# Package ‘srvyr’

**Type**  Package  
**Title**  'dplyr'-Like Syntax for Summary Statistics of Survey Data  
**Description**  Use piping, verbs like 'group_by' and 'summarize', and other 'dplyr' inspired syntactic style when calculating summary statistics on survey data using functions from the 'survey' package.

**Version**  0.4.0  
**Date**  2020-07-29  
**URL**  [http://gdfe.co/srvyr](http://gdfe.co/srvyr), [https://github.com/gergness/srvyr](https://github.com/gergness/srvyr)  
**BugReports**  [https://github.com/gergness/srvyr/issues](https://github.com/gergness/srvyr/issues)  
**Depends**  R (>= 3.1.2)  
**Imports**  dplyr (>= 0.7), magrittr, methods, rlang, survey, tibble, tidyselect  
**License**  GPL-2 | GPL-3  
**LazyData**  TRUE  
**Suggests**  convey, DBI, dbplyr, ggplot2, knitr, laeken, Matrix, markdown (>= 2.2.2), pander, RSQLite, survival, testthat  
**Encoding**  UTF-8  
**VignetteBuilder**  knitr  
**RoxygenNote**  7.1.1  
**NeedsCompilation**  no  
**Author**  Greg Freedman Ellis [aut, cre], Thomas Lumley [ctb], Tomasz Zóltak [ctb], Ben Schneider [aut, ctb]  
**Maintainer**  Greg Freedman Ellis <greg.freedman@gmail.com>  
**Repository**  CRAN  
**Date/Publication**  2020-07-30 09:30:02 UTC
as_survey can be used to create a tbl_svy using design information (as_survey_design), replicate weights (as_survey_rep), or a two phase design (as_survey_twophase), or an object created by the survey package.

Usage

as_survey(.data, ...)

## S3 method for class 'data.frame'
as_survey(.data, ...)

Create a tbl_svy from a data.frame
as_survey

## S3 method for class 'tbl_lazy'
as_survey(.data, ...)

## S3 method for class 'survey.design2'
as_survey(.data, ...)

## S3 method for class 'svyrep.design'
as_survey(.data, ...)

## S3 method for class 'twophase2'
as_survey(.data, ...)

Arguments

.data a data.frame or an object from the survey package
...
other arguments, see other functions for details

Details

See vignette("databases",package = "dplyr") for more information on setting up databases in dplyr.

Value

a tbl_svy

Examples

# Examples from ?survey::svydesign
library(survey)
library(dplyr)
data(api)

# stratified sample
dstrata <- apistrat %>%
  as_survey(strata = stype, weights = pw)

# Examples from ?survey::svrepdesign
data(scd)
# use BRR replicate weights from Levy and Lemeshow
scd$rep1 <- 2 * c(1, 0, 1, 0, 1, 0)
scd$rep2 <- 2 * c(1, 0, 0, 1, 0, 1)
scd$rep3 <- 2 * c(0, 1, 1, 0, 0, 1)
scd$rep4 <- 2 * c(0, 1, 0, 1, 1, 0)

tscrep <- scd %>%
  as_survey(type = "BRR", repweights = starts_with("rep"),
             combined_weights = FALSE)

# Examples from ?survey::twophase
# two-phase simple random sampling.
data(pbc, package="survival")

pbc <- pbc %>%
  mutate(randomized = !is.na(trt) & trt > 0,
         id = row_number())
d2pbc <- pbc %>%
  as_survey(id = list(id, id), subset = randomized)

# dplyr 0.7 introduced new style of NSE called quosures
# See `vignette("programming", package = "dplyr")` for details
st <- quo(stype)
wt <- quo(pw)
dstrata <- apiestrat %>%
  as_survey(strata = !!st, weights = !!wt)

---

as_survey_design

Create a tbl_svy survey object using sampling design

Description

Create a survey object with a survey design.

Usage

as_survey_design(.data, ...)

## S3 method for class 'data.frame'
as_survey_design(
  .data,
  ids = NULL,
  probs = NULL,
  strata = NULL,
  variables = NULL,
  fpc = NULL,
  nest = FALSE,
  check_strata = !nest,
  weights = NULL,
  pps = FALSE,
  variance = c("HT", "YG"),
  ...
)

## S3 method for class 'survey.design2'
as_survey_design(.data, ...)

## S3 method for class 'tbl_lazy'
as_survey_design(
  .data,
as_survey_design

    ids = NULL,
    probs = NULL,
    strata = NULL,
    variables = NULL,
    fpc = NULL,
    nest = FALSE,
    check_strata = !nest,
    weights = NULL,
    pps = FALSE,
    variance = c("HT", "YG"),
    ...
)

Arguments

.data A data frame (which contains the variables specified below)
... ignored
.ids Variables specifying cluster ids from largest level to smallest level (leaving the
argument empty, NULL, 1, or 0 indicate no clusters).
.probs Variables specifying cluster sampling probabilities.
.strata Variables specifying strata.
.variables Variables specifying variables to be included in survey. Defaults to all variables
in .data
.fpc Variables specifying a finite population correct, see svydesign for more details.
.nest If TRUE, relabel cluster ids to enforce nesting within strata.
.check_strata If TRUE, check that clusters are nested in strata.
.weights Variables specifying weights (inverse of probability).
.pps "brewer" to use Brewer’s approximation for PPS sampling without replacement.
"overton" to use Overton’s approximation. An object of class HR to use the
Hartley-Rao approximation. An object of class ppsmat to use the Horvitz-
Thompson estimator.
.variance For pps without replacement, use variance="YG" for the Yates-Grundy estima-
tor instead of the Horvitz-Thompson estimator

Details

If provided a data.frame, it is a wrapper around svydesign. All survey variables must be included in
the data.frame itself. Variables are selected by using bare column names, or convenience functions
described in select.

If provided a survey.design2 object from the survey package, it will turn it into a srvyr object, so
that srvyr functions will work with it

Value

An object of class tbl_svy
Examples

# Examples from survey::svydesign
library(survey)
data(api)

# stratified sample
dstrata <- apistrat %>%
  as_survey_design(strata = stype, weights = pw)

# one-stage cluster sample
dclus1 <- apiclus1 %>%
  as_survey_design(dnum, weights = pw, fpc = fpc)

# two-stage cluster sample: weights computed from population sizes.
dclus2 <- apiclus2 %>%
  as_survey_design(c(dnum, snum), fpc = c(fpc1, fpc2))

## multistage sampling has no effect when fpc is not given, so
## these are equivalent.
dclus2wr <- apiclus2 %>%
  dplyr::mutate(weights = weights(dclus2)) %>%
  as_survey_design(c(dnum, snum), weights = weights)

dclus2wr2 <- apiclus2 %>%
  dplyr::mutate(weights = weights(dclus2)) %>%
  as_survey_design(c(dnum), weights = weights)

## syntax for stratified cluster sample
## (though the data weren't really sampled this way)
apistrat %>% as_survey_design(dnum, strata = stype, weights = pw, 
  nest = TRUE)

## PPS sampling without replacement
data(election)
dpps <- election_pps %>%
  as_survey_design(fpc = p, pps = "brewer")

# dplyr 0.7 introduced new style of NSE called quosures
# See vignette("programming", package = "dplyr") for details
st <- quo(stype)
wtkw <- quo(pw)
dstrata <- apistrat %>%
  as_survey_design(strata = !!st, weights = !!wt)

as_survey_rep

Create a tbl_svy survey object using replicate weights

Description

Create a survey object with replicate weights.
Usage

as_survey_rep(.data, ...)

## S3 method for class 'data.frame'
as_survey_rep(
  .data,
  variables = NULL,
  repweights = NULL,
  weights = NULL,
  type = c("BRR", "Fay", "JK1", "JKn", "bootstrap", "other"),
  combined_weights = TRUE,
  rho = NULL,
  bootstrap_average = NULL,
  scale = NULL,
  rscales = NULL,
  fpc = NULL,
  fpctype = c("fraction", "correction"),
  mse = getOption("survey.replicates.mse"),
  ...
)

## S3 method for class 'tbl_lazy'
as_survey_rep(
  .data,
  variables = NULL,
  repweights = NULL,
  weights = NULL,
  type = c("BRR", "Fay", "JK1", "JKn", "bootstrap", "other"),
  combined_weights = TRUE,
  rho = NULL,
  bootstrap_average = NULL,
  scale = NULL,
  rscales = NULL,
  fpc = NULL,
  fpctype = c("fraction", "correction"),
  mse = getOption("survey.replicates.mse"),
  ...
)

## S3 method for class 'svyrep.design'
as_survey_rep(.data, ...)

## S3 method for class 'survey.design2'
as_survey_rep(
  .data,
  type = c("auto", "JK1", "JKn", "BRR", "bootstrap", "subbootstrap", "mrbbootstrap", "Fay"),
  rho = 0,
fpc = NULL,
fpc.type = NULL,
...
, compress = TRUE,
mse = getOption("survey.replicates.mse")
)

## S3 method for class 'tbl_svy'
as_survey_rep(
  .data,
  type = c("auto", "JK1", "JKn", "BRR", "bootstrap", "subbootstrap", "mrbbootstrap", "Fay"),
  rho = 0,
  fpc = NULL,
  fpctype = NULL,
  ..., 
  compress = TRUE,
  mse = getOption("survey.replicates.mse")
)

Arguments

.data A data frame (which contains the variables specified below)
... ignored
variables Variables to include in the design (default is all)
repweights Variables specifying the replication weight variables
weights Variables specifying sampling weights
type Type of replication weights
combined_weights TRUE if the repweights already include the sampling weights. This is usually the case.
rho Shrinkage factor for weights in Fay's method
bootstrap_average For type = "bootstrap", if the bootstrap weights have been averaged, gives the number of iterations averaged over.
scale, rscales Scaling constant for variance, see svrepdesign for more information.
fpc, fpctype Finite population correction information
mse if TRUE, compute variances based on sum of squares around the point estimate, rather than the mean of the replicates
compress if TRUE, store replicate weights in compressed form (if converting from design)

Details

If provided a data.frame, it is a wrapper around svrepdesign. All survey variables must be included in the data.frame itself. Variables are selected by using bare column names, or convenience functions described in select.
If provided a svyrep.design object from the survey package, it will turn it into a srvyr object, so that srvyr functions will work with it.

If provided a survey design (survey.design2 or tbl_svy), it is a wrapper around `as.svrepdesign`, and will convert from a survey design to replicate weights.

**Value**

An object of class `tbl_svy`

**Examples**

```r
# Examples from ?survey::svrepdesign()
library(survey)
library(dplyr)
data(scd)
# use BRR replicate weights from Levy and Lemeshow
scd <- scd %>%
  mutate(rep1 = 2 * c(1, 0, 1, 0, 1, 0),
         rep2 = 2 * c(1, 0, 0, 1, 0, 1),
         rep3 = 2 * c(0, 1, 1, 0, 0, 1),
         rep4 = 2 * c(0, 1, 0, 1, 1, 0))

scdrep <- scd %>%
  as_survey_rep(type = "BRR", repweights = starts_with("rep"),
                combined_weights = FALSE)

# dplyr 0.7 introduced new style of NSE called quosures
# See `vignette("programming", package = "dplyr")` for details
repwts <- quo(starts_with("rep"))
scdrep <- scd %>%
  as_survey_rep(type = "BRR", repweights = !!repwts,
                combined_weights = FALSE)
```

---

**as_survey_twophase**

Create a tbl_svy survey object using two phase design

**Description**

Create a survey object by specifying the survey’s two phase design. It is a wrapper around `twophase`. All survey variables must be included in the data.frame itself. Variables are selected by using bare column names, or convenience functions described in `select`.

**Usage**

```r
as_survey_twophase(.data, ...)
```

```r
## S3 method for class 'data.frame'
as_survey_twophase(```
as_survey_twophase

.data,
id,
strata = NULL,
probs = NULL,
weights = NULL,
fpc = NULL,
subset,
method = c("full", "approx", "simple"),
...)

## S3 method for class 'twophase2'
as_survey_twophase(.data, ...)

Arguments

.data A data frame (which contains the variables specified below)
...
ignored
id list of two sets of variable names for sampling unit identifiers
strata list of two sets of variable names (or NULLs) for stratum identifiers
probs list of two sets of variable names (or NULLs) for sampling probabilities
weights Only for method = "approx", list of two sets of variable names (or NULLs) for sampling weights
fpc list of two sets of variables (or NULLs for finite population corrections
subset bare name of a variable which specifies which observations are selected in phase 2
method "full" requires (much) more memory, but gives unbiased variance estimates for general multistage designs at both phases. "simple" or "approx" use less memory, and is correct for designs with simple random sampling at phase one and stratified randoms sampling at phase two. See twophase for more details.

Value

An object of class tbl_svy

Examples

# Examples from ?survey::twophase
# two-phase simple random sampling.
data(pbc, package="survival")
library(dplyr)

pbc <- pbc %>%
  mutate(randomized = !is.na(trt) & trt > 0,
         id = row_number())
d2pbc <- pbc %>%
  as_survey_twophase(id = list(id, id), subset = randomized)
```r
d2pbc %>% summarize(mean = survey_mean(bili))

# two-stage sampling as two-phase
library(survey)
data(mu284)

mu284_1 <- mu284 %>%
dplyr::slice(c(1:15, rep(1:5, n2[1:5] - 3))) %>%
mutate(id = row_number(),
       sub = rep(c(TRUE, FALSE), c(15, 34-15)))

dmu284 <- mu284 %>%
  as_survey_design(ids = c(id1, id2), fpc = c(n1, n2))

# first phase cluster sample, second phase stratified within cluster
d2mu284 <- mu284_1 %>%
  as_survey_twophase(id = list(id1, id), strata = list(NULL, id1),
                     fpc = list(n1, NULL), subset = sub)

dmu284 %>%
  summarize(total = survey_total(y1),
            mean = survey_mean(y1))

d2mu284 %>%
  summarize(total = survey_total(y1),
            mean = survey_mean(y1))

# dplyr 0.7 introduced new style of NSE called quosures
# See `vignette("programming", package = "dplyr")` for details
ids <- quo(list(id, id))
d2pbc <- pbc %>%
  as_survey_twophase(id = !!ids, subset = "randomized")
```

---

**as_tibble**  
*Coerce survey variables to a data frame (tibble)*

**Description**

Coerce survey variables to a data frame (tibble)

**Arguments**

```
x  
A tbl_svy object
```
cascadedatacollect

**Description**

cascade is similar to `summarise`, but calculates a summary statistics for the total of a group in addition to each group.

**Usage**

cascade(.data, ..., .dots, .fill = NA)

**Arguments**

- `.data`, `tbl` A tbl_svy object
- `...` Name-value pairs of summary functions
- `.dots` Used to work around non-standard evaluation. See vignette("nse", package = "dplyr") for details.
- `.fill` Value to fill in for group summaries

**Examples**

```r
library(survey)
data(api)

dstrata <- apistrat %>%
  as_survey_design(strata = stype, weights = pw)

dstrata_grp <- dstrata %>%
  group_by(stype)

dstrata_grp %>%
cascade(api99 = survey_mean(api99),
        api00 = survey_mean(api00),
        api_diff = survey_mean(api00 - api99))
```

**collect**

**Description**

collect retrieves data from a database query (and when run on a tbl_svy object adjusts weights accordingly). Use collect when you want to run a function from the survey package on a srvyr db backed object. compute stores results in a remote temporary table.
current_svy

---

**current_svy**  
*Get the survey data for the current context*

---

**Description**

This is a helper to allow srvyr’s syntactic style. In particular, it tells functions inside of a summarize call what survey to use. In general, users will not have to worry about getting (or setting) the current conext’s survey, unless they are trying to extend srvyr. See vignette("extending-srvyr") for more details.

**Usage**

`current_svy()`

**Value**

a tbl_svy (or error if called with no survey context)

---

**dpoty_svy**  
*Single table verbs from dpoty*

---

**Description**

These are data manipulation functions designed to work on tbl_svy objects.

**Details**

- `mutate` and `transmute` can add or modify variables. See `mutate` for more details.
- `select` and `rename` keep or rename variables. See `select` for more details.
- `pull` extracts a variable as a vector (whereas `select` returns a tbl_svy). See `pull` for more details.
- `filter` keeps certain observations. See `filter` for more details.
- `arrange` is not implemented for tbl_svy objects. Nor are any two table verbs such as `bind_rows`, `bind_cols` or any of the joins (full_join, left_join, etc.). These data manipulations may require modifications to the survey variable specifications and so cannot be done automatically. Instead, use dplyr to perform them while the data is still stored in data.frames.
get_var_est  
Get the variance estimates for a survey estimate

Description

This is a helper to allow srvyr's syntactic style. In general, users will not have to worry about getting survey variance estimates directly unless they are trying to extend srvyr. This function helps convert from the result of a survey function into a data.frame with an estimate and measures of variance around it in a way that summarize expects. See vignette("extending-srvyr") for more details.

Usage

get_var_est(
  stat,    
  vartype, 
  grps = "", 
  level = 0.95, 
  df = Inf, 
  pre_calc_ci = FALSE, 
  deff = FALSE 
)

Arguments

stat  A survey statistic object, usually the result of a function from the survey package or svyby.
vartype  A vector indicating which variance estimates to calculate (options are se for standard error, ci for confidence interval, var for variance or cv for coefficient of variation). Multiples are allowed.
grps  A vector indicating the names of the grouping variables for grouped surveys ("" indicates no groups).
level  One or more levels to calculate a confidence interval.
df  Degrees of freedom, many survey functions default to Inf, but srvyr functions generally default to the result of calling degf on the survey object.
pre_calc_ci  Whether the confidence interval is pre-calculated (as in svyciprop)
deff  Whether to return the design effect (calculated using survey::deff)

Value

a tbl_svy with the variables modified
groups

Get/set the grouping variables for tbl.

Description

These functions do not perform non-standard evaluation, and so are useful when programming against tbl objects. ungroup is a convenient inline way of removing existing grouping.

Arguments

x data tbl_df or tbl_svy object.

See Also

groups for information.

group_by

Group a (survey) dataset by one or more variables.

Description

Most data operations are useful when done on groups defined by variables in the dataset. The group_by function takes an existing table (or svy_table) and converts it to a grouped version, where operations are performed "by group".

Arguments

.data A tbl
... variables to group by. All tbls accept variable names, some will also accept functions of variables. Duplicated groups will be silently dropped.
.add By default, when add = FALSE, group_by will override existing groups. To instead add to the existing groups, use add = TRUE
.dots Used to work around non-standard evaluation. See vignette("nse",package = "dplyr") for details.

Details

See group_by for more information about grouping regular data tables.

On tbl_svy objects, group_by sets up the object for operations similar to those allowed in svyby.

See Also

group_by for information about group_by on normal data tables.
Examples

```r
# Examples of svy_tbl group_by
library(survey)
data(api)
dstrata <- apistrat %>%
  as_survey_design(strata = stype, weights = pw) %>%
  group_by(stype)

dstrata %>%
  summarise(api_diff = survey_mean(api00 - api99))
```

---

**set_survey_vars**

*Set the variables for the current survey variable*

**Description**

This is a helper to allow srvyr's syntactic style. In general, users will not have to worry about setting variables in a survey object unless they are trying to extend srvyr. This function helps convert a vector to a variable in the correct part of a survey object's structure so that functions can refer to it using the survey package's formula notation. See vignette("extending-srvyr") for more details.

**Usage**

```r
set_survey_vars(.svy, x, name = "__SRVYR_TEMP_VAR__", add = FALSE)
```

**Arguments**

- `.svy` A survey object
- `x` A vector to be included in the variables portion of the survey object
- `name` The name of the variable once it is added. Defaults to `"__SRVYR_TEMP_VAR__"`, which is formatted weirdly to avoid name collisions.
- `add` FALSE, the default, overwrite all current variables. If TRUE, will add this variable instead.

**Value**

a tbl_svy with the variables modified
srvyr

Description

The srvyr package provides a new way of calculating summary statistics on survey data, based on the dplyr package. There are three stages to using srvyr functions, creating a survey object, manipulating the data, and calculating survey statistics.

Functions to create a survey object

as_survey_design, as_survey_rep, and as_survey_twophase are used to create surveys based on a data.frame and design variables, replicate weights or two phase design respectively. Each is based on a function in the survey package (svydesign, svrepdesign, twophase), and it is easy to modify code that uses the survey package so that it works with the srvyr package. See vignette("srvyr_vs_survey") for more details.

The function as_survey will choose between the other three functions based on the arguments given to save some typing.

Functions to manipulate data in a survey object

Once you’ve created a survey object, you can manipulate the data as you would using dplyr with a data.frame. mutate modifies or creates a variable, select and rename select or rename variables, and filter keeps certain observations.

Note that arrange and two table verbs such as bind_rows, bind_cols, or any of the joins are not usable on survey objects because they might require modifications to the definition of your survey. If you need to use these functions, you should do so before you convert the data.frame to a survey object.

Functions to summarize a survey object

Now that you have your data set up correctly, you can calculate summary statistics. To get the statistic over the whole population, use summarise, or to calculate it over a set of groups, use group_by first.

You can calculate the mean, (with survey_mean), the total (survey_total), the quantile (survey_quantile), or a ratio (survey_ratio). By default, srvyr will return the statistic and the standard error around it in a data.frame, but with the vartype parameter, you can also get a confidence interval ("ci"), variance ("var"), or coefficient of variation ("cv").

Within summarise, you can also use unweighted, which calculates a function without taking into consideration the survey weighting.
srvy-se-deprecated  Deprecated SE versions of main srvyr verbs

Description

srvy has updated it’s standard evaluation semantics to match dplyr 0.7, so these underscore functions are no longer required (but are still supported for backward compatibility reasons). See se-deprecated or the dplyr vignette on programming (vignette("programming", package = "dplyr")) for more details.

Usage

as_survey_(.data, ...)

as_survey_design_(
  .data,
  ids = NULL,
  probs = NULL,
  strata = NULL,
  variables = NULL,
  fpc = NULL,
  nest = FALSE,
  check_strata = !nest,
  weights = NULL,
  pps = FALSE,
  variance = c("HT", "YG")
)

as_survey_rep_(
  .data,
  variables = NULL,
  repweights = NULL,
  weights = NULL,
  type = c("BRR", "Fay", "JK1", "JKn", "bootstrap", "other"),
  combined_weights = TRUE,
  rho = NULL,
  bootstrap_average = NULL,
  scale = NULL,
  rscales = NULL,
  fpc = NULL,
  fpctype = c("fraction", "correction"),
  mse = getOption("survey.replicates.mse")
)

as_survey_twophase_(
  .data,
  id,
srvyr-se-deprecated

```r
cascade_(.data, ..., .dots, .fill = NA)
```

### Arguments

- `.data` a data.frame or an object from the survey package
- `...` other arguments, see other functions for details
- `.data` Variables specifying cluster ids from largest level to smallest level (leaving the argument empty, NULL, 1, or 0 indicate no clusters).
- `.data` Variables specifying cluster sampling probabilities.
- `strata` Variables specifying strata.
- `.data` Variables specifying variables to be included in survey. Defaults to all variables in `.data`
- `.data` Variables specifying a finite population correct, see `svydesign` for more details.
- `.data` If TRUE, relabel cluster ids to enforce nesting within strata.
- `.data` If TRUE, check that clusters are nested in strata.
- `.data` Variables specifying weights (inverse of probability).
- `.data` "brewer" to use Brewer's approximation for PPS sampling without replacement. "overton" to use Overton's approximation. An object of class HR to use the Hartley-Rao approximation. An object of class ppsmat to use the Horvitz-Thompson estimator.
- `.data` For pps without replacement, use variance="YG" for the Yates-Grundy estimator instead of the Horvitz-Thompson estimator
- `.data` Variables specifying the replication weight variables
- `.data` Type of replication weights
- `.data` TRUE if the repweights already include the sampling weights. This is usually the case.
- `.data` Shrinkage factor for weights in Fay’s method
- `.data` For type = "bootstrap", if the bootstrap weights have been averaged, gives the number of iterations averaged over.
- `.data` Scaling constant for variance, see `svrepdesign` for more information.
- `.data` Scaling constant for variance, see `svrepdesign` for more information.
- `.data` Finite population correction information
summarise

if TRUE, compute variances based on sum of squares around the point estimate, rather than the mean of the replicates

id

list of two sets of variable names for sampling unit identifiers

subset

bare name of a variable which specifies which observations are selected in phase 2

method

"full" requires (much) more memory, but gives unbiased variance estimates for general multistage designs at both phases. "simple" or "approx" use less memory, and is correct for designs with simple random sampling at phase one and stratified randoms sampling at phase two. See twophase for more details.

.dots

Used to work around non-standard evaluation. See vignette("nse",package = "dplyr") for details.

.fill

Value to fill in for group summaries

summarise

Summarise multiple values to a single value.

Description

Summarise multiple values to a single value.

Arguments

.data
tbl A tbl_svy object

...
Name-value pairs of summary functions

.groups
Defaults to "drop_last" in srvyr meaning that the last group is peeled off, but if there are more groups they will be preserved. Other options are "drop", which drops all groups, "keep" which keeps all of them and "rowwise" which converts the object to a rowwise object (meaning calculations will be performed on each row).

Details

Summarise for tbl_svy objects accepts several specialized functions. Each of the functions a variable (or two, in the case of survey_ratio), from the data.frame and default to providing the measure and its standard error.

The argument vartype can choose one or more measures of uncertainty, se for standard error, ci for confidence interval, var for variance, and cv for coefficient of variation. level specifies the level for the confidence interval.

The other arguments correspond to the analagous function arguments from the survey package.

The available functions from srvyr are:

survey_mean Calculate the survey mean of the entire population or by groups. Based on svymean.
survey_total Calculate the survey total of the entire population or by groups. Based on svytotal.
**summarise_all**

- **survey_ratio** Calculate the ratio of 2 variables in the entire population or by groups. Based on `svyratio`.
- **survey_quantile** Calculate quantiles in the entire population or by groups. Based on `svyquantile`.
- **survey_median** Calculate the median in the entire population or by groups. `svyquantile`.
- **unweighted** Calculate an unweighted estimate as you would on a regular `tbl_df`. Based on `dplyr`'s `summarise`.

**Examples**

```r
library(survey)
data(api)

dstrata <- apistrat %>%
  as_survey_design(strata = stype, weights = pw)

dstrata %>%
  summarise(api99 = survey_mean(api99),
             api00 = survey_mean(api00),
             api_diff = survey_mean(api00 - api99))

dstrata_grp <- dstrata %>%
  group_by(stype)

dstrata_grp %>%
  summarise(api99 = survey_mean(api99),
             api00 = survey_mean(api00),
             api_diff = survey_mean(api00 - api99))
```

---

**summarise_all**

*Manipulate multiple columns.*

**Description**

See `summarize_all` for more details. *_each functions will be deprecated in favor of *._all/*._if/*._at functions.

---

**survey_mean**

*Calculate the mean and its variation using survey methods*

**Description**

Calculate means and proportions from complex survey data. A wrapper around `svymean`, or if `proportion = TRUE`, `svyciprop`. `survey_mean` should always be called from `summarise`. 
Usage

survey_mean(
  x,
  na.rm = FALSE,
  vartype = c("se", "ci", "var", "cv"),
  level = 0.95,
  proportion = FALSE,
  prop_method = c("logit", "likelihood", "asin", "beta", "mean"),
  deff = FALSE,
  df = NULL,
  .svy = current_svy(),
  ...
)

Arguments

x  A variable or expression, or empty
na.rm  A logical value to indicate whether missing values should be dropped
vartype  Report variability as one or more of: standard error ("se", default), confidence interval ("ci"), variance ("var") or coefficient of variation ("cv").
level  (For vartype = "ci" only) A single number or vector of numbers indicating the confidence level
proportion  Use methods to calculate the proportion that may have more accurate confidence intervals near 0 and 1. Based on svyciprop.
prop_method  Type of proportion method to use if proportion is TRUE. See svyciprop for details.
deff  A logical value to indicate whether the design effect should be returned.
df  (For vartype = "ci" only) A numeric value indicating the degrees of freedom for t-distribution. The default (NULL) uses degf, but Inf is the usual survey package’s default (except in svyciprop).
.svy  A tbl_svy object. When called from inside a summarize function the default automatically sets the survey to the current survey.
...  Ignored

Examples

library(survey)
data(api)

dstrata <- apistrat %>%
  as_survey_design(strata = stype, weights = pw)

dstrata %>%
  summarise(api99 = survey_mean(api99),
    api_diff = survey_mean(api00 - api99, vartype = c("ci", "cv")))
group_by(awards) %>%
  summarise(api00 = survey_mean(api00))

# Leave x empty to calculate the proportion in each group
dstrata %>%
  group_by(awards) %>%
  summarise(pct = survey_mean())

# Setting proportion = TRUE uses a different method for calculating confidence intervals
dstrata %>%
  summarise(high_api = survey_mean(api00 > 875, proportion = TRUE, vartype = "ci"))

# level takes a vector for multiple levels of confidence intervals
dstrata %>%
  summarise(api99 = survey_mean(api99, vartype = "ci", level = c(0.95, 0.65)))

# Note that the default degrees of freedom in srvyr is different from
# survey, so your confidence intervals might not be exact matches. To
# Replicate survey's behavior, use df = Inf
dstrata %>%
  summarise(srvyr_default = survey_mean(api99, vartype = "ci"),
            survey_default = survey_mean(api99, vartype = "ci", df = Inf))

comparison <- survey::svymean(~api99, dstrata)
confint(comparison) # survey's default
confint(comparison, df = survey::degf(dstrata)) # srvyr's default

---

**survey_quantile**  
*Calculate the quantile and its variation using survey methods*

**Description**

Calculate quantiles from complex survey data. A wrapper around `svyquantile`. `survey_quantile` and `survey_median` should always be called from `summarise`.

**Usage**

```r
survey_quantile(
  x,
  quantiles,
  na.rm = FALSE,
  vartype = NULL,
  level = 0.95,
  q_method = "linear",
  f = 1,
  interval_type = c("Wald", "score", "betaWald", "probability", "quantile"),
  ties = c("discrete", "rounded"),
  df = NULL,
)```

```r
group_by(awards) %>%
  summarise(api00 = survey_mean(api00))

# Leave x empty to calculate the proportion in each group
dstrata %>%
  group_by(awards) %>%
  summarise(pct = survey_mean())

# Setting proportion = TRUE uses a different method for calculating confidence intervals
dstrata %>%
  summarise(high_api = survey_mean(api00 > 875, proportion = TRUE, vartype = "ci"))

# level takes a vector for multiple levels of confidence intervals
dstrata %>%
  summarise(api99 = survey_mean(api99, vartype = "ci", level = c(0.95, 0.65)))

# Note that the default degrees of freedom in srvyr is different from
# survey, so your confidence intervals might not be exact matches. To
# Replicate survey's behavior, use df = Inf
dstrata %>%
  summarise(srvyr_default = survey_mean(api99, vartype = "ci"),
            survey_default = survey_mean(api99, vartype = "ci", df = Inf))

comparison <- survey::svymean(~api99, dstrata)
confint(comparison) # survey's default
confint(comparison, df = survey::degf(dstrata)) # srvyr's default
```
survey_quantile

\.svy = current_svy(),
...
)

survey_median(
  x,
  na.rm = FALSE,
  vartype = c("se", "ci"),
  level = 0.95,
  q_method = "linear",
  f = 1,
  interval_type = c("Wald", "score", "betaWald", "probability", "quantile"),
  ties = c("discrete", "rounded"),
  df = NULL,
  .svy = current_svy(),
  ...
)

Arguments

x          A variable or expression
quantiles  A vector of quantiles to calculate
na.rm      A logical value to indicate whether missing values should be dropped
vartype    NULL to report no variability (default), otherwise one or more of: standard
            error ("se") confidence interval ("ci") (variance and coefficient of variation not
            available).
level      A single number indicating the confidence level (only one level allowed)
q_method   See "method" in approxfun
f          See approxfun
interval_type See svyquantile
ties       See svyquantile
df         A number indicating the degrees of freedom for t-distribution. The default, Inf
            uses the normal distribution (matches the survey package). Also, has no effect
            for type = "betaWald".
.svy       A tbl_svy object. When called from inside a summarize function the default
            automatically sets the survey to the current survey.
...        Ignored

Examples

library(survey)
data(api)
dstrata <- apistrat %>%
  as_survey_design(strata = stype, weights = pw)
survey_ratio

```r
dstrata %>%
  summarise(api99 = survey_quantile(api99, c(0.25, 0.5, 0.75)),
            api00 = survey_median(api00, vartype = c("ci")))
```

```r
dstrata %>%
  group_by(awards) %>%
  summarise(api00 = survey_median(api00))
```

**survey_ratio** Calculate the ratio and its variation using survey methods

**Description**
Calculate ratios from complex survey data. A wrapper around `svyratio`. `survey_ratio` should always be called from `summarise`.

**Usage**

```r
survey_ratio(
  numerator,
  denominator,
  na.rm = FALSE,
  vartype = c("se", "ci", "var", "cv"),
  level = 0.95,
  deff = FALSE,
  df = NULL,
  .svy = current_svy(),
  ...)
```

**Arguments**
- **numerator** The numerator of the ratio
- **denominator** The denominator of the ratio
- **na.rm** A logical value to indicate whether missing values should be dropped
- **vartype** Report variability as one or more of: standard error ("se", default), confidence interval ("ci"), variance ("var") or coefficient of variation ("cv").
- **level** A single number or vector of numbers indicating the confidence level
- **deff** A logical value to indicate whether the design effect should be returned.
- **df** (For vartype = "ci" only) A numeric value indicating the degrees of freedom for t-distribution. The default (NULL) uses `deff`, but Inf is the usual survey package's default (except in `svyciprop`.
- **.svy** A tbl_svy object. When called from inside a summarize function the default automatically sets the survey to the current survey.
- **...** Ignored
Examples

```r
library(survey)
data(api)

dstrata <- apistrat %>%
  as_survey_design(strata = stype, weights = pw)

dstrata %>%
  summarise(enroll = survey_ratio(api00, api99, vartype = c("ci", "cv")))

dstrata %>%
  group_by(awards) %>%
  summarise(api00 = survey_ratio(api00, api99))

# level takes a vector for multiple levels of confidence intervals
dstrata %>%
  summarise(enroll = survey_ratio(api99, api00, vartype = "ci", level = c(0.95, 0.65)))

# Note that the default degrees of freedom in srvyr is different from
# survey, so your confidence intervals might not exactly match. To
# replicate survey's behavior, use df = Inf
dstrata %>%
  summarise(srvyr_default = survey_total(api99, vartype = "ci"),
            survey_defualt = survey_total(api99, vartype = "ci", df = Inf))

comparison <- survey::svytotal(~api99, dstrata)
confint(comparison) # survey's default
confint(comparison, df = survey::degf(dstrata)) # srvyr's default
```

---

**survey_tally**

*Count/tally survey weighted observations by group*

**Description**

Analogous to `tally` and `count`, calculates the survey weighted count of observations. `survey_tally` will call `survey_total` empty (resulting in the count of each group) or on `wt` if it is specified (resulting in the survey weighted total of `wt`). `survey_count` is similar, but calls `group_by` before calculating the count and then returns the data to the original groupings.

**Usage**

```r
survey_tally(
  x,
  wt,
  sort = FALSE,
  name = "n",
  vartype = c("se", "ci", "var", "cv")
)
```
survey_tally

survey_count(
  x,
  ..., 
  wt = NULL,
  sort = FALSE,
  name = "n",
  .drop = dplyr::group_by_drop_default(x),
  vartype = c("se", "ci", "var", "cv")
)

Arguments

x A tbl_svy object, as created by as_survey and related functions.

wt (Optional) A variable to weight on (in addition to the survey weights, which are always used). If left unspecified, tally() will use a variable named "n" if one exists, but count() will not. Override this behavior by specifying wt = NULL.

sort Whether to sort the results (defaults to FALSE)

name Name of count variable created (defaults to n). If the variable already exists, will add "n" to the end until it does not.

vartype What types variation estimates to calculate, passed to survey_total.

... Variables to group by, passed to group_by().

.drop When .drop = TRUE, empty groups are dropped, see group_by documentation for more details.

Details

If n already exists, tally will use it as the weight, but count will not.

Examples

library(survey)
data(api)

dstrata <- apistrat %>%
  as_survey_design(strata = stype, weights = pw)

dstrata %>%
  group_by(awards) %>%
survey_tally()

dstrata %>%
survey_count(awards)
survey_total

Calculate the total and its variation using survey methods

Description

Calculate totals from complex survey data. A wrapper around svytotal. survey_total should always be called from summarise.

Usage

survey_total(
  x,
  na.rm = FALSE,
  vartype = c("se", "ci", "var", "cv"),
  level = 0.95,
  deff = FALSE,
  df = NULL,
  .svy = current_svy(),
  ...
)

Arguments

  x              A variable or expression, or empty
  na.rm          A logical value to indicate whether missing values should be dropped
  vartype        Report variability as one or more of: standard error ("se", default), confidence interval ("ci"), variance ("var") or coefficient of variation ("cv").
  level          A single number or vector of numbers indicating the confidence level
  deff           A logical value to indicate whether the design effect should be returned.
  df             (For vartype = "ci" only) A numeric value indicating the degrees of freedom for t-distribution. The default (NULL) uses df, but Inf is the usual survey package's default.
  .svy           A tbl_svy object. When called from inside a summarize function the default automatically sets the survey to the current survey.
  ...            Ignored

Examples

library(survey)
data(api)

dstrata <- apistrat %>%
  as_survey_design(strata = stype, weights = pw)

dstrata %>%
  summarise(enroll = survey_total(enroll),
Calculate the population variance and its variation using survey methods.

**survey_var**

**Description**

Calculate population variance from complex survey data. A wrapper around `svyvar`. `survey_var` should always be called from `summarise`.

**Usage**

```r
survey_var(  
  x,  
  na.rm = FALSE,  
  vartype = c("se", "ci", "var"),  
  level = 0.95,  
  df = NULL,  
  .svy = current_svy(),  
  ...
)
```

```r
survey_sd(x, na.rm = FALSE, .svy = current_svy(), ...)
```
Arguments

- **x**: A variable or expression, or empty
- **na.rm**: A logical value to indicate whether missing values should be dropped
- **vartype**: Report variability as one or more of: standard error ("se", default) or variance ("var") (confidence intervals and coefficient of variation not available).
- **level**: (For vartype = "ci" only) A single number or vector of numbers indicating the confidence level.
- **df**: (For vartype = "ci" only) A numeric value indicating the degrees of freedom for t-distribution. The default (Inf) is equivalent to using normal distribution and in case of population variance statistics there is little reason to use any other values (see Details).
- **.svy**: A tbl_svy object. When called from inside a summarize function the default automatically sets the survey to the current survey.
- **...**: Ignored

Details

Be aware that confidence intervals for population variance statistic are computed by package survey using t or normal (with df=Inf) distribution (i.e., symmetric distributions). **This could be a very poor approximation** if even one of these conditions is met:

- there are few sampling design degrees of freedom,
- analyzed variable isn’t normally distributed,
- there is huge variation in sampling probabilities of the survey design.

Because of this be very careful using confidence intervals for population variance statistics especially while performing analysis within subsets of data or using grouped survey objects.

Sampling distribution of the variance statistic in general is asymmetric (chi-squared in case of simple random sampling of normally distributed variable) and if analyzed variable isn’t normally distributed or there is huge variation in sampling probabilities of the survey design (or both) it could converge to normality only very slowly (with growing number of survey design degrees of freedom).

Examples

```r
library(survey)
data(api)
dstrata <- apistrat %>%
  as_survey_design(strata = stype, weights = pw)
dstrata %>%
  summarise(api99_var = survey_var(api99),
            api99_sd = survey_sd(api99))
dstrata %>%
  group_by(awards) %>%
  summarise(api00_var = survey_var(api00),
             api00_sd = survey_sd(api00))
```
api00_sd = survey_sd(api00)

# standard deviation and variance of the population variance estimator
# are available with vartype argument
# (but not for the population standard deviation estimator)
dstrata %>%
  summarise(api99_variance = survey_var(api99, vartype = c("se", "var")))

---

svychisq

**Chisquared tests of association for survey data.**

**Description**

Chisquared tests of association for survey data.

**Arguments**

- `formula` See details in `svychisq`
- `design` See details in `svychisq`
- `na.rm` See details in `svychisq`
- `...` See details in `svychisq`

---

**tbl_svy**

**tbl_svy object.**

**Description**

A `tbl_svy` wraps a locally stored svydesign and adds methods for dplyr single-table verbs like `mutate`, `group_by` and `summarise`. Create a `tbl_svy` using `as_survey_design`.

**Methods**

- `tbl_df` implements these methods from dplyr.

  - `select` or `rename` Select or rename variables in a survey’s dataset.
  - `mutate` or `transmute` Modify and create variables in a survey’s dataset.
  - `group_by` and `summarise` Get descriptive statistics from survey.
Examples

library(survey)
library(dplyr)
data(api)
svy <- as_survey_design(apistrat, strata = stype, weights = pw)
svy

# Data manipulation verbs --------------------------------------------------
filter(svy, pcttest > 95)
select(svy, starts_with("acs")) # variables used in survey design are automatically kept
summarise(svy, col.grad = survey_mean(col.grad))
mutate(svy, api_diff = api00 - api99)

# Group by operations -------------------------------------------------------
# To calculate survey
svy_group <- group_by(svy, dname)

summarise(svy, col.grad = survey_mean(col.grad),
          api00 = survey_mean(api00, vartype = "ci")
)

---

tbl_vars

**List variables produced by a tbl.**

Description

List variables produced by a tbl.

Arguments

- **x**
  
  A tbl object

unweighted

**Calculate an unweighted summary statistic from a survey**

Description

Calculate unweighted summaries from a survey dataset, just as on a normal data.frame with `summarise`.

Usage

`unweighted(x, .svy = current_svy(), ...)`

Arguments

- **x**
  
  A variable or expression

- **.svy**
  
  A tbl_svy object. When called from inside a summarize function the default automatically sets the survey to the current survey.

- **...**
  
  Ignored
Details

Uses tidy evaluation semantics and so if you want to use wrapper functions based on variable names, you must use tidy evaluation, see the examples here, documentation in nse-force, or the dplyr vignette called ‘programming’ for more information.

Examples

library(survey)
library(dplyr)
data(api)

dstrata <- apistrat %>%
  as_survey_design(strata = stype, weights = pw)

dstrata %>%
  summarise(api99_unw = unweighted(mean(api99)),
            n = unweighted(n()))

dstrata %>%
  group_by(stype) %>%
  summarise(api_diff_unw = unweighted(mean(api00 - api99)))

# If you want to use a wrapper function, be sure to treat
# non-standard evaluation correctly
umean <- function(x) {
  unweighted(mean({{x}}))
}
dstrata %>%
  group_by(stype) %>%
  summarize(api_diff_unw = umean(api00 - api99))
Index

all_vars (summarise_all), 21
any_vars (summarise_all), 21
approxfun, 24
as.svrepdesign, 9
as_survey, 2, 17
as_survey_ (srvyr-se-deprecated), 18
as_survey_design, 2, 4, 17, 31
as_survey_design_
    (srvyr-se-deprecated), 18
as_survey_rep, 2, 6, 17
as_survey_rep_ (srvyr-se-deprecated), 18
as_survey_twophase, 2, 9, 17
as_survey_twophase_
    (srvyr-se-deprecated), 18
as_tibble, 11
cascade, 12
cascade_ (srvyr-se-deprecated), 18
collect, 12
compute (collect), 12
current_svy, 13
degf, 22, 25, 28
dplyr_single, 13
filter, 13, 17
filter (dplyr_single), 13
filter_ (srvyr-se-deprecated), 18
filter_all (summarise_all), 21
filter_at (summarise_all), 21
filter_if (summarise_all), 21
funs (summarise_all), 21
funs_ (srvyr-se-deprecated), 18
get_var_est, 14
group_by, 15, 15, 17, 27, 31
group_by_ (group_by), 15
group_by_all (summarise_all), 21
group_by_at (summarise_all), 21
group_by_if (summarise_all), 21
group_vars (groups), 15
groups, 15, 15
mutate, 13, 17, 31
mutate (dplyr_single), 13
mutate_ (srvyr-se-deprecated), 18
mutate_all (summarise_all), 21
mutate_at (summarise_all), 21
mutate_each (summarise_all), 21
mutate_each_ (srvyr-se-deprecated), 18
mutate_if (summarise_all), 21
nse_force, 33
pull, 13
pull (dplyr_single), 13
rename, 17, 31
rename (dplyr_single), 13
rename_ (srvyr-se-deprecated), 18
rename_all (summarise_all), 21
rename_at (summarise_all), 21
rename_if (summarise_all), 21
select, 5, 8, 9, 13, 17, 31
select (dplyr_single), 13
select_ (srvyr-se-deprecated), 18
select_all (summarise_all), 21
select_at (summarise_all), 21
select_if (summarise_all), 21
set_survey_vars, 16
srvyr, 17
srvyr-se-deprecated, 18
summarise, 12, 17, 20, 21, 23, 25, 28, 29, 31, 32
summarise_ (srvyr-se-deprecated), 18
summarise_all, 21
summarise_at (summarise_all), 21
summarise_each (summarise_all), 21
summarise_each_ (srvyr-se-deprecated), 18
summarise_if(summarise_all), 21
summarize(summarise), 20
summarize_(srvyr-se-deprecated), 18
summarize_all, 21
summarize_all(summarise_all), 21
summarize_at(summarise_all), 21
summarize_each(summarise_all), 21
summarize_each_(srvyr-se-deprecated), 18
summarize_if(summarise_all), 21
survey_count(survey_tally), 26
survey_mean, 17, 20, 21
survey_median, 21
survey_median(survey_quantile), 23
survey_quantile, 17, 21, 23
survey_ratio, 17, 21, 25
survey_sd(survey_var), 29
survey_tally, 26
survey_total, 17, 20, 26, 27, 28
survey_var, 29
svrepdesign, 8, 17, 19
svyby, 15
svychisq, 31, 31
svyciprop, 21, 22, 25
svydesign, 5, 17, 19
svymean, 20, 21
svyquantile, 21, 23, 24
svyratio, 21, 25
svytotal, 20, 28
svyvar, 29
tally, 26
tbl_df, 15
tbl_svy, 15, 31
tbl_vars, 32
transmute, 31
transmute_(dplyr_single), 13
transmute_(srvyr-se-deprecated), 18
twophase, 9, 10, 17, 20
ungroup(groups), 15
unweighted, 17, 21, 32
vars(summarise_all), 21