type = "vec", minimal = TRUE)
v1

v1["married"] <- "Married at baseline"
print(b1, var.names = v1)
## Not run:
b2 <- bal.tab(treat ~ age + race + married + educ + nodegree +
             re74 + re75 + I(re74==0) + I(re75==0),
             data = lalonde)
var.names(b2, file = "varnames.csv")

##Manually edit the CSV (e.g., in Excel), then save it.
v2 <- read.csv("varnames.csv")
love.plot(b2, var.names = v2)

## End(Not run)
```

Index

- * **datasets**
 - lalonge, 101
- .cens, 3
- .cens(), 16, 81, 82, 86, 88
- alist(), 94
- as.data.frame(), 93, 95, 115
- as.data.frame.bal.tab
 - (extract.bal.tab), 95
- available.stats (bal.compute), 4
- bal.compute, 4
- bal.compute(), 80
- bal.init (bal.compute), 4
- bal.plot, 8
- bal.plot(), 81
- bal.tab, 12
- bal.tab(), 3, 4, 7, 9, 11, 21, 24, 25, 29, 33, 36, 41, 45, 48, 52, 55, 58, 59, 62, 66, 69–72, 80–86, 88–91, 93–98, 100, 103, 104, 107, 108, 111, 112, 115, 116
- bal.tab.CBPS, 19
- bal.tab.CBPS(), 18
- bal.tab.cem.match, 22
- bal.tab.cem.match(), 18
- bal.tab.data.frame (bal.tab.formula), 36
- bal.tab.data.frame(), 18, 27, 28, 84, 85, 90, 91
- bal.tab.data.frame.list
 - (bal.tab.time.list), 62
- bal.tab.default, 25
- bal.tab.designmatch, 30
- bal.tab.designmatch(), 18
- bal.tab.ebalance, 33
- bal.tab.ebalance(), 18
- bal.tab.formula, 36
- bal.tab.formula(), 18, 27–29
- bal.tab.formula.list
 - (bal.tab.time.list), 62
- bal.tab.Match, 42
- bal.tab.Match(), 18
- bal.tab.matchit, 46
- bal.tab.matchit(), 18, 50, 52
- bal.tab.matrix (bal.tab.formula), 36
- bal.tab.mimids, 49
- bal.tab.mimids(), 18
- bal.tab.multi(), 14, 21, 22, 24, 27, 29, 32, 35, 40, 41, 44, 48, 51, 54, 58, 59, 61, 65, 66, 69, 92, 110
- bal.tab.optmatch, 52
- bal.tab.optmatch(), 18
- bal.tab.ps, 56
- bal.tab.ps(), 18
- bal.tab.sbwcau, 59
- bal.tab.sbwcau(), 18
- bal.tab.time.list, 62
- bal.tab.time.list(), 18, 27–29, 88
- bal.tab.weightit, 67
- bal.tab.weightit(), 18, 50, 52
- balance-statistics, 70, 80
- balance-summary, 72
- CBPS::balance(), 21
- CBPS::CBMSM(), 19
- CBPS::CBPS(), 19
- class-bal.tab.cens, 81
- class-bal.tab.cluster, 83
- class-bal.tab.imp, 84
- class-bal.tab.msm, 86
- class-bal.tab.multi, 88
- class-bal.tab.subclass, 90
- col_w_corr (balance-summary), 72
- col_w_corr(), 6
- col_w_cov (balance-summary), 72
- col_w_cov(), 16, 71
- col_w_dcorr (balance-summary), 72
- col_w_dcov (balance-summary), 72
- col_w_dcov(), 71
- col_w_ks (balance-summary), 72

`col_w_ks()`, [5](#), [71](#)
`col_w_mean`(`balance-summary`), [72](#)
`col_w_ovl`(`balance-summary`), [72](#)
`col_w_ovl()`, [5](#), [71](#)
`col_w_sd`(`balance-summary`), [72](#)
`col_w_sd()`, [58](#)
`col_w_smd`(`balance-summary`), [72](#)
`col_w_smd()`, [5](#), [14](#), [15](#), [20](#), [23](#), [26](#), [31](#), [35](#), [39](#),
[44](#), [47](#), [50](#), [54](#), [57](#), [61](#), [64](#), [68](#), [71](#)
`col_w_vr`(`balance-summary`), [72](#)
`col_w_vr()`, [71](#)
`colMeans()`, [72](#), [78](#)

`default method`, [13](#)
`defaults`, [11](#)
`density()`, [76](#), [78](#)
`designmatch::bmatch()`, [31](#)
`designmatch::meantab()`, [32](#)
`display options`, [13](#), [21](#), [24](#), [32](#), [36](#), [40](#), [45](#),
[48](#), [51](#), [55](#), [58](#), [61](#), [65](#), [69](#)
`display-options`, [91](#), [105](#)
`do.call()`, [94](#)

`ebal::ebalance()`, [34](#)
`ebal::ebalance.trim()`, [34](#)
`extract.bal.tab`, [95](#)

`f.build`, [97](#)
`format()`, [93](#), [115](#)
`format.bal.tab`(`extract.bal.tab`), [95](#)
`formula`, [98](#)

`get.cobalt.options`
 (`set.cobalt.options`), [111](#)
`get.cobalt.options()`, [94](#)
`get.w`, [98](#)
`get.w()`, [14](#), [20](#), [24](#), [27](#), [32](#), [35](#), [40](#), [44](#), [47](#), [51](#),
[54](#), [58](#), [61](#), [65](#), [69](#)
`getOption()`, [111](#)
`ggplot2::facet_grid()`, [10](#), [11](#)
`ggplot2::geom_bar()`, [10](#)
`ggplot2::geom_density()`, [9](#), [10](#)
`ggplot2::geom_histogram()`, [9](#), [10](#)
`ggplot2::geom_point()`, [10](#), [104](#)
`ggplot2::geom_smooth()`, [11](#)
`ggplot2::geom_step()`, [10](#)
`ggplot2::ggplot()`, [10](#)
`ggplot2::labs()`, [11](#)
`ggplot2::scale_alpha()`, [104](#)

`ggplot2::theme()`, [10](#), [105](#)
`glm()`, [98](#)
`graphics::par()`, [10](#), [104](#)
`gridExtra::arrangeGrob()`, [105](#), [106](#)

`here`, [112](#)

`integrate()`, [5](#), [77](#)

`lalonge`, [101](#)
`lalonge_mis`(`lalonge`), [101](#)
`list()`, [94](#)
`love.plot`, [102](#)
`love.plot()`, [11](#), [71](#), [93](#), [97](#), [115](#)

`match.call()`, [28](#)
`Matching::Match()`, [43](#)
`Matching::MatchBalance()`, [45](#)
`Matching::Matchby()`, [43](#)
`MatchIt::matchit()`, [46](#)
`MatchIt::summary.matchit()`, [48](#)
`MatchThem::matchthem()`, [50](#)
`MatchThem::weightthem()`, [50](#)
`model.matrix()`, [115](#)

`options()`, [111](#), [112](#)
`optmatch::fullmatch()`, [53](#)
`optmatch::pairmatch()`, [53](#)

`print()`, [93](#), [111](#), [115](#)
`print.bal.tab`, [108](#)
`print.bal.tab()`, [17](#), [84](#), [86](#), [88](#), [90](#), [91](#), [94](#),
[95](#), [97](#)

`rank()`, [78](#)
`reformulate()`, [98](#)
`reshape()`, [65](#)

`sbw::sbw()`, [60](#)
`sbw::summarize()`, [61](#)
`set.cobalt.options`, [111](#)
`set.cobalt.options()`, [14](#), [20](#), [23](#), [26](#), [31](#),
[35](#), [39](#), [44](#), [47](#), [50](#), [54](#), [57](#), [60](#), [64](#), [68](#),
[94](#)
`splitfactor`, [113](#)
`splitfactor()`, [76](#)
`stats`, [13](#), [15](#), [19](#), [23](#), [26](#), [31](#), [34](#), [39](#), [43](#), [46](#),
[50](#), [53](#), [57](#), [60](#), [64](#), [68](#), [92](#), [103](#), [104](#)

`treat`, [3](#), [82](#)

twang::bal.table(), [58](#)
twang::iptw(), [56](#)
twang::mnps(), [56](#)
twang::ps(), [56](#)
twangContinuous::ps.cont(), [56](#)

unsplitfactor(splitfactor), [113](#)
utils::write.csv(), [116](#)

var.names, [115](#)
var.names(), [93](#), [94](#), [104](#)

WeightIt::weightit(), [68](#)
WeightIt::weightitMSM(), [65](#), [68](#)