Package 'srvyr'

January 20, 2019

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 0.3.4

Date 2019-01-19

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

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

Depends R (>= 3.1.2)

Imports dplyr (>= 0.7), magrittr, rlang, survey, tibble

License GPL-2 | GPL-3

LazyData TRUE

Suggests convey, DBI, dbplyr, ggplot2, knitr, Matrix, rmarkdown, pander, RSQLite, MonetDBLite, survival, testthat, vardpoor

VignetteBuilder knitr

RoxygenNote 6.1.1

NeedsCompilation no

Author Greg Freedman Ellis [aut, cre], Thomas Lumley [ctb]

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

Repository CRAN

Date/Publication 2019-01-20 17:00:03 UTC

R topics documented:

`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 '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", "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, 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 varibles
`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 fo rweights 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 varainces 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

```
# 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 identifers
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 correect for designs with simple random sampling at phase one and stratifed 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.

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

```
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 *Force computation of a database query*

Description

`collect` retreives 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 *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)

dplyr_single *Single table verbs from dplyr*

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.

`filter` keeps certain observaions. 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 gorup 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))
```

---

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 *svrvyr: 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.

`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 compatbility 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, 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 varibles
type	Type of replication weights
combined_weights	TRUE if the repweights already include the sampling weights. This is usually the case.
rho	Shrinkage factor fo rweights in Fay's method
bootstrap_average	For type = "bootstrap", if the bootstrap weights have been averaged, gives the number of iterations averaged over.
scale	Scaling constant for variance, see svrepdesign for more information.
rscales	Scaling constant for variance, see svrepdesign for more information.
fpctype	Finite population correction information
mse	if TRUE, compute varainces 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 identifers
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 correect for designs with simple random sampling at phase one and stratifed 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.

Usage

```
summarise(.data, ...)
summarize(.data, ...)
```

Arguments

.data,	tbl A tbl_svy object
...	Name-value pairs of summary functions

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.

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

```
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 depracated 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")))

dstrata %>%
  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_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_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 = NULL,
  level = 0.95, q_method = "linear", f = 1,
  interval_type = c("Wald", "score", "betaWald", "probability",
  "quantile"), ties = c("discrete", "rounded"), df = Inf,
  .svy = current_svy(), ...)

survey_median(x, 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 = Inf, .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)

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,
  .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 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(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_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 = NULL, 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 `degf`, 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),
            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
```

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

Description

Chisquared tests of association for survey data.

Arguments

`formula`	Model formula specifying margins for the table (using + only)
`design`	survey object
`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 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.

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

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)))
```

Index