Package 'srvyr'

August 19, 2024

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 1.3.0

Date 2024-08-19

URL http://gdfe.co/srvyr/, https://github.com/gergness/srvyr/

BugReports https://github.com/gergness/srvyr/issues

Imports dplyr (>= 1.1.0), magrittr, methods, rlang, survey (>= 4.1), tibble, tidyr, tidyselect, vctrs (>= 0.3.0)

License GPL-2 | GPL-3

Suggests spelling, convey, DBI, dbplyr, ggplot2, knitr, laeken, Matrix, rmarkdown (>= 2.2.2), pander, RSQLite, survival, testthat, covr

Encoding UTF-8

VignetteBuilder knitr

RoxygenNote 7.3.2

Language en-US

NeedsCompilation no

Author Greg Freedman Ellis [aut, cre], Thomas Lumley [ctb], Tomasz Żółtak [ctb], Ben Schneider [aut, ctb], Pavel N. Krivitsky [ctb]

Maintainer Greg Freedman Ellis <greg.freedman@gmail.com>

Repository CRAN

Date/Publication 2024-08-19 17:10:02 UTC

Contents

`as_survey` *Create a tbl_svy from a data.frame*

Description

`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 'tbl_svy'
as_survey(.data, ...)

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

## 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)

scdrep <- 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 <- apistrat %>%
  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,
  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 estimator 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)
wt <- 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", "successive-difference", "ACS",
    "other"),
  combined_weights = TRUE,
  rho = NULL,
  bootstrap_average = NULL,
  scale = NULL,
  rscales = NULL,
  fpc = NULL,
  fpctype = c("fraction", "correction"),
  mse = getOption("survey.replicates.mse"),
  degf = NULL,
  ...
)

## S3 method for class 'tbl_lazy'
as_survey_rep(
  .data,
  variables = NULL,
  repweights = NULL,
  weights = NULL,
 type = c("BRR", "Fay", "JK1", "JKn", "bootstrap", "successive-difference", "ACS",
    "other"),
  combined_weights = TRUE,
  rho = NULL,
```

```
  bootstrap_average = NULL,
  scale = NULL,
  rscales = NULL,
  fpc = NULL,
  fpctype = c("fraction", "correction"),
  mse = getOption("survey.replicates.mse"),
  degf = NULL,
  ...
)

## 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,
  fpctype = 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`
: Variables specifying a finite population correction, see `svrepdesign` for more details.

`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

`degf`
: Design degrees of freedom: a single number, or `NULL`, in which case a value will be computed automatically, which can be slow for very large data sets. See `svrepdesign` for more details.

`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

```
# 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

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

## S3 method for class 'data.frame'
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)

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

`cascade` *Summarise multiple values into cascading groups*

Description

`cascade` is similar to `summarise`, but calculates a summary statistics for the total of a group in addition to each group. The groupings are chosen by "unpeeling" from the end of the groupings, and also expanding out interactions to all terms (eg the interactions of all combinations of subsets of variables as well as each variable on it's own).

Usage

```
cascade(.data, ..., .fill = NA, .fill_level_top = FALSE, .groupings = NULL)
```

Arguments

`.data`	tbl A `tbl_svy` object
`...`	Name-value pairs of summary functions
`.fill`	Value to fill in for group summaries
`.fill_level_top`	When filling factor variables, whether to put the value '.fill' in the first position (defaults to FALSE, placing it in the bottom).
`.groupings`	(Experimental) A list of lists of quosures to manually specify the groupings to use, rather than the default.

Examples

```
library(survey)
data(api)

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

# Calculates the means by stype and also for the whole
# sample
dstrata %>%
  group_by(stype) %>%
  cascade(api99_mn = survey_mean(api99),
            api00_mn = survey_mean(api00),
            api_diff = survey_mean(api00 - api99))

# Calculates the proportions by the interaction of stype & awards
# as well as by each of those variable's groups alone, and finally
# the total as well
dstrata %>%
  group_by(interact(stype, awards)) %>%
  cascade(prop = survey_mean())

# Can also specify the .groupings manually, though this interface
# is a little ugly, as it requires passing a list of quosures or
# symbols you've created, rather than the usual syntax
dstrata %>%
  cascade(
    prop = survey_mean(),
    .groupings = list(rlang::quos(stype, awards), rlang::quos(NULL))
  )
```

`collect` *Force computation of a database query*

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.

`cur_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 (for the current group with `cur_svy()` or the complete survey for `cur_svy_full()`. In general, users will not have to worry about getting (or setting) the current context's survey, unless they are trying to extend srvyr. See `vignette("extending-srvyr")` for more details. `current_svy()` is deprecated, but returns the same value as `cur_svy()`.

Usage

```
cur_svy()

cur_svy_full()

current_svy()
```

Value

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

---

cur_svy_wts *Get the full-sample weights for the current context*

Description

This is a helper to allow srvyr's syntactic style. This function allows quick access to the full-sample weights for the current group, using `cur_svy_wts()`, See `vignette("extending-srvyr")` for more details.

Usage

```
cur_svy_wts()
```

Value

a numeric vector containing full-sample weights

Examples

```
data(api, package = 'survey')

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

dstrata %>%
  summarize(sum_of_weights = sum(cur_svy_wts()),
            kish_deff = var(cur_svy_wts())/(mean(cur_svy_wts())^2))
```

dplyr_filter_joins *Filtering joins from dplyr*

Description

These are data manipulation functions designed to work on a `tbl_svy` object and another data frame or `tbl_svy` object.

Details

`semi_join` and `anti_join` filter certain observations from a `tbl_svy` depending on the presence or absence of matches in another table. See `filter-joins` for more details.

Mutating joins (`full_join`, `left_join`, etc.) are not implemented for any `tbl_svy` objects. 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,
  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.

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

```
# 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))
```

---

group_map_dfr *Apply a function to each group*

Description

group_map(), group_walk and group_map_dfr are purrr-style functions that can be used to iterate on grouped survey objects (note that group_map_dfr replaces dplyr::group_modify because we are changing the data from a tbl_svy to a regular tibble).

Usage

```
group_map_dfr(.data, .f, ..., .keep = FALSE)

## S3 method for class 'tbl_svy'
group_map(.data, .f, ..., .keep = FALSE)

group_map_dfr(.data, .f, ..., .keep = FALSE)
```

Arguments

.data	A tbl_svy object
.f	A function or purrr-style formula to apply to each group
...	Other arguments passed to .f
.keep	Whether the grouping variables are kept when passed into .f

Value

For `group_map` a list, for `group_map_dfr` a 'tbl_df', and for `group_walk` invisibly the original `tbl_svy`.

Examples

```
data(api, package = "survey")
dstrata <- apistrat %>%
  as_survey_design(strata = stype, weights = pw)

results <- dstrata %>%
  group_by(both) %>%
  group_map(~survey::svyglm(api00~api99 + stype, .))

# group_map_dfr calls `bind_rows` on the list returned and includes
# grouping variables. This is most useful with a package like `broom`
# but could also be used with survey package functions.
result_coef <- dstrata %>%
  group_by(both) %>%
  group_map_dfr(
    ~data.frame(
      api99_coef = coef(survey::svyglm(api00~api99 + stype, .))[["api99"]]
    )
  )
```

`group_trim` *Single table verbs from dplyr and tidyr*

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`, `rename`, and `rename_with` 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.

#' `drop_na` drops observations containing missing values. See `drop_na` 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.

`interact` *Create interaction terms to group by when summarizing*

Description

Allows multiple grouping by multiple variables as if they were a single variable, which allows calculating proportions that sum to 100 more than a single grouping variable with `survey_mean`.

Usage

```
interact(...)
```

Arguments

`...`	variables to group by. All types of tbls accept variable names, and most will also accept functions of variables (though some database-backed tbls do not allow creating variables).

Details

Behind the scenes, this function creates a special column type that is split back into the component columns automatically by `summarize`.

Value

A vector of type `srvyr_interaction`, which is generally expected to be automatically split apart.

Examples

```
data(api, package = "survey")

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

# The sum of the whole prop column is equal to 100%
dstrata %>%
  group_by(interact(stype, awards)) %>%
  summarize(prop = survey_mean())

# But if you didn't interact, the sum of each stype's prop is 100%
dstrata %>%
  group_by(stype, awards) %>%
  summarize(prop = survey_mean())
```

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

```
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 *srvyr: A package for 'dplyr'-Like Syntax for Summary Statistics of Survey Data.*

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.

Author(s)

Maintainer: Greg Freedman Ellis <greg.freedman@gmail.com>

Authors:

- Ben Schneider [contributor]

Other contributors:

- Thomas Lumley [contributor]
- Tomasz Żółtak [contributor]
- Pavel N. Krivitsky <pavel@statnet.org> [contributor]

See Also

Useful links:

- http://gdfe.co/srvyr/
- https://github.com/gergness/srvyr/
- Report bugs at https://github.com/gergness/srvyr/issues

srvyr-se-deprecated *Deprecated SE versions of main srvyr verbs*

Description

srvyr 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", "successive-difference", "ACS",
    "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,
  strata = NULL,
  probs = NULL,
  weights = NULL,
  fpc = NULL,
  subset,
  method = c("full", "approx", "simple")
)

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

Arguments

.data	a data.frame or an object from the survey package
...	other arguments, see other functions for details
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 estimator instead of the Horvitz-Thompson estimator
repweights	Variables specifying the replication weight variables
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.
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

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

---

srvyr_interaction *srvyr interaction column*

Description

srvyr_interaction columns help calculate proportions of the interaction of 2 or more variables. They are created by interact, generally used as grouping variables in group_by and then automatically split apart by summarise.

---

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 summarizing expressions, see details
.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).
.unpack	Whether to "unpack" named data.frame columns. srvyr predates dplyr's support for data.frame columns so it does not treat them the same way by default.

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 analogous function arguments from the survey package.

The available functions from srvyr are:

survey_mean Calculate the mean of a numeric variable or the proportion falling into groups for the entire population or by groups. Based on svymean and svyciprop..

survey_total Calculate the survey total of the entire population or by groups. Based on svytotal.

survey_prop Calculate the proportion of the entire population or by groups. Based on svyciprop.

survey_ratio Calculate the ratio of 2 variables in the entire population or by groups. Based on svyratio.

survey_quantile & survey_median Calculate quantiles in the entire population or by groups. Based on svyquantile.

unweighted Calculate an unweighted estimate as you would on a regular tbl_df. Based on dplyr's summarise.

You can use expressions both in the ... of summarize and also in the arguments to the summarizing functions. Though this is valid syntactically it can also allow you to calculate incorrect results (for example if you multiply the mean by 100, the standard error is also multiplied by 100, but the variance is not).

Examples

```
data(api, package = "survey")

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

dstrata %>%
  summarise(api99_mn = survey_mean(api99),
            api00_mn = survey_mean(api00),
            api_diff = survey_mean(api00 - api99))

dstrata_grp <- dstrata %>%
  group_by(stype)

dstrata_grp %>%
  summarise(api99_mn = survey_mean(api99),
            api00_mn = survey_mean(api00),
            api_diff = survey_mean(api00 - api99))

# `dplyr::across` can be used to programmatically summarize multiple columns
# See https://dplyr.tidyverse.org/articles/colwise.html for details
```

```
# A basic example of working on 2 columns at once and then calculating the total
# the mean
total_vars <- c("enroll", "api.stu")
dstrata %>%
  summarize(across(c(all_of(total_vars)), survey_total))

# Expressions are allowed in summarize arguments & inside functions
# Here we can calculate binary variable on the fly and also multiply by 100 to
# get percentages
dstrata %>%
  summarize(api99_over_700_pct = 100 * survey_mean(api99 > 700))

# But be careful, the variance doesn't scale the same way, so this is wrong!
dstrata %>%
  summarize(api99_over_700_pct = 100 * survey_mean(api99 > 700, vartype = "var"))
# Wrong variance!
```

---

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_corr *Calculate correlation and its variation using survey methods*

Description

Calculate correlation from complex survey data. A wrapper around svyvar. survey_corr should always be called from summarise. Note this is Pearson's correlation.

Usage

```
survey_corr(
  x,
  y,
  na.rm = FALSE,
  vartype = c("se", "ci", "var", "cv"),
  level = 0.95,
  df = NULL,
  ...
)
```

Arguments

x	A variable or expression
y	A variable or expression
na.rm	A logical value to indicate whether missing values should be dropped
vartype	NULL to report no variability. Otherwise one or more of: standard error ("se", the 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
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
...	Ignored

Examples

```
data('api', package = 'survey')

apisrs %>%
  as_survey_design(.ids = 1) %>%
  summarize(api_corr = survey_corr(x = api00, y = api99))

apisrs %>%
  as_survey_design(.ids = 1) %>%
  group_by(sch.wide) %>%
  summarize(
    api_emer_corr = survey_corr(x = api00, y = emer, na.rm=TRUE, vartype="ci")
  )
```

---

survey_mean *Calculate mean/proportion and its variation using survey methods*

Description

Calculate means and proportions from complex survey data. survey_mean with proportion = FALSE (the default) or survey_prop with proportion = FALSE is a wrapper around svymean. survey_prop with proportion = TRUE (the default) or survey_mean with proportion = TRUE is a wrapper around svyciprop. survey_mean and survey_prop 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", "xlogit"),
  deff = FALSE,
  df = NULL,
  ...
)

survey_prop(
  vartype = c("se", "ci", "var", "cv"),
  level = 0.95,
  proportion = TRUE,
  prop_method = c("logit", "likelihood", "asin", "beta", "mean", "xlogit"),
  deff = FALSE,
  df = NULL,
  ...
)
```

Arguments

x	A variable or expression, or empty
na.rm	A logical value to indicate whether missing values should be dropped. See the section "Missing Values" later in this help page.
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.
...	Ignored

Details

Using survey_prop is equivalent to leaving out the x argument in survey_mean and setting proportion = TRUE and this calculates the proportion represented within the data, with the last grouping variable "unpeeled". interact allows for "unpeeling" multiple variables at once.

Missing Values

When calculating proportions for a grouping variable x, NA values will affect the estimated proportions unless they are first removed by calling filter(!is.na(x)).

When calculating means for a numeric variable, equivalent results are obtained by calling `filter(!is.na(x))` or using `survey_mean(x, na.rm = TRUE)`. However, it is better to use `survey_mean(x, na.rm = TRUE)` if you are simultaneously producing summaries for other variables that might not have missing values for the same rows as `x`.

Examples

```
data(api, package = "survey")

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

dstrata %>%
  summarise(api99_mn = survey_mean(api99),
            api_diff = survey_mean(api00 - api99, vartype = c("ci", "cv")))

dstrata %>%
  group_by(awards) %>%
  summarise(api00 = survey_mean(api00))

# Use `survey_prop` calculate the proportion in each group
dstrata %>%
  group_by(awards) %>%
  summarise(pct = survey_prop())

# Or you can also leave  out `x` in `survey_mean`, so this is equivalent
dstrata %>%
  group_by(awards) %>%
  summarise(pct = survey_mean())

# When there's more than one group, the last group is "peeled" off and proportions are
# calculated within that group, each adding up to 100%.
# So in this example, the sum of prop is 200% (100% for awards=="Yes" &
# 100% for awards=="No")
dstrata %>%
  group_by(stype, awards) %>%
  summarize(prop = survey_prop())

# The `interact` function can help you calculate the proportion over
# the interaction of two or more variables
# So in this example, the sum of prop is 100%
dstrata %>%
  group_by(interact(stype, awards)) %>%
  summarize(prop = survey_prop())

# 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_defualt = 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_old_quantile` *Calculate the quantile and its variation using survey methods*

Description

Calculate quantiles from complex survey data. A wrapper around `oldsvyquantile`, which is a version of the function from before version 4.1 of the survey package, available for backwards compatibility. `survey_old_quantile` and `survey_old_median` should always be called from `summarise`. See Thomas Lumley's blogpost <https://notstatschat.rbind.io/2021/07/20/what-s-new-in-the-survey-package/> for more details.

Usage

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

survey_old_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,
  ...
)
```

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 oldsvyquantile
ties	See oldsvyquantile
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".
...	Ignored

Examples

```
library(survey)
data(api)

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

dstrata %>%
  summarise(api99 = survey_old_quantile(api99, c(0.25, 0.5, 0.75)),
            api00 = survey_old_median(api00, vartype = c("ci")))

dstrata %>%
  group_by(awards) %>%
  summarise(api00 = survey_old_median(api00))
```

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

```
survey_quantile(
  x,
  quantiles,
  na.rm = FALSE,
  vartype = c("se", "ci", "var", "cv"),
  level = 0.95,
  interval_type = c("mean", "beta", "xlogit", "asin", "score", "quantile"),
 qrule = c("math", "school", "shahvaish", "hf1", "hf2", "hf3", "hf4", "hf5", "hf6",
    "hf7", "hf8", "hf9"),
  df = NULL,
  ...
)

survey_median(
  x,
  na.rm = FALSE,
  vartype = c("se", "ci", "var", "cv"),
  level = 0.95,
  interval_type = c("mean", "beta", "xlogit", "asin", "score", "quantile"),
 qrule = c("math", "school", "shahvaish", "hf1", "hf2", "hf3", "hf4", "hf5", "hf6",
    "hf7", "hf8", "hf9"),
  df = NULL,
  ...
)
```

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. Otherwise one or more of: standard error ("se", the default), confidence interval ("ci"), variance ("var") or coefficient of variation ("cv").
level	A single number indicating the confidence level (only one level allowed). Note that this may effect estimated standard errors (see svyquantile details on alpha, which equals 1-level).

interval_type	See svyquantile. Note that interval_type = "quantile" is only available for replicate designs, and interval_type = "score" is unavailable for replicate designs.
qrule	See svyquantile
df	A number indicating the degrees of freedom for t-distribution. The default, NULL, uses the design degrees of freedom (matches the survey package).
...	Ignored

Details

Note that the behavior of these functions has changed in srvyr version 1.1, but the old functions are still (currently) supported as survey_old_quantile and survey_old_median if you need to replicate the old results. For more details about what has changed, see Thomas Lumley's blog post on the changes, available here: <https://notstatschat.rbind.io/2021/07/20/what-s-new-in-the-survey-package/>

Examples

```
library(survey)
data(api)

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

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

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

```
survey_ratio(
  numerator,
  denominator,
  na.rm = FALSE,
  vartype = c("se", "ci", "var", "cv"),
```

```
  level = 0.95,
  deff = FALSE,
  df = NULL,
  ...
)
```

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 degf, but Inf is the usual survey package's default (except in svyciprop.
...	Ignored

Examples

```
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

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

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,
  ...
)
```

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 degf, but Inf is the usual survey package's default.
...	Ignored

Examples

```
library(survey)
data(api)

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

dstrata %>%
  summarise(enroll_tot = survey_total(enroll),
            tot_meals = survey_total(enroll * meals / 100, vartype = c("ci", "cv")))

dstrata %>%
  group_by(awards) %>%
  summarise(api00 = survey_total(enroll))

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

# level takes a vector for multiple levels of confidence intervals
dstrata %>%
  summarise(enroll = survey_total(enroll, 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_var *Calculate the population variance and its variation using survey methods*

Description

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

Usage

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

survey_sd(x, na.rm = FALSE, ...)
```

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*).
...	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

```
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))

# 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

uninteract *Break interaction vectors back into component columns*

Description

This function will not generally be needed by users because `summarise` automatically un-interacts interaction columns for you.

Usage

```
uninteract(x)

## S3 method for class 'srvyr_interaction'
uninteract(x)

## S3 method for class 'data.frame'
uninteract(x)

is.interaction(x)
```

Arguments

`x`	Either a `srvyr_interaction` column or a `data.frame`

Value

A `data.frame`

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

Description

Calculate unweighted summaries from a survey dataset, just as on a normal data.frame with `summarise`. Though it is possible to use regular functions directly, because the survey package doesn't always remove rows when filtering (instead setting the weight to 0), this can sometimes give bad results. See examples for more details.

Usage

```
unweighted(...)
```

Arguments

`...`	variables or expressions, calculated on the unweighted data.frame behind the `tbl_svy` object.

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)))


# Some survey designs, like ones with raked weights, are not removed
# when filtered to preserve the structure. So if you don't use `unweighted()`
# your results can be wrong.
# Declare basic clustered design ----
cluster_design <- as_survey_design(
  .data = apiclus1,
  id = dnum,
  weights = pw,
  fpc = fpc
)

# Add raking weights for school type ----
pop.types <- data.frame(stype=c("E","H","M"), Freq=c(4421,755,1018))
pop.schwide <- data.frame(sch.wide=c("No","Yes"), Freq=c(1072,5122))

raked_design <- rake(
  cluster_design,
  sample.margins = list(~stype,~sch.wide),
  population.margins = list(pop.types, pop.schwide)
)

raked_design %>%
filter(cname != "Alameda") %>%
  group_by(cname) %>%
  summarize(
    direct_unw_mean = mean(api99),
    wrapped_unw_mean = unweighted(mean(api99))
  ) %>%
  filter(cname == "Alameda")
```

```
# Notice how the results are different when using `unweighted()`
```

Index