

Package ‘weightflow’

June 30, 2026

Title Declarative API for Staged Survey Weights

Version 0.1.0

Description Builds survey weights from design base weights by chaining hierarchical adjustments (unknown eligibility, nonresponse and calibration) through a declarative, pipeable, 'tidymodels'-style API. Calibration follows Deville and Sarndal (1992) <[doi:10.2307/2290268](https://doi.org/10.2307/2290268)>. Variances are obtained with a bootstrap that resamples primary sampling units and re-applies the whole recipe on each replicate, following the rescaling bootstrap of Rao and Wu (1988) <[doi:10.1080/01621459.1988.10478591](https://doi.org/10.1080/01621459.1988.10478591)>, so the replicate weights carry the variability of every adjustment. The weights also bridge to the 'survey' and 'srvyr' packages for design-based inference.

License MIT + file LICENSE

Encoding UTF-8

Language en-US

Depends R (>= 4.1.0)

Imports stats, utils, graphics

Suggests rpart, ranger, testthat (>= 3.0.0), survey, srvyr, dplyr, knitr, rmarkdown, spelling

Config/roxygen2/version 8.0.0

URL <https://github.com/jpferreira33/weightflow>,
<https://jpferreira33.github.io/weightflow/>

BugReports <https://github.com/jpferreira33/weightflow/issues>

Config/testthat/edition 3

LazyData true

VignetteBuilder knitr

NeedsCompilation no

Author Juan Pablo Ferreira [aut, cre]

Maintainer Juan Pablo Ferreira <juanpablo.ferreira@fcea.edu.uy>

Repository CRAN

Date/Publication 2026-06-30 11:30:02 UTC

Contents

as_svydesign	2
bootstrap_estimate	3
bootstrap_weights	4
collect_replicate_weights	5
collect_weights	6
design_effect	7
plot.prepped_weighting_spec	7
population	8
prep	9
report_weighting	9
sample_one	10
sample_survey	11
step_assert	11
step_calibrate	12
step_drop_ineligible	14
step_model_calibration	15
step_nonresponse	16
step_rescale	17
step_round	18
step_select_within	19
step_trim	19
step_trim_weights	21
step_unknown_eligibility	21
summary.prepped_weighting_spec	22
weighting_spec	23
weight_factors	24
y_model	24
Index	26

as_svydesign	<i>Export weightflow weights to a survey design</i>
--------------	---

Description

as_svydesign() builds a linearization (ultimate-cluster) design from a prepped recipe; as_svrepdesign() builds a replicate-weights design from a bootstrap object, so survey/srvyr standard errors include the recipe's adjustments. Both require the 'survey' package.

Usage

```
as_svydesign(object, ids, strata = NULL, weight_name = ".weight", ...)
```

```
as_svrepdesign(boot, ...)
```

Arguments

object	a prepped recipe (for <code>as_svydesign</code>) or a data frame with the weight and design columns.
ids, strata	column names of the PSU and the stratum.
weight_name	name of the weight column.
...	passed to the survey constructor.
boot	a <code>weightflow_boot</code> object.

Value

A `survey.design` / `svyrep.design` object.

`bootstrap_estimate` *Bootstrap estimate, standard error and confidence interval*

Description

Applies a statistic to the point weights and to every replicate, and summarises it with the bootstrap variance $(1/B) \sum (\theta_b^* - \theta)^2$.

Usage

```
bootstrap_estimate(boot, statistic, level = 0.95)
```

```
boot_total(boot, variable)
```

```
boot_mean(boot, variable)
```

Arguments

boot	a <code>weightflow_boot</code> object.
statistic	a function <code>function(w, data)</code> returning a numeric scalar (or vector) given a weight vector and the data.
level	confidence level for the (normal) interval.
variable	name of the variable to estimate.

Value

A data frame with `estimate`, `se`, `ci_lower`, `ci_upper`.

bootstrap_weights *Bootstrap replicate weights that re-apply the recipe*

Description

Builds bootstrap replicate weights by resampling primary sampling units (PSUs) with replacement within strata and re-running the whole recipe on each replicate. Because every adjustment (non-response, calibration, ...) is recomputed per replicate, the resulting replicate weights propagate the variability introduced by each weighting stage.

Usage

```
bootstrap_weights(
  object,
  replicates = 200L,
  strata = NULL,
  psu = NULL,
  m = NULL,
  seed = NULL,
  progress = TRUE
)
```

Arguments

object	a <code>weighting_spec</code> (or a prepped one) holding the recipe.
replicates	number of bootstrap replicates.
strata, psu	column names of the stratum and the PSU. If <code>psu</code> is <code>NULL</code> each unit is its own PSU; if <code>strata</code> is <code>NULL</code> a single stratum is assumed.
m	PSUs drawn per stratum (default $n - 1$).
seed	optional RNG seed.
progress	print progress every 25 replicates.

Details

The multiplier is the Rao-Wu rescaling bootstrap: within a stratum with n PSUs, m PSUs are drawn with replacement (default $m = n - 1$) and unit i in PSU k gets $\lambda = 1 - \sqrt{m/(n-1)} + \sqrt{m/(n-1)} (n/m) t_k$, with t_k the number of times its PSU was drawn.

Value

An object of class `weightflow_boot` with the `replicates` matrix (units x replicates), the point weights, and the design metadata.

Examples

```
spec <- weighting_spec(sample_survey, base_weights = pw) |>
  step_calibrate(method = "raking",
                margins = list(region = c(table(population$region))))
boot <- bootstrap_weights(spec, replicates = 50, strata = "region",
                        psu = "psu", seed = 1)
boot_total(boot, "responded")
```

```
collect_replicate_weights
```

Collect replicate weights into a data frame ready for srvyr

Description

Returns the data with the point weight and the bootstrap replicate weights as columns, so it can be fed directly to `srvyr::as_survey_rep()` (or `survey::svrepdesign()`). Replicate columns are full weights, so use `combined.weights = TRUE`, `scale = 1 / R`, `rscals = 1`, `mse = TRUE`.

Usage

```
collect_replicate_weights(
  boot,
  weight_name = ".weight",
  prefix = "rep_",
  drop_zero = TRUE
)
```

Arguments

<code>boot</code>	a <code>weightflow_boot</code> object.
<code>weight_name</code>	name of the point-weight column to add.
<code>prefix</code>	prefix for the replicate-weight columns (<code>rep_1</code> , <code>rep_2</code> , ...).
<code>drop_zero</code>	keep only active units (point weight > 0).

Value

A data frame: the original columns, `weight_name`, and one column per replicate. The number of replicates is stored in attribute "R".

Examples

```
spec <- weighting_spec(sample_survey, base_weights = pw) |>
  step_calibrate(method = "raking",
                margins = list(region = c(table(population$region))))
boot <- bootstrap_weights(spec, replicates = 30, strata = "region",
                        psu = "psu", seed = 1, progress = FALSE)
df <- collect_replicate_weights(boot)
```

```

if (requireNamespace("srvyr", quietly = TRUE) &&
    requireNamespace("dplyr", quietly = TRUE)) {
  srvyr::as_survey_rep(df, weights = .weight,
                      repweights = dplyr::starts_with("rep_"),
                      type = "bootstrap", combined.weights = TRUE,
                      scale = 1 / attr(df, "R"), rscales = 1, mse = TRUE)
}

```

collect_weights	<i>Extract the data with the computed weights</i>
-----------------	---

Description

Extract the data with the computed weights

Usage

```

collect_weights(
  object,
  drop_zero = TRUE,
  keep_intermediate = FALSE,
  weight_name = ".weight"
)

```

Arguments

object	a prepped object (output of prep()).
drop_zero	logical. If TRUE, drops rows with final weight 0 (ineligible / nonresponse). Default TRUE.
keep_intermediate	logical. If TRUE, adds one column per stage.
weight_name	name of the final weight column. Default ".weight".

Value

data.frame.

Examples

```

fitted <- weighting_spec(sample_survey, base_weights = pw) |>
  step_nonresponse(respondent = responded, method = "weighting_class", by = "region") |>
  prep()
head(collect_weights(fitted))

```

design_effect	<i>Kish design effect from unequal weighting</i>
---------------	--

Description

$deff = 1 + CV^2(w) = m * \sum(w^2) / (\sum(w))^2$, over the active weights. The effective sample size is $n_{eff} = m / deff$.

Usage

```
design_effect(w)
```

Arguments

w vector of weights (zeros are dropped).

Value

list with deff, n_eff, cv and n.

Examples

```
design_effect(sample_survey$pw)
```

plot.prepped_weighting_spec	<i>Diagnostic plots for the weights</i>
-----------------------------	---

Description

Diagnostic plots for the weights

Usage

```
## S3 method for class 'prepped_weighting_spec'
plot(x, type = c("all", "factors", "summary"), ...)
```

Arguments

x a prepped object (output of prep()).

type "all" (default): per-step adjustment-factor histograms PLUS the summary panel (final weights, cumulative factor, base vs final, deff by stage), all in one grid. "factors": only the per-step factor histograms. "summary": only the summary panel.

... ignored.

Value

(invisibly) x. Called for its side effect of drawing the diagnostic plots.

Examples

```
fitted <- weighting_spec(sample_survey, base_weights = pw) |>
  step_nonresponse(respondent = responded, method = "weighting_class", by = "region") |>
  prep()
plot(fitted)
```

population

Example target population for weightflow

Description

A simulated population (sampling frame) of individuals nested in households and primary sampling units (PSUs) within strata (regions), with demographic auxiliaries and two outcomes. Used to illustrate calibration targets and model calibration, and to validate weighted estimates. Generated by data-raw/weightflow_data.R.

Usage

```
population
```

Format

A data frame with one row per person:

person_id individual identifier

household_id household identifier (cluster)

psu primary sampling unit (segment) within the stratum

region stratum: North, South, East or West

sex F or M

age age in years (18-95)

income annual income

employed employment indicator (0/1)

```
prep
```

Estimate the weighting cascade

Description

Walks the steps in the order they were added, starting from the base weights. Each step multiplies the current weight by its adjustment factor.

Usage

```
prep(spec)
```

Arguments

```
spec
```

a `weighting_spec`.

Value

a "prepped_weighting_spec" object.

Examples

```
rec <- weighting_spec(sample_survey, base_weights = pw) |>
  step_nonresponse(respondent = responded, method = "weighting_class", by = "region")
prep(rec)
```

```
report_weighting
```

Build a nice HTML report of the weighting recipe

Description

Writes a self-contained HTML file (no dependencies, no server) showing the pipeline, the parameters requested at each step, the per-stage summary (n, sum, CV, Kish deff, effective n) and per-step diagnostics, and opens it in the browser.

Usage

```
report_weighting(object, file = NULL, open = TRUE, plots = TRUE)
```

Arguments

```
object
```

a prepped object (output of `prep()`).

```
file
```

output path; if `NULL`, a temporary `.html` file.

```
open
```

logical; open the file in the browser.

```
plots
```

logical; add per-step plots (weight before-vs-after scatter and adjustment-factor histogram). Uses `ggplot2` if installed, else base graphics.

Value

(invisibly) the path to the written HTML file.

Examples

```
fitted <- weighting_spec(sample_survey, base_weights = pw) |>
  step_nonresponse(respondent = responded, method = "weighting_class", by = "region") |>
  prep()
f <- tempfile(fileext = ".html")
report_weighting(fitted, file = f, open = FALSE)
```

sample_one

Example survey sample (select-one-person, multistage)

Description

A realistic multistage design (stratum -> PSU -> household, then one selected person per household). Unknown-eligibility and ineligible addresses appear as single rows with no roster; resolved eligible households are either reached (a roster is obtained) or are household nonresponse; in reached households one person is selected with an unequal within-household probability and may or may not respond. Supports the full household pipeline: household-level eligibility (`cluster`), dropping ineligibles, household and person nonresponse, and `step_select_within`. Generated by `data-raw/weightflow_data.R`.

Usage

```
sample_one
```

Format

A data frame with one row per sampled household (the selected person, or a single placeholder row for non-roster cases):

person_id, household_id, psu identifiers

region stratum

sex, age selected person's attributes (NA on non-roster rows)

pw design base weight (product of the stage selection probabilities)

status "eligible", "ineligible" or "unknown"

unknown_elig 1 if eligibility is unknown (no roster)

ineligible 1 if the address is out of scope (no roster)

hh_responded 1 reached, 0 household nonresponse, NA for non-eligible

responded 1 if the selected person responded (NA on non-roster rows)

n_elig number of eligible persons in the household (NA on non-roster rows)

p_within within-household selection probability of the selected person

income, employed survey outcomes; NA unless the person responded

sample_survey	<i>Example survey sample (take-all roster)</i>
---------------	--

Description

A stratified household sample drawn from population where every eligible person in the household is kept (take-all roster). Carries unequal design base weights, an unknown-eligibility flag and a person-level response indicator; survey outcomes (income, employed) are observed only for respondents. Generated by `data-raw/weightflow_data.R`.

Usage

```
sample_survey
```

Format

A data frame with one row per sampled person:

person_id, household_id, psu identifiers

region, sex, age frame auxiliaries, known for all units

pw design base weight (inverse sampling fraction)

unknown_elig 1 if eligibility is unknown

responded 1 if the person responded

income, employed survey outcomes; NA for nonrespondents

step_assert	<i>Assert conditions on the weights at this point of the cascade</i>
-------------	--

Description

A checkpoint that does NOT change the weights; it verifies conditions and fails (error) or warns if they are not met. Useful to guard a production pipeline (tidymodels-style tests inside the recipe).

Usage

```
step_assert(
  spec,
  max_deff = NULL,
  max_weight_ratio = NULL,
  min_n_eff = NULL,
  on_fail = c("error", "warning")
)
```

Arguments

spec a weighting_spec.
 max_deff numeric or NULL. Maximum acceptable Kish design effect.
 max_weight_ratio numeric or NULL. Maximum allowed final/base weight ratio (per active unit).
 min_n_eff numeric or NULL. Minimum acceptable effective sample size.
 on_fail "error" (stop the cascade) or "warning".

Value

The input weighting_spec with this step appended (for piping). The step is evaluated later by prep().

Examples

```
weighting_spec(sample_survey, base_weights = pw) |>
  step_assert(max_deff = 5, on_fail = "warning") |> prep()
```

step_calibrate	<i>Calibration to population totals</i>
----------------	---

Description

Calibration to population totals

Usage

```
step_calibrate(
  spec,
  margins = NULL,
  method = c("raking", "poststratify", "linear"),
  formula = NULL,
  totals = NULL,
  cluster = NULL,
  equal_within_cluster = FALSE,
  calfun = c("linear", "logit"),
  bounds = NULL,
  maxit = 50L,
  tol = 1e-06
)
```

Arguments

spec	a weighting_spec.
margins	named list (for "raking"/"poststratify"). Each element is a named numeric vector with the target totals per category. E.g.: list(sex = c(M = 5000, F = 5200), region = c(N = 3000, S = 7200)).
method	"raking" (IPF, categorical margins), "poststratify" (a single categorical variable) or "linear" (GREG / regression estimator; handles continuous and categorical auxiliaries together).
formula	(only "linear") auxiliary formula, e.g. ~ sex + income. Uses model.matrix; includes the intercept unless you write ~ 0 + ...
totals	(only "linear") named numeric vector with the population totals, names matching the model.matrix columns (including "(Intercept)" = N if there is an intercept). If names do not match, the error lists the expected ones.
cluster	(only "linear") name of the cluster id column (e.g. "household"), for equal weights within the cluster.
equal_within_cluster	(only "linear") logical. If TRUE, Lemaitre-Dufour (1987) integrative calibration: a single weight per cluster. Requires cluster. Final weights are equal within the cluster provided the incoming weight is also uniform within the cluster.
calfun	(only "linear") distance function: "linear" ($g = 1 + u$) or "logit" (bounded by construction). With "logit", bounds is required.
bounds	(only "linear") numeric c(L, U) with $L < 1 < U$. Bounds on the calibration factor g (g-weights). With "linear" it truncates; with "logit" it is enforced smoothly. Avoids extreme/negative weights without a separate trimming step.
maxit, tol	convergence control for raking and bounded calibration.

Value

The input weighting_spec with this step appended (for piping). The step is evaluated later by prep().

Examples

```
# Raking to population margins
weighting_spec(sample_survey, base_weights = pw) |>
  step_nonresponse(respondent = responded, method = "weighting_class", by = "region") |>
  step_calibrate(method = "raking",
                 margins = list(sex = c(table(population$sex)),
                               region = c(table(population$region)))) |>
  prep()
```

step_drop_ineligible *Drop ineligible (out-of-scope) units*

Description

Sets the weight of known-ineligible units to zero so they leave the cascade (excluded from every later step and from collect_weights). No redistribution is done.

Usage

```
step_drop_ineligible(spec, ineligible)
```

Arguments

spec	a weighting_spec.
ineligible	a 0/1 dummy column (1 = ineligible) or any logical condition (unquoted) that is TRUE for out-of-scope units.

Details

Apply it AFTER step_unknown_eligibility: ineligibles must be present and NOT flagged as unknown during that step, so they take part in the known-eligibility group and receive their share of the redistributed unknown weight. Their weight is then correctly discarded here (it represents the ineligible share of the unknown units, which are out of scope).

Value

The input weighting_spec with this step appended (for piping). The step is evaluated later by prep().

Examples

```
df <- transform(sample_survey,  
                 ineligible = as.integer(region == "West" & age > 90))  
weighting_spec(df, base_weights = pw) |>  
  step_drop_ineligible(ineligible = ineligible) |>  
  prep()
```

`step_model_calibration`*Model calibration (model-assisted, Wu & Sitter 2001)*

Description

Fits a working model for each study variable y , predicts over the population, and calibrates the weights so that the sample total of each prediction equals its population total (model-assisted efficiency). It also calibrates to the X totals (consistency with the auxiliary controls).

Usage

```
step_model_calibration(  
  spec,  
  x_formula,  
  models,  
  population,  
  cluster = NULL,  
  equal_within_cluster = FALSE  
)
```

Arguments

<code>spec</code>	a <code>weighting_spec</code> .
<code>x_formula</code>	formula of the consistency auxiliaries, e.g. <code>~ sex + region</code> .
<code>models</code>	named list of models created with <code>y_model()</code> . The names label the prediction constraints.
<code>population</code>	population data.frame with the auxiliary and predictor columns (the y variables are not needed; they are predicted).
<code>cluster</code>	name of the cluster id column (e.g. "household"), for equal weights within the cluster.
<code>equal_within_cluster</code>	logical. If TRUE, integrative calibration: a single weight per cluster. Requires <code>cluster</code> and that the incoming weight be uniform within the cluster.

Details

Requires COMPLETE auxiliary information: a data.frame population with the `x_formula` columns and the model predictors for the whole population (or a reference frame/census).

Value

The input `weighting_spec` with this step appended (for piping). The step is evaluated later by `prep()`.

Examples

```
weighting_spec(sample_survey, base_weights = pw) |>
  step_nonresponse(respondent = responded, method = "weighting_class", by = "region") |>
  step_model_calibration(
    x_formula = ~ sex + region,
    models = list(income = y_model(income ~ age + sex, engine = "glm")),
    population = population) |>
  prep()
```

step_nonresponse	<i>Nonresponse adjustment</i>
------------------	-------------------------------

Description

Nonresponse adjustment

Usage

```
step_nonresponse(
  spec,
  respondent,
  method = c("weighting_class", "propensity"),
  by = NULL,
  formula = NULL,
  engine = c("logit", "tree", "forest"),
  num_classes = 5L,
  cluster = NULL
)
```

Arguments

spec	a <code>weighting_spec</code> .
respondent	a 0/1 dummy column (1 = responded) or any logical condition (unquoted) TRUE for respondents. Eligible cases that are not respondents are treated as nonresponse.
method	"weighting_class" (cells) or "propensity" (predictive model).
by	character. Adjustment cells for method = "weighting_class".
formula	predictor formula (right-hand side only), e.g. ~ age + region, used when method = "propensity".
engine	engine to estimate the propensity when method = "propensity": "logit" (logistic regression, base R), "tree" (CART via package 'rpart') or "forest" (random forest via package 'ranger'). 'rpart' and 'ranger' are optional: only needed if you pick that engine.
num_classes	integer or NULL. Controls how propensities are used: an integer forms that many propensity classes (cell adjustment within each class); NULL applies the direct factor 1/p to each unit.

cluster character or NULL. If given, the adjustment is done at the cluster (e.g. household) level for whole-household nonresponse: each household counts once with its (uniform) weight; in "weighting_class" the redistribution is between responding and nonresponding households within the cells, and in "propensity" the model is fitted with one row per household (household auxiliaries), predicting the household response. The resulting factor is assigned to every member; nonresponding households go to zero. As always, only active units (weight > 0) take part, so units already dropped (unknown eligibility, ineligible) are excluded automatically.

Value

The input `weighting_spec` with this step appended (for piping). The step is evaluated later by `prep()`.

Examples

```
weighting_spec(sample_survey, base_weights = pw) |>
  step_nonresponse(respondent = responded, method = "weighting_class",
                  by = "region")

# household-level nonresponse (whole household responds or not)
weighting_spec(sample_survey, base_weights = pw) |>
  step_nonresponse(respondent = responded, method = "weighting_class",
                  by = "region", cluster = "household_id") |>
  prep()
```

step_rescale	<i>Rescale (normalize) the weights</i>
--------------	--

Description

Rescale (normalize) the weights

Usage

```
step_rescale(spec, to = c("n", "total"), total = NULL, by = NULL)
```

Arguments

<code>spec</code>	a <code>weighting_spec</code> .
<code>to</code>	"n" (weights sum to the number of active units, i.e. mean weight 1) or "total" (weights sum to total).
<code>total</code>	numeric. Target sum when <code>to = "total"</code> .
<code>by</code>	character. Rescale within these groups (optional). With <code>to = "n"</code> , each group sums to its own active count.

Value

The input `weighting_spec` with this step appended (for piping). The step is evaluated later by `prep()`.

Examples

```
weighting_spec(sample_survey, base_weights = pw) |>
  step_rescale(to = "n") |> prep()
```

step_round	<i>Round the final weights</i>
------------	--------------------------------

Description

Optional step, typically the last one (after calibration). Simple rounding ("nearest") slightly breaks the calibrated totals; "preserve_total" uses the largest-remainder method to keep the exact total.

Usage

```
step_round(spec, digits = 0L, method = c("nearest", "preserve_total"))
```

Arguments

spec	a <code>weighting_spec</code> .
digits	integer. Decimals to keep (0 = integers).
method	"nearest" (simple rounding) or "preserve_total" (keeps the sum of weights). Note: "preserve_total" can break equality of weights within a cluster; if you need integer and equal weights per household, use "nearest".

Value

The input `weighting_spec` with this step appended (for piping). The step is evaluated later by `prep()`.

Examples

```
weighting_spec(sample_survey, base_weights = pw) |>
  step_round(digits = 0) |> prep()
```

step_select_within	<i>Within-household selection adjustment</i>
--------------------	--

Description

When one (or a subsample) of the eligible persons is selected within each household, the selected person represents all eligible persons, so the weight is multiplied by the inverse of the within-household selection probability. Apply it after the (household-level) eligibility adjustment and before the nonresponse adjustment.

Usage

```
step_select_within(spec, prob = NULL, n_eligible = NULL)
```

Arguments

spec	a <code>weighting_spec</code> .
prob	unquoted column with the within-household selection probability of the selected person (need not be $1/n_eligible$). The weight is multiplied by $1/prob$.
n_eligible	unquoted column with the number of eligible persons in the household, for simple random selection of one person. The weight is multiplied by <code>n_eligible</code> (equivalent to $prob = 1/n_eligible$).

Value

The input `weighting_spec` with this step appended (for piping). The step is evaluated later by `prep()`.

Examples

```
# simple random selection of one eligible person per household
df <- transform(sample_survey,
                 n_elig = ave(person_id, household_id, FUN = length))
weighting_spec(df, base_weights = pw) |>
  step_select_within(n_eligible = n_elig)
```

step_trim	<i>Trim extreme weights</i>
-----------	-----------------------------

Description

Caps weights above a limit and, optionally, redistributes the excess among the others to preserve the weighted total (Potter 1988, 1990; Liu et al. 2004). Optional step that can be inserted anywhere in the recipe, even several times. Operates on the CURRENT weights at that point of the cascade.

Usage

```
step_trim(
  spec,
  max_ratio,
  min_ratio = NULL,
  reference = c("base", "median", "value"),
  redistribute = TRUE,
  by = NULL,
  maxit = 50L
)
```

Arguments

spec	a <code>weighting_spec</code> .
max_ratio	number. Upper cap. Its meaning depends on reference. E.g. with reference = "base" and max_ratio = 4, no weight may exceed 4 times its design weight.
min_ratio	number or NULL. Lower floor (same units as max_ratio).
reference	"base" (multiple of each unit's base weight), "median" (multiple of the median of current weights) or "value" (absolute weight value).
redistribute	logical. If TRUE, redistributes the trimmed excess among the uncapped weights to preserve the total (iterating). If you calibrate afterwards you can use FALSE: calibration restores the totals.
by	character. Groups within which to redistribute (optional).
maxit	integer. Maximum cap+redistribution iterations.

Details

There is no standard threshold: `max_ratio` is an analyst decision, a bias-variance trade-off. Use Kish's design effect (see summary) to judge whether trimming is worth it.

Value

The input `weighting_spec` with this step appended (for piping). The step is evaluated later by `prep()`.

Examples

```
weighting_spec(sample_survey, base_weights = pw) |>
  step_trim(max_ratio = 3, reference = "base")
```

step_trim_weights *Automatic weight trimming (survey-style)*

Description

Caps weights into [lower, upper] and redistributes the change among the untrimmed units to preserve the total, mirroring `survey::trimWeights()`. By default no weight may fall below 1, and the upper cap is set by an automatic empirical rule (Tukey far-out fence: $Q3 + 3*IQR$).

Usage

```
step_trim_weights(spec, lower = 1, upper = NULL, strict = TRUE, maxit = 50L)
```

Arguments

spec	a <code>weighting_spec</code> .
lower	numeric. Lower floor (default 1: no weight below 1).
upper	numeric or NULL. Upper cap. If NULL, automatic rule $Q3 + 3*IQR$ of the active weights.
strict	logical. If TRUE (default), iterate cap+redistribution until no weight is outside [lower, upper] (like <code>survey</code> 's <code>strict = TRUE</code>). If FALSE, a single pass (redistribution may push some weights slightly past the cap).
maxit	integer. Maximum iterations when <code>strict = TRUE</code> .

Value

The input `weighting_spec` with this step appended (for piping). The step is evaluated later by `prep()`.

Examples

```
weighting_spec(sample_survey, base_weights = pw) |>
  step_nonresponse(respondent = responded, method = "weighting_class", by = "region") |>
  step_trim_weights(lower = 1, strict = TRUE) |> prep()
```

step_unknown_eligibility

Unknown-eligibility adjustment

Description

Redistributes the weight of unknown-eligibility cases among the known-eligibility cases, within the cells defined by `by`.

Usage

```
step_unknown_eligibility(spec, unknown, by = NULL, cluster = NULL)
```

Arguments

spec	a <code>weighting_spec</code> .
unknown	a 0/1 dummy column (1 = eligibility unknown) or any logical condition (unquoted) that is TRUE for unknown-eligibility cases. Evaluated on the data.
by	character. Variables defining the adjustment cells (optional).
cluster	character. Cluster (e.g. household) id column. If given, the redistribution is done at the cluster level: each cluster counts once with its (uniform) weight, the weight of unknown-eligibility clusters is redistributed among the known ones, and the adjusted weight is assigned to every member. Use this when unknown-eligibility units have no roster (one row per address) while resolved units are expanded by person.

Value

The input `weighting_spec` with this step appended (for piping). The step is evaluated later by `prep()`.

Examples

```
weighting_spec(sample_survey, base_weights = pw) |>
  step_unknown_eligibility(unknown = unknown_elig, by = "region")

# household-level redistribution (unknown units without roster)
weighting_spec(sample_survey, base_weights = pw) |>
  step_unknown_eligibility(unknown = unknown_elig, by = "region",
                          cluster = "household_id")
```

```
summary.prepped_weighting_spec
```

Detailed per-step diagnostics

Description

Detailed per-step diagnostics

Usage

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

Arguments

object	a prepped object (output of <code>prep()</code>).
...	ignored.

Value

(invisibly) the prepped object.

Examples

```
fitted <- weighting_spec(sample_survey, base_weights = pw) |>  
  step_nonresponse(respondent = responded, method = "weighting_class", by = "region") |>  
  prep()  
summary(fitted)
```

weighting_spec	<i>Start a weighting specification</i>
----------------	--

Description

Creates an inert recipe object. Nothing is computed until `prep()` is called.

Usage

```
weighting_spec(data, base_weights)
```

Arguments

<code>data</code>	data.frame with the sample units (one row per case).
<code>base_weights</code>	unquoted name of the design base-weight column.

Value

an object of class "weighting_spec".

Examples

```
rec <- weighting_spec(sample_survey, base_weights = pw)  
rec
```

weight_factors	<i>Per-unit adjustment factors table</i>
----------------	--

Description

Returns a data.frame with the weight at each stage and the factor of each step (stage weight / previous-stage weight), handy for custom plots.

Usage

```
weight_factors(object)
```

Arguments

object a prepped object (output of prep()).

Value

data.frame with one weight column per stage and one factor per step.

Examples

```
fitted <- weighting_spec(sample_survey, base_weights = pw) |>
  step_nonresponse(respondent = responded, method = "weighting_class", by = "region") |>
  prep()
head(weight_factors(fitted))
```

y_model	<i>Specify a working model for a study variable y</i>
---------	---

Description

Specify a working model for a study variable y

Usage

```
y_model(formula, engine = c("glm", "tree", "forest"), family = NULL)
```

Arguments

formula full formula, e.g. income ~ sex + age_g.
engine "glm", "tree" (rpart) or "forest" (ranger).
family for engine = "glm": "gaussian", "binomial" or "poisson". For tree/forest, regression vs classification is inferred from y.

Value

a model specification list.

Examples

```
y_model(income ~ age + sex, engine = "glm")
```

Index

* datasets

- population, [8](#)
- sample_one, [10](#)
- sample_survey, [11](#)

as_svrepdesign (as_svydesign), [2](#)
as_svydesign, [2](#)

boot_mean (bootstrap_estimate), [3](#)
boot_total (bootstrap_estimate), [3](#)
bootstrap_estimate, [3](#)
bootstrap_weights, [4](#)

collect_replicate_weights, [5](#)
collect_weights, [6](#)

design_effect, [7](#)

plot.prepped_weighting_spec, [7](#)
population, [8](#)
prep, [9](#)

report_weighting, [9](#)

sample_one, [10](#)
sample_survey, [11](#)
step_assert, [11](#)
step_calibrate, [12](#)
step_drop_ineligible, [14](#)
step_model_calibration, [15](#)
step_nonresponse, [16](#)
step_rescale, [17](#)
step_round, [18](#)
step_select_within, [19](#)
step_trim, [19](#)
step_trim_weights, [21](#)
step_unknown_eligibility, [21](#)
summary.prepped_weighting_spec, [22](#)

weight_factors, [24](#)
weighting_spec, [23](#)

y_model, [24](#)