Package ‘surveysd’

March 19, 2018

Type Package

Title Survey Standard Error Estimation for Cumulated Estimates and their Differences in Complex Panel Designs

Version 0.1.0

Author Johannes Gussenbauer, Alexander Kowarik, Matthias Till

Maintainer Johannes Gussenbauer <Johannes.Gussenbauer@statistik.gv.at>

Description Estimate point estimates and their standard errors in complex household surveys using bootstrap replicates. Bootstraping considers survey design with rotating panel.

Encoding UTF-8

LazyData true

License GPL (>= 2)

Imports Rcpp (>= 0.12.12),data.table,simPop,matrixStats

LinkingTo Rcpp

RoxygenNote 6.0.1

R topics documented:

calc.stError 2
draw.bootstrap 6
generate.HHID 9
plot.surveysd 10
recalib 10
rescaled.bootstrap 12

Index **15**

calc.stError — *Calcualte point estimates and their standard errors using bootstrap weights.*

Description

Calculate point estimates as well as standard errors of variables in surveys. Standard errors are estimated using bootstrap weights (see draw.bootstrap and recalib). In addition the standard error of an estimate can be calcualted using the survey data for 3 or more consecutive years, which results in a reduction of the standard error.

Usage

```
calc.stError(dat,weights="RB050",b.weights=paste0("w",1:1000),year="RB010",var="HX080",
                        fun="weightedRatio",cross_var=NULL,year.diff=NULL,
                      year.mean=3,bias=FALSE,add.arg=NULL,size.limit=20,cv.limit=10)
```

Arguments

`dat` either data.frame or data.table containing the survey data. Surveys can be a panel survey or rotating panel survey, but does not need to be. For rotating panel survey bootstrap weights can be created using draw.bootstrap and recalib.

`weights` character specifying the name of the column in `dat` containing the original sample weights. Used to calculate point estimates.

`b.weights` character vector specifying the names of the columns in `dat` containing bootstrap weights. Used to calculate standard errors.

`year` character specifying the name of the column in `dat` containing the sample years.

`var` character vector containing variable names in `dat` on which `fun` shall be applied for each sample year.

`fun` character specifying the function which will be applied on `var` for each sample year. Possible arguments are `weightedRatio,weightedRatioNat,weightedSum,sampSize,popSize` as well as any other function which returns a double or integer and uses weights as its second argument.

`cross_var` character vectors or list of character vectors containig variables in `dat`. For each list entry `dat` will be split in subgroups according to the containing variables as well as `year`. The pointestimates are then estimated for each subgroup seperately. If `cross_var=NULL` the data will split into sample years by default.

`year.diff` character vectors, defining years for which the differences in the point estimate as well it's standard error is calculated. Each entry must have the form of `"year1 - year2"`. Can be NULL

`year.mean` integer, defining the range of years over which the sample mean of point estimates is additionally calcualted.

`bias` boolean, if TRUE the sample mean over the point estimates of the bootstrap weights is returned.

`add.arg` character specifying additional arguments for `fun`. Can be `NULL`.

`size.limit` integer defining a lower bound on the number of observations on `dat` in each group defined by `year` and the entries in `cross_var`. Warnings are returned if the number of observations in a subgroup falls below `size.limit`. In addition the concerned groups are available in the function output.

`cv.limit` non-negativ value defining a upper bound for the standard error in relation to the point estimate. If this relation exceed `cv.limit`, for a point estimate, they are flagged and available in the function output.

Details

`calc.stError` takes survey data (`dat`) and returns point estimates as well as their standard Errors defined by `fun` and `var` for each sample year in `dat`. `dat` must be household data where household members correspond to multiple rows with the same household identifier. The data should at least containt the following columns:

- Column indicating the sample year;
- Column indicating the household ID;
- Column containing the household sample weights;
- Columns which contain the bootstrap weights (see output of `recalib`);
- Columns listed in `var` as well as in `cross_var`

For each variable in `var` as well as sample year the function `fun` is applied using the original as well as the bootstrap sample weights.
The point estimate is then selected as the result of `fun` when using the original sample weights and it's standard error is estimated with the result of `fun` using the bootstrap sample weights.

`fun` can be any function which returns a double or integer and uses sample weights as it's second argument. The predifined options are `weightedRatio`,`weightedSum`,`sampSize` and `popSize`, for wich `sampSize` and `popSize` indicate the sample and population size respectively.

For the option `weightedRatio` a weighted ratio (in %) of `var` is calculated for `var` equal to 1, e.g `sum(weight[var==1])/sum(weight[!is.na(var)])*100`.
Using the option `weightedRatioNat` the weighted ratio (in %) is divided by the weighted ratio at the national level for each `year`.
If `cross_var` is not `NULL` but a vector of variables from `dat` then `fun` is applied on each subset of `dat` defined by all combinations of values in `cross_var`.
For instance if `cross_var = "sex"` with "sex" having the values "Male" and "Female" in `dat` the point estimate and standard error is calculated on the subsets of `dat` with only "Male" or "Female" value for "sex". This is done for each value of `year`. For variables in `cross_var` which have `NA`s in `dat` the rows containing the missings will be discarded.
When `cross_var` is a list of character vectors, subsets of `dat` and the following estimation of the point estimate, including the estimate for the standard error, are calculated for each list entry.

When defining `year.diff` the difference of point estimates between years as well their standard errors are calculated.
The entries in `year.diff` must have the form of `"year1 - year2"` which means that the results of the point estimates for `year2` will be substracted from the results of the point estimates for `year1`.

Specifying year.mean leads to an improvement in standard error by averaging the results for the point estimates, using the bootstrap weights, over year.mean years. Setting, for instance, year.mean = 3 the results in averaging these results over each consecutive set of 3 years.
Estimating the standard error over these averages gives an improved estimate of the standard error for the central year, which was used for averaging.
The averaging of the results is also applied in differences of point estimates. For instance defining year.diff = "2015-2009" and year.mean = 3 the differences in point estimates of 2015 and 2009, 2016 and 2010 as well as 2017 and 2011 are calcualated and finally the average over these 3 differences is calculated. The years set in year.diff are always used as starting years from which year.mean-1 consecutive years are used to build the average.
Setting bias to TRUE returns the calculation of a mean over the results from the bootstrap replicates. In the output the corresponding columns is labeled *_mean* at the end.

If fun needs more arguments they can be set in add.arg.

The parameter size.limit indicates a lower bound of the sample size for subsets in dat created by cross_var. If the sample size of a subset falls below size.limit a warning will be displayed. In addition all subsets for which this is the case can be selected from the output of calc.stError with $smallGroups.
With the parameter cv.limit one can set an upper bound on the coefficient of variantion. Estimates which exceed this bound are flagged with TRUE and are available in the function output with $cvHigh. cv.limit must be a positive integer and is treated internally as %, e.g. for cv.limit=1 the estimate will be flagged if the coefficient of variantion exceeds 1%.

When specifying year.mean, the decrease in standard error for choosing this method is internally calcualted and a rough estimate for an implied increase in sample size is available in the output with $stEDecrease. The rough estimate for the increase in sample size uses the fact that for a sample of size n the sample estimate for the standard error of most point estimates converges with a factor $1/\sqrt{n}$ against the true standard error σ.

Value

Returns a list containing:

- Estimates: data.table containing yearly, differences and/or k year averages for estimates of fun applied to var as well as the corresponding standard errors, which are calculated using the bootstrap weights.
- smallGroups: data.table containing groups for which the number of observation falls below size.limit.
- cvHigh: data.table containing a boolean variable which indicates for each estimate if the estimated standard error exceeds cv.limit.
- stEDecrease: data.table indicating for each estimate the theoretical increase in sample size which is gained when averaging over k years. Only returned if year.mean is not NULL.

Author(s)

Johannes Gussenbauer, Alexander Kowarik, Statistics Austria

See Also

draw.bootstrap
recalib

Examples

```
library(data.table)
# run on EU-SILC UDB data for Austria
# dat_at <- fread("path//to//austrian//data.csv")

dat_boot <- draw.bootstrap(dat=dat_at,REP=250,hid="DB030",weights="RB050",strata="DB040",
                           year="RB010",split=TRUE,pid="RB030")

# calibrate weight for bootstrap replicates
dat_boot_calib <- recalib(dat=copy(dat_boot),hid="DB030",weights="RB050",
                     year="RB010",b.rep=paste0("w",1:250),conP.var=c("RB090"),conH.var = c("DB040"))

# estimate weightedRatio for HX080 per year
err.est <- calc.stError(dat,weights="RB050",b.weights=paste0("w",1:250),year="RB010",var="HX080",
                       fun="weightedRatio",cross_var=NULL,year.diff=NULL,year.mean=NULL)

# estimate weightedRatio for HX080 per year and RB090
cross_var <- "RB090"
err.est <- calc.stError(dat,weights="RB050",b.weights=paste0("w",1:250),year="RB010",var="HX080",
                   fun="weightedRatio",cross_var=cross_var,year.diff=NULL,year.mean=NULL)


# use average over 3 years for standard error estimation
err.est <- calc.stError(dat,weights="RB050",b.weights=paste0("w",1:250),year="RB010",var="HX080",
                     fun="weightedRatio",cross_var=cross_var,year.diff=NULL,year.mean=3)

# get estimate for difference of year 2016 and 2013
year.diff <- c("2015-2009")
err.est <- calc.stError(dat,weights="RB050",b.weights=paste0("w",1:250),year="RB010",var="HX080",
                   fun="weightedRatio",cross_var=cross_var,year.diff=year.diff,year.mean=3)

# apply function to multiple variables and define different subsets
var <- c("HX080","arose")
cross_var <- list("RB090","DB040",c("RB090","DB040"))
err.est <- calc.stError(dat,weights="RB050",b.weights=paste0("w",1:250),year="RB010",var="HX080",
                   fun="weightedRatio",cross_var=cross_var,year.diff=year.diff,year.mean=3)

# use a function from an other package that has sampling weights as its second argument
# for example ging() from laeken
library(laeken)

# set up help function that returns only the gini index
help_gini <- function(x,w){
 return(gini(x,w)$value)
}
```

```
err.est <- calc.stError(dat,weights="RB050",b.weights=paste0("w",1:250),year="RB010",var="HX090",
                    fun="help_gini",cross_var=cross_var,year.diff=year.diff,year.mean=3)

# exporting data
# get point estimates
results <- err.est$Estimates
write2.csv(results,file="My_Results.csv",row.names=FALSE)
```

draw.bootstrap *Draw bootstrap replicates*

Description

Draw bootstrap replicates from survey data with rotating panel design. Survey information, like ID, sample weights, strata and population totals per strata, should be specified to ensure meaningfull survey bootstraping.

Usage

```
draw.bootstrap(dat,REP=1000,hid="DB030",weights="RB050",strata="DB040",
                        year="RB010",totals=NULL,boot.names=NULL)
```

Arguments

`dat` either data.frame or data.table containing the survey data with rotating panel design.

`REP` integer indicating the number of bootstrap replicates.

`hid` character specifying the name of the column in `dat` containing the household ID.

`weights` character specifying the name of the column in `dat` containing the sample weights.

`year` character specifying the name of the column in `dat` containing the sample years.

`strata` character vector specifying the name of the column in `dat` by which the population was stratified. If `strata` is a vector stratification will be assumed as the combination of column names contained in `strata`. Setting in addition `cluter` not NULL stratification will be assumed on multiple stages, where each additional entry in `strata` specifies the stratification variable for the next lower stage. see Details for more information.

`cluster` character vector specifying cluster in the data. If `NULL` household ID is taken es the lowest level cluster.

`totals` character specifying the name of the column in `dat` containing the the totals per strata and/or cluster. Is ONLY optional if `cluster` is `NULL` or equal `hid` and `strata` contains one columnname! Then the households per strata will be calcualted using the `weights` argument. If clusters and strata for multiple stages are specified `totals` needs to be a vector of `length(strata)` specifying the column on `dat` that contain the total number of PSUs at each stage. `totals` is interpreted from left the right, meaning that the first argument corresponds to the number of PSUs at the first and the last argument to the number of PSUs at the last stage.

`boot.names` character indicating the leading string of the column names for each bootstrap replica. If NULL defaults to "w".

`country` character specifying the name of the column in `dat` containing the country name. Is only used if `dat` contains data from multiple countries. In this case the bootstep procedure will be applied on each country seperately. If `country=NULL` the household identifier must be unique for each household.

`split` logical, if TRUE split households are considered using `pid`, for more information see Details.

`pid` column in `dat` specifying the personal identifier. This identifier needs to be unique for each person throught the whole data set.

Details

`draw.bootstrap` takes `dat` and draws `REP` bootstrap replicates from it. `dat` must be household data where household members correspond to multiple rows with the same household identifier. The data should at least containt the following columns:

- Column indicating the sample year;
- Column indicating the household ID;
- Column containing the household sample weights;
- Columns by which population was stratified during the sampling process.

For single stage sampling design a column the argument `totals` is optional, meaning that a column of the number of PSUs at the first stage does not need to be supplied. For this case the number of PSUs is calculated and added to `dat` using `strata` and `weights`. By setting `cluster` to NULL single stage sampling design is always assumed and if `strata` contains of multiple column names the combination of all those column names will be used for stratification.
In the case of multi stage sampling design the argument `totals` needs to be specified and needs to have the same number of arguments as `strata`.

If `cluster` is `NULL` or does not contain the `hid` at the last stage `hid` it will automatically be used as the final cluster. If, besides `hid`, clustering in additional stages is specified the number of column names in `strata` and `cluster` (including `hid`) must be the same. If for any stage there was no clustering or stratification one can set "1" or "I" for this stage.
For example `strata=c("REGION","I"),cluster=c("MUNICIPALITY","HID")` would speficy a 2 stage sampling design where at the first stage the municipalities where drawn stratified by regions and at the 2nd stage housholds are drawn in each municipality without stratification.

The bootstrap replicates are drawn for each survey year (`year`) using the function `bootstrap`. Afterwards the bootstrap replicates for each household are carried forward from the first year the household enters the survey to all the censecutive years it stays in the survey.
This ensures that the bootstrap replicates follow the same logic as the sampled households, making the bootstrap replicates more comparable to the actual sample units.

If `split` ist set to `TRUE` and `pid` is specified, the bootstrap replicates are carried forward using the personal identifiers instead of the houshold identifier. This takes into account the issue of a houshold splitting up. Any person in this new split household will get the same bootstrap replicate as the person that has come from an other household in the survey. People who enter already existing households will also get the same bootstrap replicate as the other households members had in the previous years.

Value

the survey data with the number of REP bootstrap replicates added as columns.

Returns a data.table containing the original data as well as the number of `REP` columns containing the bootstrap replicates for each repetition.
The columns of the bootstrap replicates are by default labeled "w*Number*" where *Number* goes from 1 to `REP`. If the column names of the bootstrap replicates should start with a different character or string the parameter `boot.names` can be used.

Author(s)

Johannes Gussenbauer, Alexander Kowarik, Statistics Austria

See Also

`data.table` for more information on data.table objects.

Examples

```
library(data.table)

# run on UDB SILC-data

# example for SILC-data for Spain
# dat_es <- fread("path//to//spanish//data.csv")

# approximate Number of clusters if not known
strata <- dat_es[,.(STRATA_sum=sum(RB050[!duplicated(DB030)])),by=list(DB050,RB010)]
strata[,STRATA_ratio:=STRATA_sum/sum(STRATA_sum),by=RB010]
strata[,N.cluster:=random_round(STRATA_ratio*35917),by=RB010]
strata[,N.households:=STRATA_sum/N.cluster]#'
dat_es <- merge(dat_es,strata[,.(DB050,RB010,N.cluster,N.households)],by=c("DB050","RB010"))

dat_boot <- draw.bootstrap(dat=dat_es,REP=250,hid="DB030",weights="RB050",strata=c("DB050","I"),cluster="DB060"
                      year="RB010",totals=c("N.cluster","N.households"),split=TRUE,pid="RB030")

# example for SILC-data for Austria
# dat_at <- fread("path//to//austrian//data.csv")
dat_boot <- draw.bootstrap(dat=dat_at,REP=250,hid="DB030",weights="RB050",strata="DB040",
                           year="RB010",split=TRUE,pid="RB030")


# save bootstrap replicates as .RData
save(dat_boot,file="dat_replicates.RData")
```

```
# or .csv-file
write.csv2(dat_boot,file="dat_replicates.csv",row.names=FALSE)
```

generate.HHID *Generate new houshold ID for survey data with rotating panel design taking into account split households*

Description

Generating a new houshold ID for survey data using a houshold ID and a personal ID. For surveys with rotating panel design containing housholds, houshold members can move from an existing household to a new one, that was not originally in the sample. This leads to the creation of so called split households. Using a peronal ID (that stays fixed over the whole survey), an indicator for different time steps and a houshold ID, a new houshold ID is assigned to the original and the split household.

Usage

```
generate.HHID(dat, time.step = "RB010", pid = "RB030", hid = "DB030")
```

Arguments

`dat`	data table of data frame containing the survey data
`time.step`	column name of `dat` containing an indicator for the rotations, e.g years, quarters, months, ect...
`pid`	column name of `dat` containing the personal identifier. This needs to be fixed for an indiviual throught the whole survey
`hid`	column name of `dat` containing the household id. This needs to for a household throught the whole survey

Value

the survey data `dat` as data.table object containing a new and an old household ID. The new household ID which considers the split households is now named `hid` and the original household ID has a trailing "_orig".

plot.surveysd *Plot surveysd-Objects*

Description

Plot results of `calc.stError()`

Usage

```
## S3 method for class 'surveysd'
plot(dat, variable = dat$param$var[1],
  type = c("summary", "grouping"), groups = NULL, sd.type = c("dot",
  "ribbon"))
```

Arguments

`dat`	object of class 'surveysd' output of function `calc.stError`
`variable`	Name of the variable for which standard errors have been calcualated in `dat`
`type`	can bei either 'summary' or 'grouping', default value is 'summary'. For 'summary' a barplot is created giving an overview of the number of estimates having the flag `smallGroup`, `cvHigh`, both or none of them. For 'grouping' results for point estimate and standard error are plotted for pre defined groups.
`groups`	If `type='grouping'` variables must be defined by which the data is grouped. Only 2 levels are supported as of right now. If only one group is defined the higher group will be the estimate over the whole year. Results are plotted for the first argument in `groups` as well as for the combination of `groups[1]` and `groups[2]`.
`sd.type`	can bei either `'ribbon'` or `'dot'` and is only used if `type='grouping'`. Default is `"dot"` For `sd.type='dot'` point estimates are plotted and flagged if the corresponding standard error and/or the standard error using the mean over k-years exceeded the value `cv.limit` (see `calc.stError`). For `sd.type='ribbon'` the point estimates including ribbons, defined by point estimate +- estimated standard error are plotted. The calculated standard errors using the mean over k years are plotted using less transparency. Results for the higher level (~`groups[1]`) are coloured grey.

recalib *Calibrate weights*

Description

Calibrate weights for bootstrap replicates by using iterative proportional updating to match population totals on various household and personal levels.

Usage

```
recalib(dat,hid="DB030",weights="RB050",b.rep=paste0("w",1:1000),year="RB010",
              country=NULL,conP.var=c("RB090"),conH.var=c("DB040","DB100"),...)
```

Arguments

`dat` either data.frame or data.table containing the sample survey for various years.

`hid` character specifying the name of the column in `dat` containing the household ID.

`weights` character specifying the name of the column in `dat` containing the sample weights.

`b.rep` character specifying the names of the columns in `dat` containing bootstrap weights which should be recalibratet

`year` character specifying the name of the column in `dat` containing the sample years.

`country` character specifying the name of the column in `dat` containing the country name. Is only used if `dat` contains data from multiple countries. In this case the calibration procedure will be applied on each country seperately. If `country=NULL` the household identifier must be unique for each household.

`conP.var` character vector containig person-specific variables to which weights should be calibrated. for which contingency tables for the population tables are calculatet per `year` and

`conH.var` character vector containig household-specific variables to which weights should be calibrated.

`...` additional arguments passed on to function `ipu2` from the `simPop` package.

Details

`recalib` takes survey data (`dat`) containing the bootstrap replicates generated by `draw.bootstrap` and calibrates weights for each bootstrap replication according to population totals for person- or household-specific variables.
`dat` must be household data where household members correspond to multiple rows with the same household identifier. The data should at least containt the following columns:

- Column indicating the sample year;
- Column indicating the household ID;
- Column containing the household sample weights;
- Columns which contain the bootstrap replicates (see output of `draw.bootstrap`);
- Columns indicating person- or household-specific variables for which sample weight should be adjusted.

For each year and each variable in `conP.var` and/or `conH.var` contingency tables are estimated to get margin totals on personal- and/or household-specific variables in the population.
Afterwards the bootstrap replicates are multiplied with the original sample weight and the resulting product ist then adjusted using `ipu2` to match the previously calcualted contingency tables. In this process the columns of the bootstrap replicates are overwritten by the calibrated weights.

Value

Returns a data.table containing the survey data as well as the calibrated weights for the bootstrap replicates, which are labeled like the bootstrap replicates. If calibration of a bootstrap replicate does not converge the bootsrap weight is not returned and numeration of the returned bootstrap weights is reduced by one.

Author(s)

Johannes Gussenbauer, Alexander Kowarik, Statistics Austria

See Also

ipu2 for more information on iterative proportional fitting.

Examples

```
library(data.table)
# run on UDB SILC-data
# example for SILC-data for Austria
# dat_at <- fread("path//to//austrian//data.csv")

dat_boot <- draw.bootstrap(dat=dat_at,REP=250,hid="DB030",weights="RB050",strata="DB040",
                           year="RB010",split=TRUE,pid="RB030")


# calibrate weight for bootstrap replicates
dat_boot_calib <- recalib(dat=copy(dat_boot),hid="DB030",weights="RB050",
                    year="RB010",b.rep=paste0("w",1:250),conP.var=c("RB090"),conH.var = c("DB040"))


# calibrate on other variable
dat_boot_calib <- recalib(dat=copy(dat_boot),hid="DB030",weights="RB050",
                    year="RB010",b.rep=paste0("w",1:250),conP.var=c("RB090"),conH.var = c("HX080","DB100"))


# save calibrated bootstrap weights as .RData
save(dat_boot_calib,file="dat_calibweight.RData")
# or .csv-file
write.csv2(dat_boot_calib,file="dat_calibweight.csv",row.names=FALSE)
```

rescaled.bootstrap *Draw bootstrap replicates*

Description

Draw bootstrap replicates from survey data using the rescaled bootstrap for stratified multistage sampling, presented by Preston, J. (2009).

Usage

```
rescaled.bootstrap(dat,REP=1000,strata="DB050>1",cluster="DB060>DB030",
                        fpc="N.cluster>N.households",check.input=TRUE,single.PSU=c("merge"),
                              return.value=c("data"))
```

Arguments

`dat` either data frame or data table containing the survey sample

`REP` integer indicating the number of bootstraps to be drawn

`strata` string specifying the column name in `dat` that is used for stratification. For multistage sampling multiple column names can be specified by `strata=c("strata1>strata2>strata3")`. See Details for more information.

`cluster` string specifying the column name in `dat` that is used for clustering. For multistage sampling multiple column names can be specified by `cluster=c("cluster1>cluster2>cluster3` See Details for more information.

`fpc` string specifying the column name in `dat` that contains the number of PSUs at the first stage. For multistage sampling the number of PSUs at each stage must be specified by `strata=c("fpc1>fpc2>fpc3")`.

`single.PSU` either "merge" or "mean" defining how single PSUs need to be dealt with. For `single.PSU="merge"` single PSUs at each stage are merged with the strata or cluster with the next least number of PSUs. If multiple of those exist one will be select via random draw For `single.PSU="mean"` single PSUs will get the mean over all bootstrap replicates at the stage which did not contain single PSUs.

`return.value` either "data" or "replicates" specifying the return value of the function. For "data" the survey data is returned as class `data.table`, for "replicates" only the bootstrap replicates are returned as `data.table`.

`check.input` logical, if TRUE the input will be checked before applying the bootstrap procedure

Details

For specifying multistage sampling designs the column names in `strata`,`cluster` and `fpc` need to seperated by ">".
For multistage sampling the strings are read from left to right meaning that the column name before the first ">" is taken as the column for stratification/clustering/number of PSUs at the first and the column after the last ">" is taken as the column for stratification/clustering/number of PSUs at the last stage. If for some stages the sample was not stratified or clustered one must specify this by "1" or "I", e.g. `strata=c("strata1>I>strata3")` if there was no stratification at the second stage or `cluster=c("cluster1>cluster2>I")` if there were no clusters at the last stage.
The number of PSUs at each stage is not calculated internally and must be specified for any sampling design. For single stage sampling using stratification this can usually be done by adding over all sample weights of each PSU by each strata-code.
Spaces in each of the strings will be removed, so if column names contain spaces they should be renamed before calling this procedure!

Value

returns the complete data set including the bootstrap replicates or just the bootstrap replicates, depending on `return.value="data"` or `return.value="replicates"` respectively.

Author(s)

Johannes Gussenbauer, Statistics Austria

References

Preston, J. (2009). Rescaled bootstrap for stratified multistage sampling. Survey Methodology. 35. 227-234.

Examples

```
library(laeken)
library(data.table)

data("eusilc")
eusilc <- data.table(eusilc)

eusilc[!duplicated(db030),N.housholds:=sum(db090),by=db040]
rescaled.bootstrap(eusilc,REP=100,strata="db040",cluster="db030",fpc="N.households")

eusilc[,new_strata:=paste(db040,rb090,sep="_")]
eusilc[!duplicated(db030),N.housholds:=sum(db090),by=new_strata]
rescaled.bootstrap(eusilc,REP=100,strata=c("new_strata"),cluster="db030",fpc="N.households")
```

Index

∗Topic **manip**
    calc.stError, 2
∗Topic **survey**
    calc.stError, 2

bootstrap, *7*

calc.stError, 2, *10*

data.table, *8*
draw.bootstrap, *2*, *5*, 6, *11*

generate.HHID, 9

ipu2, *11*, *12*

plot.surveysd, 10

recalib, *2*, *3*, *5*, 10
rescaled.bootstrap, 12