Package 'svrep'

July 6, 2022

Type Package

Title Tools for Creating, Updating, and Analyzing Survey Replicate Weights

Version 0.3.0

Author Ben Schneider

Maintainer Ben Schneider <benjamin.julius.schneider@gmail.com>

Description Provides tools for creating and working with survey replicate weights, extending functionality of the 'survey' package from Lumley (2004) <doi:10.18637/jss.v009.i08>. Methods are provided for applying nonresponse adjustments to both full-sample and replicate weights as suggested by Rust and Rao (1996) <doi:10.1177/096228029600500305>. Implements methods for sample-based calibration described by Opsomer and Erciulescu (2021) <https://www150.statcan.gc.ca/n1/pub/12-001-x/2021002/article/00006-eng.htm>. Diagnostic functions are included to compare weights and weighted estimates from different sets of replicate weights.

License GPL (>= 3)

URL https://github.com/bschneidr/svrep

BugReports https://github.com/bschneidr/svrep/issues

Encoding UTF-8

LazyData true

RoxygenNote 7.2.0

Imports stats, survey (>= 4.1), utils

Suggests knitr, covr, testthat (>= 3.0.0), rmarkdown, tidycensus, dplyr, srvyr

Config/testthat/edition 3

Depends R (>= 2.10)

VignetteBuilder knitr

NeedsCompilation no

Repository CRAN

Date/Publication 2022-07-06 00:00:02 UTC

R topics documented:

`as_data_frame_with_weights` *Convert a survey design object to a data frame with weights stored as columns*

Description

Convert a survey design object to a data frame with weights stored as columns

Usage

```
as_data_frame_with_weights(
  design,
  full_wgt_name = "FULL_SAMPLE_WGT",
  rep_wgt_prefix = "REP_WGT_"
)
```

Arguments

`design`	A survey design object, created with either the `survey` or `srvyr` packages.
`full_wgt_name`	The column name to use for the full-sample weights
`rep_wgt_prefix`	For replicate design objects, a prefix to use for the column names of the replicate weights. The column names will be created by appending the replicate number after the prefix.

Value

A data frame, with new columns containing the weights from the survey design object

Examples

```
data("lou_vax_survey", package = 'svrep')

library(survey)
# Create a survey design object
survey_design <- svydesign(data = lou_vax_survey,
                           weights = ~ SAMPLING_WEIGHT,
                           ids = ~ 1)

rep_survey_design <- as.svrepdesign(survey_design,
                                    type = "boot",
                                    replicates = 10)

# Adjust the weights for nonresponse
nr_adjusted_design <- redistribute_weights(
  design = rep_survey_design,
  reduce_if = RESPONSE_STATUS == "Nonrespondent",
  increase_if = RESPONSE_STATUS == "Respondent",
  by = c("RACE_ETHNICITY", "EDUC_ATTAINMENT")
)

# Save the survey design object as a data frame
nr_adjusted_data <- as_data_frame_with_weights(
  nr_adjusted_design,
  full_wgt_name = "NR_ADJUSTED_WGT",
  rep_wgt_prefix = "NR_ADJUSTED_REP_WGT_"
)
head(nr_adjusted_data)

# Check the column names of the result
colnames(nr_adjusted_data)
```

`calibrate_to_estimate` *Calibrate weights from a primary survey to estimated totals from a control survey, with replicate-weight adjustments that account for variance of the control totals*

Description

Calibrate the weights of a primary survey to match estimated totals from a control survey, using adjustments to the replicate weights to account for the variance of the estimated control totals. The adjustments to replicate weights are conducted using the method proposed by Fuller (1998). This method can be used to implement general calibration as well as post-stratification or raking specifically (see the details for the `calfun` parameter).

Usage

```
calibrate_to_estimate(
  rep_design,
  estimate,
  vcov_estimate,
  cal_formula,
  calfun = survey::cal.linear,
  bounds = list(lower = -Inf, upper = Inf),
  verbose = FALSE,
  maxit = 50,
  epsilon = 1e-07,
  variance = NULL,
  col_selection = NULL
)
```

Arguments

`rep_design`	A replicate design object for the primary survey, created with either the `survey` or `srvyr` packages.
`estimate`	A vector of estimated control totals. The names of entries must match the names from calling `svytotal(x = cal_formula, design = rep_design)`.
`vcov_estimate`	A variance-covariance matrix for the estimated control totals. The column names and row names must match the names of `estimate`.
`cal_formula`	A formula listing the variables to use for calibration. All of these variables must be included in `rep_design`.
`calfun`	A calibration function from the `survey` package, such as cal.linear, cal.raking, or cal.logit. Use `cal.linear` for ordinary post-stratification, and `cal.raking` for raking. See calibrate for additional details.
`bounds`	Parameter passed to grake for calibration. See calibrate for details.
`verbose`	Parameter passed to grake for calibration. See calibrate for details.
`maxit`	Parameter passed to grake for calibration. See calibrate for details.
`epsilon`	Parameter passed to grake for calibration. After calibration, the absolute difference between each calibration target and the calibrated estimate will be no larger than `epsilon` times (1 plus the absolute value of the target). See calibrate for details.
`variance`	Parameter passed to grake for calibration. See calibrate for details.
`col_selection`	Optional parameter to determine which replicate columns will have their control totals perturbed. If supplied, `col_selection` must be an integer vector with length equal to the length of `estimate`.

Details

With the Fuller method, each of `k` randomly-selected replicate columns from the primary survey are calibrated to control totals formed by perturbing the `k`-dimensional vector of estimated control

totals using a spectral decomposition of the variance-covariance matrix of the estimated control totals. Other replicate columns are simply calibrated to the unperturbed control totals.

Because the set of replicate columns whose control totals are perturbed should be random, there are multiple ways to ensure that this matching is reproducible. The user can either call set.seed before using the function, or supply a vector of randomly-selected column indices to the argument `perturbed_control_cols`.

Value

A replicate design object, with full-sample weights calibrated to totals from `estimate`, and replicate weights adjusted to account for variance of the control totals. The element `col_selection` indicates, for each replicate column of the calibrated primary survey, which column of replicate weights it was matched to from the control survey.

References

Fuller, W.A. (1998). "Replication variance estimation for two-phase samples." **Statistica Sinica**, *8*: 1153-1164.

Opsomer, J.D. and A. Erciulescu (2021). "Replication variance estimation after sample-based calibration." **Survey Methodology**, *47*: 265-277.

Examples

```
# Load example data for primary survey ----

  suppressPackageStartupMessages(library(survey))
  data(api)

  primary_survey <- svydesign(id=~dnum, weights=~pw, data=apiclus1, fpc=~fpc) |>
    as.svrepdesign(type = "JK1")

# Load example data for control survey ----

  control_survey <- svydesign(id = ~ 1, fpc = ~fpc, data = apisrs) |>
    as.svrepdesign(type = "JK1")

# Estimate control totals ----

  estimated_controls <- svytotal(x = ~ stype + enroll,
                                 design = control_survey)
  control_point_estimates <- coef(estimated_controls)
  control_vcov_estimate <- vcov(estimated_controls)

# Calibrate totals for one categorical variable and one numeric ----

  calibrated_rep_design <- calibrate_to_estimate(
    rep_design = primary_survey,
    estimate = control_point_estimates,
    vcov_estimate = control_vcov_estimate,
```

```
  cal_formula = ~ stype + enroll
)

# Inspect estimates before and after calibration ----

  ##_ For the calibration variables, estimates and standard errors
  ##_ from calibrated design will match those of the control survey

  svytotal(x = ~ stype + enroll, design = primary_survey)
  svytotal(x = ~ stype + enroll, design = control_survey)
  svytotal(x = ~ stype + enroll, design = calibrated_rep_design)

  ##_ Estimates from other variables will be changed as well

  svymean(x = ~ api00 + api99, design = primary_survey)
  svymean(x = ~ api00 + api99, design = control_survey)
  svymean(x = ~ api00 + api99, design = calibrated_rep_design)

# Inspect weights before and after calibration ----

  summarize_rep_weights(primary_survey, type = 'overall')
  summarize_rep_weights(calibrated_rep_design, type = 'overall')

# For reproducibility, specify which columns are randomly selected for Fuller method ----

  column_selection <- calibrated_rep_design$perturbed_control_cols
  print(column_selection)

  calibrated_rep_design <- calibrate_to_estimate(
    rep_design = primary_survey,
    estimate = control_point_estimates,
    vcov_estimate = control_vcov_estimate,
    cal_formula = ~ stype + enroll,
    col_selection = column_selection
  )
```

`calibrate_to_sample` *Calibrate weights from a primary survey to estimated totals from a control survey, with replicate-weight adjustments that account for variance of the control totals*

Description

Calibrate the weights of a primary survey to match estimated totals from a control survey, using adjustments to the replicate weights to account for the variance of the estimated control totals. The adjustments to replicate weights are conducted using the method proposed by Opsomer and Erciulescu (2021). This method can be used to implement general calibration as well as post-stratification or raking specifically (see the details for the `calfun` parameter).

Usage

```
calibrate_to_sample(
  primary_rep_design,
  control_rep_design,
  cal_formula,
  calfun = survey::cal.linear,
  bounds = list(lower = -Inf, upper = Inf),
  verbose = FALSE,
  maxit = 50,
  epsilon = 1e-07,
  variance = NULL,
  control_col_matches = NULL
)
```

Arguments

primary_rep_design	A replicate design object for the primary survey, created with either the survey or srvyr packages.
control_rep_design	A replicate design object for the control survey.
cal_formula	A formula listing the variables to use for calibration. All of these variables must be included in both primary_rep_design and control_rep_design.
calfun	A calibration function from the survey package, such as cal.linear, cal.raking, or cal.logit. Use cal.linear for ordinary post-stratification, and cal.raking for raking. See calibrate for additional details.
bounds	Parameter passed to grake for calibration. See calibrate for details.
verbose	Parameter passed to grake for calibration. See calibrate for details.
maxit	Parameter passed to grake for calibration. See calibrate for details.
epsilon	Parameter passed to grake for calibration. After calibration, the absolute difference between each calibration target and the calibrated estimate will be no larger than epsilon times (1 plus the absolute value of the target). See calibrate for details.
variance	Parameter passed to grake for calibration. See calibrate for details.
control_col_matches	Optional parameter to control which control survey replicate is matched to each primary survey replicate. Entries of NA denote a primary survey replicate not matched to any control survey replicate. If this parameter is not used, matching is done at random.

Details

With the Opsomer-Erciulescu method, each column of replicate weights from the control survey is randomly matched to a column of replicate weights from the primary survey, and then the column from the primary survey is calibrated to control totals estimated by perturbing the control sample's full-sample estimates using the estimates from the matched column of replicate weights from the

control survey.

If there are fewer columns of replicate weights in the control survey than in the primary survey, then not all primary replicate columns will be matched to a replicate column from the control survey.

If there are more columns of replicate weights in the control survey than in the primary survey, then the columns of replicate weights in the primary survey will be duplicated `k` times, where `k` is the smallest positive integer such that the resulting number of columns of replicate weights for the primary survey is greater than or equal to the number of columns of replicate weights in the control survey.

Because replicate columns of the control survey are matched *at random* to primary survey replicate columns, there are multiple ways to ensure that this matching is reproducible. The user can either call set.seed before using the function, or supply a mapping to the argument `control_col_matches`.

Value

A replicate design object, with full-sample weights calibrated to totals from `control_rep_design`, and replicate weights adjusted to account for variance of the control totals. If `primary_rep_design` had fewer columns of replicate weights than `control_rep_design`, then the number of replicate columns and the length of `rscales` will be increased by a multiple `k`, and the `scale` will be updated by dividing by `k`.

The element `control_column_matches` indicates, for each replicate column of the calibrated primary survey, which column of replicate weights it was matched to from the control survey. Columns which were not matched to control survey replicate column are indicated by `NA`.

The element `degf` will be set to match that of the primary survey to ensure that the degrees of freedom are not erroneously inflated by potential increases in the number of columns of replicate weights.

References

Opsomer, J.D. and A. Erciulescu (2021). "Replication variance estimation after sample-based calibration." **Survey Methodology**, *47*: 265-277.

Examples

```
# Load example data for primary survey ----

  suppressPackageStartupMessages(library(survey))
  data(api)

  primary_survey <- svydesign(id=~dnum, weights=~pw, data=apiclus1, fpc=~fpc) |>
    as.svrepdesign(type = "JK1")

# Load example data for control survey ----
```

```
  control_survey <- svydesign(id = ~ 1, fpc = ~fpc, data = apisrs) |>
    as.svrepdesign(type = "JK1")

# Calibrate totals for one categorical variable and one numeric ----

  calibrated_rep_design <- calibrate_to_sample(
    primary_rep_design = primary_survey,
    control_rep_design = control_survey,
    cal_formula = ~ stype + enroll,
  )

# Inspect estimates before and after calibration ----

  ##_ For the calibration variables, estimates and standard errors
  ##_ from calibrated design will match those of the control survey

    svytotal(x = ~ stype + enroll, design = primary_survey)
    svytotal(x = ~ stype + enroll, design = control_survey)
    svytotal(x = ~ stype + enroll, design = calibrated_rep_design)

  ##_ Estimates from other variables will be changed as well

    svymean(x = ~ api00 + api99, design = primary_survey)
    svymean(x = ~ api00 + api99, design = control_survey)
    svymean(x = ~ api00 + api99, design = calibrated_rep_design)

# Inspect weights before and after calibration ----

  summarize_rep_weights(primary_survey, type = 'overall')
  summarize_rep_weights(calibrated_rep_design, type = 'overall')

# For reproducibility, specify how to match replicates between surveys ----

  column_matching <- calibrated_rep_design$control_col_matches
  print(column_matching)

  calibrated_rep_design <- calibrate_to_sample(
    primary_rep_design = primary_survey,
    control_rep_design = control_survey,
    cal_formula = ~ stype + enroll,
    control_col_matches = column_matching
  )
```

`lou_pums_microdata` *ACS PUMS Data for Louisville*

Description

Person-level microdata from the American Community Survey (ACS) 2015-2019 public-use microdata sample (PUMS) data for Louisville, KY. This microdata sample represents all adults (persons

aged 18 or over) in Louisville, KY.

These data include replicate weights to use for variance estimation.

Usage

```
data(lou_pums_microdata)
```

Format

A data frame with 80 rows and 85 variables

- UNIQUE_IDUnique identifier for records
- AGEAge in years (copied from the AGEP variable in the ACS microdata)
- RACE_ETHNICITYRace and Hispanic/Latino ethnicity derived from RAC1P and HISP variables of ACS microdata and collapsed to a smaller number of categories.
- SEXMale or Female
- EDUC_ATTAINMENTHighest level of education attained ('Less than high school' or 'High school or beyond') derived from SCHL variable in ACS microdata and collapsed to a smaller number of categories.
- PWGTPWeights for the full-sample
- PWGTP1-PWGTP8080 columns of replicate weights created using the Successive Differences Replication (SDR) method.

Examples

```
data(lou_pums_microdata)

# Prepare the data for analysis with the survey package
  library(survey)

  lou_pums_rep_design <- survey::svrepdesign(
    data = lou_pums_microdata,
    variables = ~ UNIQUE_ID + AGE + SEX + RACE_ETHNICITY + EDUC_ATTAINMENT,
    weights = ~ PWGTP, repweights = "PWGTP\\d{1,2}",
    type = "successive-difference",
    mse = TRUE
  )

# Estimate population proportions
  svymean(~ SEX, design = lou_pums_rep_design)
```

lou_vax_survey *Louisville Vaccination Survey*

Description

A survey measuring Covid-19 vaccination status and a handful of demographic variables, based on a simple random sample of 1,000 residents of Louisville, Kentucky with an approximately 50% response rate.

These data were created using simulation.

Usage

```
data(lou_vax_survey)
```

Format

A data frame with 1,000 rows and 6 variables

RESPONSE_STATUS Response status to the survey ('Respondent' or 'Nonrespondent')

RACE_ETHNICITY Race and Hispanic/Latino ethnicity derived from RAC1P and HISP variables of ACS microdata and collapsed to a smaller number of categories.

SEX Male or Female

EDUC_ATTAINMENT Highest level of education attained ('Less than high school' or 'High school or beyond') derived from SCHL variable in ACS microdata and collapsed to a smaller number of categories.

VAX_STATUS Covid-19 vaccination status ('Vaccinated' or 'Unvaccinated')

SAMPLING_WEIGHT Sampling weight: equal for all cases since data come from a simple random sample

lou_vax_survey_control_totals *Control totals for the Louisville Vaccination Survey*

Description

Control totals to use for raking or post-stratification for the Louisville Vaccination Survey data. Control totals are population size estimates from the ACS 2015-2019 5-year Public Use Microdata Sample (PUMS) for specific demographic categories among adults in Jefferson County, KY.

These data were created using simulation.

Usage

```
data(lou_vax_survey_control_totals)
```

Format

A nested list object with two lists, `poststratification` and `raking`, each of which contains two elements: `estimates` and `variance-covariance`.

poststratification Control totals for the combination of `RACE_ETHNICITY`, `SEX`, and `EDUC_ATTAINMENT`.

- estimates: A numeric vector of estimated population totals.
- variance-covariance: A variance-covariance matrix for the estimated population totals.

raking Separate control totals for each of `RACE_ETHNICITY`, `SEX`, and `EDUC_ATTAINMENT`.

- estimates: A numeric vector of estimated population totals.
- variance-covariance: A variance-covariance matrix for the estimated population totals.

`redistribute_weights` *Redistribute weight from one group to another*

Description

Redistributes weight from one group to another: for example, from non-respondents to respondents. Redistribution is conducted for the full-sample weights as well as each set of replicate weights. This can be done separately for each combination of a set of grouping variables, for example to implement a nonresponse weighting class adjustment.

Usage

```
redistribute_weights(design, reduce_if, increase_if, by)
```

Arguments

`design`	A survey design object, created with either the `survey` or `srvyr` packages.
`reduce_if`	An expression indicating which cases should have their weights set to zero. Must evaluate to a logical vector with only values of TRUE or FALSE.
`increase_if`	An expression indicating which cases should have their weights increased. Must evaluate to a logical vector with only values of TRUE or FALSE.
`by`	(Optional) A character vector with the names of variables used to group the redistribution of weights. For example, if the data include variables named `"stratum"` and `"wt_class"`, one could specify `by = c("stratum", "wt_class")`.

Value

The survey design object, but with updated full-sample weights and updated replicate weights. The resulting survey design object always has its value of `combined.weights` set to `TRUE`.

References

See Chapter 2 of Heeringa, West, and Berglund (2017) or Chapter 13 of Valliant, Dever, and Kreuter (2018) for an overview of nonresponse adjustment methods based on redistributing weights.

- Heeringa, S., West, B., Berglund, P. (2017). Applied Survey Data Analysis, 2nd edition. Boca Raton, FL: CRC Press. "Applied Survey Data Analysis, 2nd edition." Boca Raton, FL: CRC Press.

- Valliant, R., Dever, J., Kreuter, F. (2018). "Practical Tools for Designing and Weighting Survey Samples, 2nd edition." New York: Springer.

Examples

```
# Load example data
suppressPackageStartupMessages(library(survey))
data(api)

dclus1 <- svydesign(id=~dnum, weights=~pw, data=apiclus1, fpc=~fpc)
dclus1$variables$response_status <- sample(x = c("Respondent", "Nonrespondent",
                                                 "Ineligible", "Unknown eligibility"),
                                           size = nrow(dclus1),
                                           replace = TRUE)
rep_design <- as.svrepdesign(dclus1)

# Adjust weights for cases with unknown eligibility
ue_adjusted_design <- redistribute_weights(
    design = rep_design,
    reduce_if = response_status %in% c("Unknown eligibility"),
    increase_if = !response_status %in% c("Unknown eligibility"),
    by = c("stype")
)

# Adjust weights for nonresponse
nr_adjusted_design <- redistribute_weights(
    design = ue_adjusted_design,
    reduce_if = response_status %in% c("Nonrespondent"),
    increase_if = response_status == "Respondent",
    by = c("stype")
)
```

`stack_replicate_designs` *Stack replicate designs, combining data and weights into a single object*

Description

Stack replicate designs: combine rows of data, rows of replicate weights, and the respective full-sample weights. This can be useful when comparing estimates before and after a set of adjustments made to the weights. Another more delicate application is when combining sets of replicate weights

from multiple years of data for a survey, although this must be done carefully based on guidance from a data provider.

Usage

```
stack_replicate_designs(..., .id = "Design_Name")
```

Arguments

`...`	Replicate-weights survey design objects to combine. These can be supplied in one of two ways. • Option 1 - A series of design objects, for example `'adjusted' = adjusted_design, 'orig' = orig_design`. • Option 2 - A list object containing design objects, for example `list('nr' = nr_adjusted_design, 'ue' = ue_adjusted_design)`. All objects must have the same specifications for `type`, `rho`, `mse`, `scales`, and `rscales`.
`.id`	A single character value, which becomes the name of a new column of identifiers created in the output data to link each row to the design from which it was taken. The labels used for the identifiers are taken from named arguments.

Value

A replicate-weights survey design object, with class `svyrep.design` and `svyrep.stacked`. The resulting survey design object always has its value of `combined.weights` set to `TRUE`.

Examples

```
# Load example data, creating a replicate design object
suppressPackageStartupMessages(library(survey))
data(api)

dclus1 <- svydesign(id=~dnum, weights=~pw, data=apiclus1, fpc=~fpc)
dclus1$variables$response_status <- sample(x = c("Respondent", "Nonrespondent",
                                                 "Ineligible", "Unknown eligibility"),
                                           size = nrow(dclus1),
                                           replace = TRUE)
orig_rep_design <- as.svrepdesign(dclus1)

# Adjust weights for cases with unknown eligibility
ue_adjusted_design <- redistribute_weights(
    design = orig_rep_design,
    reduce_if = response_status %in% c("Unknown eligibility"),
    increase_if = !response_status %in% c("Unknown eligibility"),
    by = c("stype")
)

# Adjust weights for nonresponse
nr_adjusted_design <- redistribute_weights(
```

```
    design = ue_adjusted_design,
    reduce_if = response_status %in% c("Nonrespondent"),
    increase_if = response_status == "Respondent",
    by = c("stype")
)

# Stack the three designs, using any of the following syntax options
stacked_design <- stack_replicate_designs(orig_rep_design, ue_adjusted_design, nr_adjusted_design,
                                          .id = "which_design")
stacked_design <- stack_replicate_designs('original' = orig_rep_design,
                                    'unknown eligibility adjusted' = ue_adjusted_design,
                                          'nonresponse adjusted' = nr_adjusted_design,
                                          .id = "which_design")
list_of_designs <- list('original' = orig_rep_design,
                        'unknown eligibility adjusted' = ue_adjusted_design,
                        'nonresponse adjusted' = nr_adjusted_design)
stacked_design <- stack_replicate_designs(list_of_designs, .id = "which_design")
```

`summarize_rep_weights` *Summarize the replicate weights*

Description

Summarize the replicate weights of a design

Usage

```
summarize_rep_weights(rep_design, type = "both", by)
```

Arguments

`rep_design`	A replicate design object, created with either the `survey` or `srvyr` packages.
`type`	Default is `"both"`. Use `type = "overall"`, for an overall summary of the replicate weights. Use `type = "specific"` for a summary of each column of replicate weights, with each column of replicate weights summarized in a given row of the summary. Use `type = "both"` for a list containing both summaries, with the list containing the names `"overall"` and `"both"`.
`by`	(Optional) A character vector with the names of variables used to group the summaries.

Value

If `type = "both"` (the default), the result is a list of data frames with names `"overall"` and `"specific"`. If `type = "overall"`, the result is a data frame providing an overall summary of the replicate weights.

The contents of the `"overall"` summary are the following:

- "nrows": Number of rows for the weights
- "ncols": Number of columns of replicate weights
- "degf_svy_pkg": The degrees of freedom according to the survey package in R
- "rank": The matrix rank as determined by a QR decomposition
- "avg_wgt_sum": The average column sum
- "sd_wgt_sums": The standard deviation of the column sums
- "min_rep_wgt": The minimum value of any replicate weight
- "max_rep_wgt": The maximum value of any replicate weight

If `type = "specific"`, the result is a data frame providing a summary of each column of replicate weights, with each column of replicate weights described in a given row of the data frame. The contents of the `"specific"` summary are the following:

- "Rep_Column": The name of a given column of replicate weights. If columns are unnamed, the column number is used instead
- "N": The number of entries
- "N_NONZERO": The number of nonzero entries
- "SUM": The sum of the weights
- "MEAN": The average of the weights
- "CV": The coefficient of variation of the weights (standard deviation divided by mean)
- "MIN": The minimum weight
- "MAX": The maximum weight

Examples

```
# Load example data
suppressPackageStartupMessages(library(survey))
data(api)

dclus1 <- svydesign(id=~dnum, weights=~pw, data=apiclus1, fpc=~fpc)
dclus1$variables$response_status <- sample(x = c("Respondent", "Nonrespondent",
                                                 "Ineligible", "Unknown eligibility"),
                                           size = nrow(dclus1),
                                           replace = TRUE)
rep_design <- as.svrepdesign(dclus1)

# Adjust weights for cases with unknown eligibility
ue_adjusted_design <- redistribute_weights(
    design = rep_design,
    reduce_if = response_status %in% c("Unknown eligibility"),
    increase_if = !response_status %in% c("Unknown eligibility"),
    by = c("stype")
)

# Summarize replicate weights
```

```
summarize_rep_weights(rep_design, type = "both")

# Summarize replicate weights by grouping variables

summarize_rep_weights(ue_adjusted_design, type = 'overall',
                      by = c("response_status"))

summarize_rep_weights(ue_adjusted_design, type = 'overall',
                      by = c("stype", "response_status"))

# Compare replicate weights

rep_wt_summaries <- lapply(list('original' = rep_design,
                                'adjusted' = ue_adjusted_design),
                           summarize_rep_weights,
                           type = "overall")
print(rep_wt_summaries)
```

`svyby_repwts` *Compare survey statistics calculated separately from different sets of replicate weights*

Description

A modified version of the `svyby()` function from the `survey` package. Whereas `svyby()` calculates statistics separately for each subset formed by a specified grouping variable, `svyby_repwts()` calculates statistics separately for each replicate design, in addition to any additional user-specified grouping variables.

Usage

```
svyby_repwts(
  rep_designs,
  formula,
  by,
  FUN,
  ...,
  deff = FALSE,
  keep.var = TRUE,
  keep.names = TRUE,
  verbose = FALSE,
  vartype = c("se", "ci", "ci", "cv", "cvpct", "var"),
  drop.empty.groups = TRUE,
  return.replicates = FALSE,
  na.rm.by = FALSE,
  na.rm.all = FALSE,
  multicore = getOption("survey.multicore")
)
```

Arguments

rep_designs	The replicate-weights survey designs to be compared. Supplied either as: • A named list of replicate-weights survey design objects, for example `list('nr' = nr_adjusted_design, 'ue' = ue_adjusted_design)`. • A 'stacked' replicate-weights survey design object created by `stack_replicate_designs()`. The designs must all have the same number of columns of replicate weights, of the same type (bootstrap, JKn, etc.)
formula	A formula specifying the variables to pass to FUN
by	A formula specifying factors that define subsets
FUN	A function taking a formula and survey design object as its first two arguments. Usually a function from the survey package, such as svytotal or svymean.
...	Other arguments to FUN
deff	A value of TRUE or FALSE, indicating whether design effects should be estimated if possible.
keep.var	A value of TRUE or FALSE. If FUN returns a svystat object, indicates whether to extract standard errors from it.
keep.names	Define row names based on the subsets
verbose	If TRUE, print a label for each subset as it is processed.
vartype	Report variability as one or more of standard error, confidence interval, coefficient of variation, percent coefficient of variation, or variance
drop.empty.groups	If FALSE, report NA for empty groups, if TRUE drop them from the output
return.replicates	If TRUE, return all the replicates as the "replicates" attribute of the result. This can be useful if you want to produce custom summaries of the estimates from each replicate.
na.rm.by	If true, omit groups defined by NA values of the by variables
na.rm.all	If true, check for groups with no non-missing observations for variables defined by formula and treat these groups as empty
multicore	Use multicore package to distribute subsets over multiple processors?

Value

An object of class "svyby": a data frame showing the grouping factors and results of FUN for each combination of the grouping factors. The first grouping factor always consists of indicators for which replicate design was used for an estimate.

Examples

```
suppressPackageStartupMessages(library(survey))
data(api)

dclus1 <- svydesign(id=~dnum, weights=~pw, data=apiclus1, fpc=~fpc)
dclus1$variables$response_status <- sample(x = c("Respondent", "Nonrespondent",
```

```
                                                "Ineligible", "Unknown eligibility"),
                                            size = nrow(dclus1),
                                            replace = TRUE)
orig_rep_design <- as.svrepdesign(dclus1)

# Adjust weights for cases with unknown eligibility
ue_adjusted_design <- redistribute_weights(
    design = orig_rep_design,
    reduce_if = response_status %in% c("Unknown eligibility"),
    increase_if = !response_status %in% c("Unknown eligibility"),
    by = c("stype")
)

# Adjust weights for nonresponse
nr_adjusted_design <- redistribute_weights(
    design = ue_adjusted_design,
    reduce_if = response_status %in% c("Nonrespondent"),
    increase_if = response_status == "Respondent",
    by = c("stype")
)

# Compare estimates from the three sets of replicate weights

  list_of_designs <- list('original' = orig_rep_design,
                          'unknown eligibility adjusted' = ue_adjusted_design,
                          'nonresponse adjusted' = nr_adjusted_design)

  ##_ First compare overall means for two variables
  means_by_design <- svyby_repwts(formula = ~ api00 + api99,
                                  FUN = svymean,
                                  rep_design = list_of_designs)

  print(means_by_design)

  ##_ Next compare domain means for two variables
  domain_means_by_design <- svyby_repwts(formula = ~ api00 + api99,
                                         by = ~ stype,
                                         FUN = svymean,
                                         rep_design = list_of_designs)

  print(domain_means_by_design)

# Calculate confidence interval for difference between estimates

ests_by_design <- svyby_repwts(rep_designs = list('NR-adjusted' = nr_adjusted_design,
                                                  'Original' = orig_rep_design),
                               FUN = svymean, formula = ~ api00 + api99)

differences_in_estimates <- svycontrast(stat = ests_by_design, contrasts = list(
  'Mean of api00: NR-adjusted vs. Original' = c(1,-1,0,0),
  'Mean of api99: NR-adjusted vs. Original' = c(0,0,1,-1)
))
```

```
print(differences_in_estimates)

confint(differences_in_estimates, level = 0.95)
```

Index