survey data analysis with srvyr : : CHEAT SHEET (1 of 2)

Describing the Survey Design

To properly analyze survey data, create a **survey design object.** This object contains the data, weights, and other metadata used for analysis.

```
design <- as_survey_design(
  .data      = my_data_frame,
  ids        = _,   (Required) - Variable(s) identifying sampling units (sometimes called "clusters")
  strata     = _,   (Optional) - Variable(s) identifying sampling strata
  fpc        = _,   (Optional) - Variable(s) in the data giving population sizes or sampling fractions for each stratum.
  probs      = _,   (Optional) - Variable(s) in the data giving sampling probabilities for each sampling unit
  weights    = _,   (Optional) - A variable in the data listing the sampling weight for each observation in the data
)
```

FOR MULTISTAGE SAMPLES: List one variable for each stage of sampling: e.g., `c(SCHOOL, STUDENT)`. Only applies to **ids**, **strata**, **fpc**, and **probs**.

EXAMPLE DATA: Stratified Cluster Sample

Survey of students: data includes *every* student in four sampled schools, where two schools were selected in each of two school districts

Obs	DISTRICT	SCHOOL	SCHOOL_PROB	DISTRICT_SIZE	WEIGHT
1	District 1	School 1	0.40	5	2.5
2	District 1	School 1	0.40	5	2.5
3	District 1	School 2	0.40	5	2.5
4	District 1	School 2	0.40	5	2.5
5	District 2	School 3	0.20	10	5.0
6	District 2	School 3	0.20	10	5.0
7	District 2	School 4	0.20	10	5.0
8	District 2	School 4	0.20	10	5.0

```
design <- as_survey_design(
  .data      = school_survey_data,
  ids        = SCHOOL,
  strata     = DISTRICT,
  fpc        = DISTRICT_SIZE,
  weights    = WEIGHT
)
```

EXAMPLE DATA: Multistage Sample

Survey of students: data includes a *sample* of students in four sampled schools, where two schools were selected in each of two school districts

Obs	DISTRICT	SCHOOL	STUDENT	DISTRICT_SIZE	SCHOOL_SIZE	WEIGHT
1	District 1	School 1	Student 1	4	100	100
2	District 1	School 1	Student 2	4	100	100
3	District 1	School 2	Student 3	4	150	150
4	District 1	School 2	Student 4	4	150	150
5	District 2	School 3	Student 5	10	200	500
6	District 2	School 3	Student 6	10	200	500
7	District 2	School 4	Student 7	10	250	625
8	District 2	School 4	Student 8	10	250	625

```
design <- as_survey_design(
  .data      = school_survey_data,
  ids        = c(SCHOOL, STUDENT),
  strata     = c(DISTRICT, SCHOOL),
  fpc        = c(DISTRICT_SIZE, SCHOOL_SIZE),
  weights    = WEIGHT
)
```

TIPS FOR CREATING A DESIGN OBJECT

- If each observation in the data is its own sampling unit, then use `ids = NULL`
- If analyzing only a subset of the data, create the design object first with ALL the data, *then* subset the design object using `filter()`
- If the weights argument isn't used, then weights will automatically be created based on the probs argument (if available) or the fpc argument (if probs is unavailable)

DEALING WITH "LONELY PSUS"

- A "lonely PSU" is a sampling unit that is the only sampling unit in its stratum (sometimes referred to as a "singleton stratum").
- This can be a problem for variance estimation. You can use the "survey.lonely.psu" option to address this problem.
- When analyzing subsets of data (i.e., "domains"), this option can be applied to subsets with only one PSU by setting the following option to TRUE: `options("survey.adjust.domain.lonely")`

`options("survey.lonely.psu" = "fail")`

Throw an error message if there are any lonely PSUs

`options("survey.lonely.psu" = "adjust")`

This option assumes that the lonely PSU comes from a stratum whose average is the same as the average PSU.

`options("survey.lonely.psu" = "average")`

This option assumes that the lonely PSU contributes to the variance the same as the average stratum.

`options("survey.lonely.psu" = "remove")`

Ignore the lonely PSU when estimating variances.

Database-backed Surveys

For large datasets, you can keep your data in a database table, only loading data into R as-needed. To read more about database-backed surveys, see the srvyr vignette **"Databases in srvyr"**.

1. Create a database connection using the DBI package.
2. Use the dplyr function tbl(), to refer to a specific table in the database
3. Use the table for the .data argument of as_survey_design() or as_survey_rep()

```
library(DBI)
library(dplyr)

db_conn <- dbConnect([dbdriver],…)

tbl(db_conn, "TABLE_NAME") |>
  as_survey_design(…)
```

Replicate Weights

An alternative to creating a survey design object is to create a **replicate design object**, using **replicate weights** provided on the input dataset.

```
design <- as_survey_rep(
  .data      = my_data_frame,
  weights    = _,
  repweights = _,
  type       = _,
  mse        = _
)
```

ARGUMENTS

- **weights:** The variable name of the full-sample weights
- **repweights**: The variable names of the replicate weights. Some useful helper functions for listing them are:
  - **num_range**("REP_WGT", 1:80)
  - **starts_with**("REP_WGT")
- **type:** The replication method used to create the replicate weights (e.g., "bootstrap", "JK1", or "BRR"). For more flexibility, you can specify type = "other" and use the arguments "scale" and "rscales" (see the "Scale Factors" section below).
- **mse:** Use TRUE to compute variances based on sum of squares around the full-sample estimate; use FALSE to use squares around the mean of the replicate estimates
- **degf**: (Optionally) Specify the degrees of freedom

EXAMPLE CODE

```
design <- as_survey_rep(
  .data      = my_data_frame,
  weights    = FULL_SAMPLE_WGT,
  repweights = num_range("REP_WGT", 1:50),
  type       = "bootstrap",
  mse        = TRUE
)
```

SCALE FACTORS

All replication types use the same formula for variances, involving scale factors named **scale** and **rscales**. Those factors are determined automatically unless **type = "other" or "JKn"**, in which case **the following arguments** can be used**:**

- **scale:** Overall scale factor
- **rscales:** Replicate-specific scale factors

$$\mathrm{V}(\hat{\theta}) = A \sum_{r=1}^{R} a_r \left(\hat{\theta}_r - \hat{\theta}\right)^2$$

scale: A — rscales: a_r

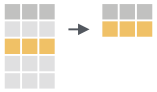
if **mse=FALSE**, this is replaced by the average: $\frac{1}{R}\sum_{r=1}^{R}\hat{\theta}_r$

 • Learn more at http://gdfe.co/srvyr/ • package version 1.3.0 • Updated: 2025-01

survey data analysis with srvyr : : **CHEAT SHEET (2 of 2)**

After creating a **survey design object** with srvyr, you can manipulate data and compute summaries using dplyr verbs such as **filter()** or **summarize()**.
srvyr provides specialized statistical summary functions to calculate weighted estimates along with standard errors and confidence intervals.

Manipulating Data



filter(.data, …, .preserve = FALSE**)**
Extract rows that meet logical criteria.

```
design |> filter(AGE >= 16, AGE <= 65)
```

mutate(.data, …**)**
Compute new column(s).

```
design |> mutate(BMI = WGT/HEIGHT)
```

Summarizing Functions

Apply summary functions to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).

summarize(.data, …**)**
Compute table of summaries.

```
design |> summarise(mean_age = survey_mean(AGE))
```

survey_count(.data, …, sort = FALSE, name = NULL**)**
Weighted counts and standard errors for each group defined by the variables in …

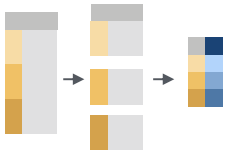
```
survey_count(mtcars, cyl)
```

Grouping Cases

Use **group_by(.data, …)** to create a "grouped" copy of a table grouped by columns in ...

dplyr and srvyr functions will manipulate each "group" separately and combine the results.

TIP: Grouped operations work with `summarize()`, `mutate()`, and `filter()`



```
design |>
  group_by( REGION ) |>
  summarize(
    mean_age = survey_mean( AGE )
  )
```

Column-wise Operations

Use functions from **dplyr** such as `across()` to apply summaries to multiple columns.

Example: design |> summarise(across(is.numeric, survey_mean))

Read more about column-wise functions from **dplyr** here:
https://dplyr.tidyverse.org/articles/colwise.html

Statistical Summary Functions

These functions are used ***within*** the dplyr functions **summarize()** and **mutate()**, returning an estimate and its standard error (with the suffix "_se")

Example Code

```
design |>
  group_by(region) |>
  summarize(pop_size    = survey_total(),
            mean_age = survey_mean(AGE))
```

Example Output

region	pop_size	pop_size_se	mean_age	mean_age_se
North	15,342,875	25,410	46	1.2
South	5,861,942	37,902	44	1.1

survey_total(.data, x**)**

Estimate the population total, either for a numeric variable **x** or—if **x** is unspecified—for the current group.

survey_mean(.data, x, …, .preserve = FALSE**)**

Estimate the population mean.

survey_mean(.data, x, proportion = TRUE, prop_method = "logit"**)**

Estimate a proportion if **x** is a binary variable with values 0 and 1 or a logical variable.

x can be an expression such as `AGE > 25`

Use the **prop_method** argument to choose a specialized method for computing confidence intervals.

survey_median(.data, x**)**

Estimate the population median.

survey_quantile(.data, x, quantiles = 0.5**)**

Estimate population quantiles such as the median or quartiles.

survey_sd(.data, x**)**

Estimate the population standard deviation of a variable.

survey_var(.data, x**)**

Estimate the population variance of a variable.

survey_ratio(.data, numerator, denominator**)**

Estimate the ratio of population means for two variables.

survey_corr(.data, x, y**)**

Estimate the population correlation between two variables.

STANDARD ERRORS AND CONFIDENCE INTERVALS

Every statistical summary function includes a **vartype** argument which can be used to request one or more of the following measures of sampling variation.

- **"var"**: The sampling variance
- **"se"**: The standard error (included by default)
- **"cv"**: The coefficient of variation
- **"ci"**: A confidence interval, with confidence level controlled by the **level** argument

Example Code

```
design |>
  summarize(
    age = survey_mean(AGE, vartype = c("se", "ci"), level = 0.95)
  )
```

Proportions and Percentages

For a grouped dataset, you can estimate each group's proportion of the total population by using **survey_prop()**.

survey_prop(.data, …, prop_method = "logit"**)**

By default, if there are *multiple* grouping variables, **x** and **y**, then the result will be proportions of **y** within categories of **x**.

The **interact()** function can be used to obtain proportions for combinations of variables.

Nested Proportions: Proportion of one variable *within* another

```
design |>
  group_by(region, agree) |>
  summarize(prop = survey_prop())
```

region	agree	prop	prop_se
North	Yes	0.9	0.01
North	No	0.1	0.01
South	Yes	0.5	0.05
South	No	0.5	0.05

Cross-classified Proportions: Proportions of *combinations* of variables

```
design |>
  group_by(interact(region, agree)) |>
  summarize(prop = survey_prop())
```

region	agree	prop	prop_se
North	Yes	0.675	0.01
North	No	0.075	0.01
South	Yes	0.125	0.05
South	No	0.125	0.05

 • Learn more at **http://gdfe.co/srvyr/** • package version 1.3.0 • Updated: 2025-01