CASSION · COURSE

Survey Analysis, Sampling and Weighting

Analyse a two-stage cluster survey properly — weights, strata, design effect and confidence intervals — using the conventions DHS, MICS and SMART already impose.

Instructor	Pierre Robentz Cassion
Level	Intermediate
Learner hours	22 h
Taught in	Python · R
Updated	July 28, 2026
Sectors	Public Health · Nutrition · Food Security
Standards and methodologies	SMART survey · Demographic and Health Survey (DHS) · Multiple Indicator Cluster Survey (MICS) · Sustainable Development Goals (SDG)

Read and run the course online at data-analysis.cassion.dev/courses/survey-analysis-sampling

What you will be able to do

- Reconstruct sampling weights from a frame, including the non-response adjustment, and prove they sum to the population
- Say why an unweighted mean from a cluster survey is the wrong number, and how far wrong it is on a given survey
- Compute a sample size for a prevalence estimate, and state what precision the budget actually bought
- Estimate a proportion with a design-adjusted confidence interval in both R and Python, and read the design effect off the result
- Handle non-response and replacement clusters explicitly rather than by ignoring them
- Refuse to disaggregate past the point the sample supports, and say in a report where that point is

Prerequisites

None. This course assumes no prior programming experience.

Contents

About this course

A survey estimate is not a number. It is a number, an interval, and a design that produced both — and dropping either of the last two is the most common way survey findings are misreported in this sector.

This is the heaviest course in the programme so far, and the first that genuinely needs statistics. It earns that weight because the alternative is expensive: a household survey costs vehicles, enumerators and six weeks, and analysing it as though it were a spreadsheet of independent observations throws away most of what was bought.

Almost everything runs against one dataset built for this course — 996 household interviews drawn from a 470-area frame by stratified sampling with probability proportional to size. **It ships the frame rather than the weights**, in the same way the SMART survey ships raw measurements rather than z-scores, because reconstructing the weights is the exercise.

Three findings from that survey shape the course. Food insecurity is 32.8% unweighted and 29.1% weighted, because the smallest stratum is the most oversampled and the worst off. The design effect for that estimate is 1.60, so the survey's 996 interviews are worth about 623 independent ones. And the design effect for improved water access is **0.91** — below one, because the stratification gain outweighs the clustering loss — which is the result that stops "design effect" being memorised as "a penalty".

Both Python and R, throughout. R's `survey` package is the reference implementation this sector uses and the course teaches it; the Python examples compute the same quantities directly, so the arithmetic is visible rather than delegated.

You should have done *Indicator Design and the LogFrame*, or at least be in the habit of writing a denominator down before computing anything. This course adds the interval to the number that course taught you to define.

Part I

The sample is not the population

CHAPTER 1

The mean of your sample is not the mean of anything

Lesson 1 of 8 · 120 min

32.8% against 29.1% on the same 996 interviews, and 62.8% against 69.8% on the same households. The gap is the design, and it is not noise.

1.1 Two numbers from one file

Take the household survey, count the households classified food insecure, divide by the number of households. That is 32.8%.

Now weight each household by the number of households it represents in the population, and the same file gives 29.1%.

```python
import pandas as pd

survey = pd.read_csv("household-survey-2025.v1.csv")
frame = pd.read_csv("household-survey-frame-2025.v1.csv")

unweighted = (survey["food_insecure"] == "true").mean()
print(f"unweighted: {unweighted:.1%}")
```

```r
library(dplyr)
library(readr)

survey <- read_csv("household-survey-2025.v1.csv")
frame  <- read_csv("household-survey-frame-2025.v1.csv")

mean(survey$food_insecure == "true")
```

Both numbers are computed correctly. One of them is an estimate of the population and the other is a description of who happened to be interviewed, and **only one of those is what the report claims to be reporting.**

1.2 Where the gap comes from

Nothing about the arithmetic. Everything about how the sample was drawn.

Stratum	Households in the frame	Areas sampled	Households interviewed	Food insecure
Urban	21,270	25	316	14.6%
Rural accessible	28,958	25	337	36.2%
Rural remote	6,200	25	343	46.4%

Read the first and last columns together. **Rural remote is 11% of the population and 34% of the sample**, and it has the worst food insecurity by a wide margin.

An unweighted average of the three columns therefore over-represents the worst-off stratum by a factor of three. The 32.8% is not the district's food insecurity; it is the food insecurity of a population in which a third of households are remote rural, and no such population exists.

```python
by_stratum = (
    survey.assign(insecure=survey["food_insecure"] == "true")
    .groupby("stratum")
    .agg(interviews=("insecure", "size"), rate=("insecure", "mean"))
)
frame_totals = frame.groupby("stratum")["households"].sum()

comparison = by_stratum.join(frame_totals.rename("frame_households"))
comparison["sample_share"] = comparison["interviews"] / comparison["interviews"].sum()
comparison["population_share"] = (
    comparison["frame_households"] / comparison["frame_households"].sum()
)
print(comparison.round(3))
```

```r
survey |>
  summarise(interviews = n(), rate = mean(food_insecure == "true"), .by = stratum) |>
  left_join(summarise(frame, frame_households = sum(households), .by = stratum),
            by = "stratum") |>
  mutate(sample_share     = interviews / sum(interviews),
         population_share = frame_households / sum(frame_households))
```

Stratum	Sample share	Population share
Urban	31.7%	37.7%
Rural accessible	33.8%	51.3%
Rural remote	34.4%	11.0%

When those two columns differ, an unweighted estimate is biased, and the size and direction of the bias are entirely predictable from the table.

1.3 Why anyone would design it that way

The obvious reaction is that the sample was drawn badly. It was not, and knowing why is most of what this lesson is for.

Equal allocation across unequal strata is a deliberate, standard choice, and it buys something specific: **a usable estimate for each stratum separately.** Rural remote holds 11% of households, so a sample proportional to population would have put about 110 interviews there — enough for a national figure and nowhere near enough to say anything about the stratum on its own.

The trade is explicit:

- **Proportional allocation** gives the best national estimate and weak sub-national ones.
- **Equal allocation** gives comparable sub-national estimates and requires weighting for anything national.

DHS, MICS and most humanitarian assessments choose the second, because the whole point of the survey is usually to compare places. **The weights are not a correction for a mistake. They are the price of the design.**

1.4 The stratum estimates need no weights

A useful thing falls out of the design and it is worth seeing early.

```
print(by_stratum["rate"].round(3))
```

```
survey |> summarise(rate = mean(food_insecure == "true"), .by = stratum)
```

Within a stratum, this design is **self-weighting** — every household had the same probability of selection — so the unweighted stratum rate is already the estimate. Weights matter only when you combine strata.

That is why a report can legitimately show unweighted stratum figures beside a weighted total, and why doing so without saying which is which confuses everyone.

1.5 The direction is not always the same

Food insecurity falls when you weight. Improved water access rises, and by more.

```
for outcome in ["food_insecure", "improved_water_source"]:
    print(f"{outcome:24} unweighted {(survey[outcome] == 'true').mean():.1%}")
```

```
survey |>
  summarise(across(c(food_insecure, improved_water_source), ~ mean(.x == "true")))
```

Outcome	Unweighted	Weighted
Food insecure	32.8%	29.1%
Improved water source	62.8%	69.8%

Seven points on the second one. There is no general rule that unweighted estimates are too high or too low: the bias goes in whichever direction the oversampled stratum differs, and it differs by a different amount for every outcome.

So you cannot correct an unweighted figure by rule of thumb, and a report that presents one with a note saying "unweighted, so the true figure is somewhat lower" is guessing.

1.6 What to do when there is no frame

Sometimes you inherit a dataset with no frame and no weights. Three honest positions, and none of them is to report the unweighted mean as an estimate.

- **Report by stratum only**, if the strata are recorded. Sub-national figures are usually what the audience wanted anyway.
- **Reconstruct approximate weights** from an external population source, and document that they are approximate.
- **Say the survey cannot support a population estimate.** This is a real answer and it is better than a number that will be quoted for three years.

1.7 What comes next

The weights are computable, from the frame this dataset ships. The next lesson builds them from first principles — selection probability at each stage, base weight, non-response adjustment — and then proves them, by checking that they sum to the number of households the frame says exist.

CHAPTER 2

Building the weights from the frame

Lesson 2 of 8 · 120 min

Selection probability at each stage, the base weight it implies, the non-response adjustment, and the check that proves the whole thing — weights that sum to 56,428, which is exactly what the frame says exists.

2.1 A weight is one number with one meaning

A sampling weight is how many population units this sampled unit stands for.

That sentence is the whole of this lesson. Everything else is arithmetic to get there, and every check is a way of asking whether the weights still mean it.

The weight is the reciprocal of the probability of selection. A household with a 1-in-60 chance of being picked represents 60 households; a household with a 1-in-18 chance represents 18. Nothing more mysterious than that.

2.2 Stage one: probability proportional to size

Twenty-five enumeration areas were drawn from each stratum, with a probability proportional to the households each area holds.

```
P(area i selected) = n_h * M_i / M_h
```

where `n_h` is areas sampled in stratum h (25), `M_i` is households in area i, and `M_h` is households in the whole stratum.

```python
import pandas as pd

frame = pd.read_csv("household-survey-frame-2025.v1.csv")

stratum_households = frame.groupby("stratum")["households"].sum()
selected = frame[frame["selected"] == True].copy()

selected["p_stage1"] = (
    25 * selected["households"] / selected["stratum"].map(stratum_households)
)
print(selected[["ea_id", "stratum", "households", "p_stage1"]].head())
```

```r
library(dplyr)

stratum_households <- frame |> summarise(M_h = sum(households), .by = stratum)

selected <- frame |>
  filter(selected) |>
  left_join(stratum_households, by = "stratum") |>
  mutate(p_stage1 = 25 * households / M_h)
```

A larger area is more likely to be selected. That is the point of PPS: it puts the interviews where the people are, which stops a survey spending a third of its budget on hamlets.

2.3 Stage two: a fixed take

Fourteen households were drawn from each selected area, from that area's listing.

```
P(household selected | area selected) = m / M_i
```

with $m = 14$.

```
selected["p_stage2"] = 14 / selected["households"]
selected["p_overall"] = selected["p_stage1"] * selected["p_stage2"]
print(selected.groupby("stratum")["p_overall"].describe()[["min", "max"]])
```

```
selected <- selected |>
  mutate(p_stage2 = 14 / households,
         p_overall = p_stage1 * p_stage2)

selected |> summarise(min = min(p_overall), max = max(p_overall), .by = stratum)
```

2.4 The two probabilities cancel

Multiply them out and `M_i` disappears:

```
P(household) = (n_h * M_i / M_h) * (m / M_i) = n_h * m / M_h
```

Every household in a stratum has the same overall probability, regardless of how big its area is. **The design is self-weighting within a stratum.**

Run the code and the minimum and maximum `p_overall` are identical within each stratum, which is the arithmetic confirming it. This is not a coincidence or a convenience — it is precisely why PPS is paired with a fixed take, in DHS, in MICS and in SMART. The size measure that decides which areas are visited is cancelled by the size measure that decides how many households are taken.

Two consequences worth holding. A weight that varies within a stratum in a PPS design means something is wrong — usually a frame whose size measure is out of date. And the base weight is a property of the stratum, not of the household, so it can be computed in three numbers.

```
base_weight = stratum_households / (25 * 14)
print(base_weight.round(1))
```

```
stratum_households |> mutate(base_weight = M_h / (25 * 14))
```

Stratum	Frame households	Base weight
Urban	21,270	60.8
Rural accessible	28,958	82.7
Rural remote	6,200	17.7

One rural remote household stands for eighteen; one rural accessible household stands for eighty-three. That ratio is the entire difference between 32.8% and 29.1%.

2.5 The non-response adjustment

Fourteen households were selected per area. Fewer were interviewed — 996 of the 1,050 selected. The households that were interviewed have to carry the ones that were not.

```python
survey = pd.read_csv("household-survey-2025.v1.csv")

interviewed = selected.set_index("ea_id")["households_interviewed"]

survey["base_weight"] = survey["stratum"].map(base_weight)
survey["nr_adjustment"] = 14 / survey["ea_id"].map(interviewed)
survey["weight"] = survey["base_weight"] * survey["nr_adjustment"]

print(survey["weight"].describe()[["min", "max"]].round(1))
```

```r
survey <- survey |>
  left_join(select(selected, ea_id, households_interviewed), by = "ea_id") |>
  left_join(mutate(stratum_households, base_weight = M_h / (25 * 14)), by = "stratum") |>
  mutate(weight = base_weight * 14 / households_interviewed)

range(survey$weight)
```

Weights now run from 17.7 to 96.5.

Adjust within the area, not within the stratum. The households that did not answer in a given area resemble the households that did answer in that area far more than they resemble the stratum average. Adjusting at stratum level is easier and throws away exactly the information that makes the adjustment worth making.

The assumption is stated and it is not free: **non-respondents are assumed to resemble respondents in the same area.** Where you have reason to think otherwise — locked compounds in one neighbourhood, a security incident on one day — say so in the limitations, because no weight can fix it.

2.6 The check that proves the whole construction

```python
total = survey["weight"].sum()
frame_total = frame["households"].sum()
print(f"weights sum to {total:,.0f}; frame holds {frame_total:,}")
assert abs(total - frame_total) < 1
```

```r
c(weights = sum(survey$weight), frame = sum(frame$households))
```

56,428 and 56,428. The weights sum to exactly the number of households the frame says exist, which is what "how many units this one stands for" has to mean if it means anything.

Run this check every time. It catches a stratum whose base weight used the wrong denominator, a non-response adjustment applied twice, and a join that dropped areas — three mistakes that are otherwise invisible because the resulting estimate still looks like a percentage.

2.7 Normalised weights, and when to use them

Some software wants weights averaging one rather than summing to the population.

```python
survey["weight_norm"] = survey["weight"] / survey["weight"].mean()
```

```r
survey <- survey |> mutate(weight_norm = weight / mean(weight))
```

Proportions and means are identical either way — the scaling cancels. **Totals are not.** A normalised weight cannot estimate "how many households are food insecure", only "what share are". Keep the population-scaled weight as the primary and derive the normalised one where a tool insists.

2.8 Person-level and sub-sample weights

Two more weights fall out of the same logic, and both are routinely got wrong.

A person-level weight is the household weight times the household size, because a household of eight represents eight times as many people as it does households.

```python
people = survey[survey["household_size"].notna()].copy()
people["person_weight"] = people["weight"] * people["household_size"]
print(f"estimated population {people['person_weight'].sum():,.0f}")
```

```r
people <- survey |>
  filter(!is.na(household_size)) |>
  mutate(person_weight = weight * household_size)

sum(people$person_weight)
```

That gives 333,336 against 328,127 in the frame listing — 1.6% apart. Twenty households have no recorded size and drop out, and a frame listing is itself an estimate made months earlier. **A gap of that size is normal and worth stating; a gap of 20% means something is wrong.**

A sub-sample weight applies when one unit is chosen from several. One child under five was measured per household, so a measured child in a household with three eligible children represents three children.

```python
children = survey[survey["child_muac_mm"].notna()].copy()
children["child_weight"] = children["weight"] * children["children_under5"]
```

```r
children <- survey |>
  filter(!is.na(child_muac_mm)) |>
  mutate(child_weight = weight * children_under5)
```

Skip that multiplication and children in large households are under-represented — and large households are systematically poorer, so the bias runs in a predictable direction. On this survey, MUAC-based acute malnutrition is 13.3% among the measured children and **11.4%** once the child weights are applied.

2.9 Save the weights beside the data

```python
survey[["household_id", "ea_id", "stratum", "base_weight",
        "nr_adjustment", "weight"]].to_csv("outputs/weights.csv", index=False)
```

```
survey |>
  select(household_id, ea_id, stratum, base_weight, households_interviewed, weight) |>
  readr::write_csv(here::here("outputs", "weights.csv"))
```

Keep the **components**, not just the final weight. When somebody asks in a year why one household counts for 96 and another for 18, the answer is in the columns rather than in your memory of a script.

2.10 What comes next

You can now produce a population estimate. The next unit asks what it cost: how many households a given precision needs, and why the textbook sample size formula gives an answer that is about half of what a cluster survey actually requires.

Part II

What the design buys

CHAPTER 3

What precision did the budget buy?

Lesson 3 of 8 · 120 min

The sample size formula, the four numbers it needs, and why the answer for a cluster survey is roughly double the textbook one. Plus reading it backwards, which is what you usually have to do.

3.1 The question is always asked too late

Sample size is decided in a proposal, months before anyone analyses anything, and usually by whoever is writing the budget. The analyst inherits it.

So this lesson runs in both directions. Forwards, to size a survey you are designing. **Backwards, to work out what precision the survey you already have can support** — which is the version you will need more often, and the one that decides what you are allowed to publish.

3.2 The formula, and the four numbers in it

For a proportion:

```
n = z^2 * p * (1 - p) / d^2 * deff
```

Four inputs, and each one is a decision somebody has to make.

- `z` — the confidence level. 1.96 for 95%. Almost never changed.
- `p` — the expected proportion. You do not know it, which is the point of the survey. Use the last round, a comparable area, or 0.5 if you genuinely have nothing, because 0.5 maximises the required sample and is therefore the safe guess.
- `d` — the precision you need, as an absolute margin. This is the number that is actually negotiated, and the one nobody writes down.
- `deff` — the design effect. The next lesson is entirely about it; for sizing, use the previous round's value or 2.0 as a placeholder.

```python
import math

def sample_size(p, d, deff=1.0, z=1.96, response=1.0):
    n = z**2 * p * (1 - p) / d**2 * deff
    return math.ceil(n / response)

print(sample_size(0.30, 0.05))            # simple random
print(sample_size(0.30, 0.05, deff=2.4))  # cluster survey
print(sample_size(0.30, 0.03, deff=2.4))  # tighter precision
```

```r
sample_size <- function(p, d, deff = 1, z = 1.96, response = 1) {
  ceiling(z^2 * p * (1 - p) / d^2 * deff / response)
```

```
}

c(sample_size(0.30, 0.05),
  sample_size(0.30, 0.05, deff = 2.4),
  sample_size(0.30, 0.03, deff = 2.4))
```

Expected p	Precision	deff	Households needed
30%	±5 points	1.0	323
30%	±5 points	2.4	775
30%	±3 points	2.4	2,152

Three things to take from that table.

Clustering more than doubles the requirement. 323 becomes 775 for the same precision. Any sample size that does not mention a design effect is sizing a simple random sample, and nobody runs one of those in this sector.

Precision is brutally expensive. Going from ±5 to ±3 points nearly triples the sample. Precision costs with the *square* of the improvement, which is the single most useful fact in this lesson when somebody asks for a tighter figure.

Non-response is a divisor, not an afterthought. Expecting 90% response means dividing by 0.9 — and it must be applied to the households *selected*, not to the households interviewed, which is the same distinction the weighting lesson made.

3.3 Reading it backwards

You have 996 interviews and a design effect of 1.60. What precision does that buy?

```
def precision(n, p, deff=1.0, z=1.96):
    return z * math.sqrt(p * (1 - p) / n * deff)

print(f"+/- {precision(996, 0.291, deff=1.60):.1%}")
print(f"+/- {precision(343, 0.464, deff=1.60):.1%}")   # rural remote alone
```

```
precision <- function(n, p, deff = 1, z = 1.96) z * sqrt(p * (1 - p) / n * deff)

c(overall = precision(996, 0.291, deff = 1.60),
  remote  = precision(343, 0.464, deff = 1.60))
```

About ±3.6 points overall, and about ±5.3 points for rural remote on its own.

Do this before you promise anything. It tells you which comparisons the survey can settle and which it cannot, and it is the calculation behind the last lesson of this course.

3.4 The number that is really being negotiated

`d` is where the argument is, and it is usually conducted in the wrong currency — people argue about the sample size when the thing they disagree about is how precise the answer needs to be.

Reframe it. Precision is only meaningful against a decision:

- **A threshold to cross.** If the question is whether GAM exceeds 15%, and the estimate is near 14%, you need an interval narrow enough to sit on one side of the line. That is a precision requirement with a number in it.

- **A change to detect.** Detecting a five-point improvement between rounds needs roughly *four times* the sample of estimating a level to ±5 points, because two estimates each carry error.
- **A comparison between groups.** Same problem: two intervals, both of which have to be narrow.

```python
# Detecting a difference between two groups of equal size
def sample_per_group(p1, p2, deff=1.0, power_z=0.84, z=1.96):
    pbar = (p1 + p2) / 2
    n = (z + power_z) ** 2 * 2 * pbar * (1 - pbar) / (p1 - p2) ** 2
    return math.ceil(n * deff)

print(sample_per_group(0.30, 0.25, deff=2.4))
```

```r
sample_per_group <- function(p1, p2, deff = 1) {
  pbar <- (p1 + p2) / 2
  ceiling((1.96 + 0.84)^2 * 2 * pbar * (1 - pbar) / (p1 - p2)^2 * deff)
}

sample_per_group(0.30, 0.25, deff = 2.4)
```

Run it and the number is uncomfortable. **Say the number out loud early**, because a survey sized to estimate a level and then used to claim a change is the single commonest overreach in this sector's reporting.

3.5 Clusters versus households per cluster

For a fixed budget you choose between more clusters and more households in each, and the two are not equivalent.

More clusters is almost always better, because the design effect grows with the number of households *per cluster*. Thirty clusters of ten beats ten clusters of thirty for the same 300 interviews, by a wide margin.

The counter-pressure is cost: a cluster costs a vehicle and a day, and a household costs an hour. SMART's convention of about 30 clusters exists exactly at that trade-off, and it is a reasonable default when you have nothing better.

```python
for m in [8, 14, 20, 30]:
    deff = 1 + (m - 1) * 0.11
    clusters = math.ceil(775 * (deff / 2.4) / m)
    print(f"{m:>3} households x {clusters:>3} clusters -> deff {deff:.2f}")
```

```r
for (m in c(8, 14, 20, 30)) {
  deff <- 1 + (m - 1) * 0.11
  cat(sprintf("%3d households, deff %.2f\n", m, deff))
}
```

At an intra-cluster correlation of 0.11 — the value measured on this survey — fourteen households per cluster gives a clustering design effect of about 2.4, and thirty would give 4.2. **The fourteen was a design decision, not an accident.**

3.6 Write the calculation down

A sample size is an argument, and it belongs in the survey protocol with its inputs visible.

```
Indicator         Household food insecurity prevalence
Expected p        0.30 (2023 round, same instrument)
Precision         +/- 5 percentage points, 95% confidence
Design effect     2.0 (2023 round measured 1.9; rounded up)
Expected response 92%
Households        323 * 2.0 / 0.92 = 703, rounded to 25 clusters x 14 = 350 per
                  stratum, 1,050 in total across three strata
Rationale         Equal allocation, because each stratum must support its own
                  estimate. National figures require weighting.
```

Every number in that block is challengeable, which is the point. A protocol saying "1,050 households were surveyed" invites the question and cannot answer it.

3.7 What comes next

Two of the four inputs were the design effect, and it has been a placeholder so far. The next lesson measures it — where it comes from, how to compute it from your own data, and why one outcome in this survey has a design effect below one.

CHAPTER 4

The design effect, and the one that came out below one

Lesson 4 of 8 · 120 min

Clustering costs precision, stratification buys it back, and the design effect is the net. 1.60 for food insecurity and 0.91 for water access, on the same 996 interviews.

4.1 What it is

The design effect is a ratio:

```
deff = variance under this design / variance under simple random sampling
```

A design effect of 2 means your estimate is as precise as a simple random sample half the size would have been. It converts a sample of 996 into an **effective sample size** — the number of independent observations your survey is actually worth.

```
effective n = n / deff
```

That second quantity is the one to put in a report. "996 households, effective sample 623" says more than either number alone, and it is the honest answer to "how big was your survey".

4.2 Why clustering costs you

Households in the same enumeration area resemble each other. They share a water point, a market, a road, a harvest. So the second household you interview in an area tells you less than the first did, and the fourteenth tells you very little.

The measure of that resemblance is the **intra-cluster correlation**, and it maps to the clustering penalty directly:

```
deff_clustering = 1 + (m - 1) * ICC
```

with `m` the number of interviews per cluster.

```
import pandas as pd

by_area = (
    survey.assign(insecure=survey["food_insecure"] == "true")
    .groupby("ea_id")["insecure"]
    .agg(["mean", "size"])
)

p = (survey["food_insecure"] == "true").mean()
m = by_area["size"].mean()
between = by_area["mean"].var(ddof=0)
icc = (between - p * (1 - p) / m) / (p * (1 - p))

print(f"mean cluster size {m:.1f}, ICC {icc:.3f}, "
      f"clustering deff {1 + (m - 1) * icc:.2f}")
```

```r
by_area <- survey |>
  summarise(rate = mean(food_insecure == "true"), n = n(), .by = ea_id)

p <- mean(survey$food_insecure == "true")
m <- mean(by_area$n)
icc <- (var(by_area$rate) * (nrow(by_area) - 1) / nrow(by_area) -
        p * (1 - p) / m) / (p * (1 - p))

c(m = m, icc = icc, deff = 1 + (m - 1) * icc)
```

On this survey: 13.3 interviews per area, ICC 0.11, clustering design effect about **2.4**. Two things about ICC worth carrying:

- **It is a property of the outcome, not of the survey.** Water source is highly clustered — a village has a borehole or it does not. Household size is barely clustered at all. One design effect per outcome, always.
- **Published values are the best guess you have.** DHS and SMART reports publish design effects, and using last round's for the same indicator in the same place is far better than assuming 1.

4.3 Why stratification buys some back

Stratification does the opposite. By forcing the sample to cover each stratum, it removes the possibility of a sample that landed mostly in one — and that possibility is part of the variance of a simple random sample.

The gain is large exactly when strata differ a lot on the outcome. Improved water access runs 88% urban against 41% rural remote, and stratification guarantees those are represented in fixed proportions rather than by luck.

4.4 The net effect, measured

The two pull in opposite directions, so **compute the design effect from the data rather than reasoning about it.** Estimate the variance under the real design and divide by what simple random sampling would have given.

```python
import numpy as np

def design_effect(df, outcome, weight="weight"):
    p = np.average(df[outcome] == "true", weights=df[weight])
    total_w = df[weight].sum()

    variance = 0.0
    for _, stratum in df.groupby("stratum"):
        areas = stratum.groupby("ea_id").apply(
            lambda g: (g[weight] * ((g[outcome] == "true") - p)).sum()
        )
        n = len(areas)
        if n < 2:
            continue
        variance += n / (n - 1) * ((areas - areas.mean()) ** 2).sum()
    variance /= total_w**2

    srs = p * (1 - p) / len(df)
    return p, np.sqrt(variance), variance / srs
```

```python
for outcome in ["food_insecure", "improved_water_source"]:
    p, se, deff = design_effect(survey, outcome)
    print(f"{outcome:24} p={p:.1%} se={se:.4f} deff={deff:.2f} "
          f"n_eff={len(survey) / deff:.0f}")
```

```r
library(survey)

design <- svydesign(ids = ~ea_id, strata = ~stratum, weights = ~weight,
                    data = survey, nest = TRUE)

svymean(~I(food_insecure == "true"), design, deff = TRUE)
svymean(~I(improved_water_source == "true"), design, deff = TRUE)
```

Outcome	Estimate	Design effect	Effective n
Food insecure	29.1%	**1.60**	623
Improved water source	69.8%	**0.91**	1,098

4.5 The one below one

Read the second row again. **A design effect of 0.91 means the survey is more precise than a simple random sample of the same size.**

That is not an error, and it is the most useful result in this lesson. Water access differs enormously between strata — 88%, 62%, 41% — and stratification guarantees all three are represented in the estimate in fixed proportions. The variance that a simple random sample carries from *not* knowing how many urban households it will happen to catch is removed entirely, and that gain outweighs the clustering loss.

Food insecurity also differs between strata, but its within-area correlation is higher, so clustering wins and the net is 1.60.

So "design effect" is not a synonym for "penalty". It is the net of two opposing forces, and which one wins depends on the outcome. A survey report quoting one design effect for the whole survey has averaged something that should not be averaged.

Note also that the clustering-only figure computed earlier was 2.4, and the full design effect for the same outcome is 1.60. **The difference is what stratification bought**, and it is worth reporting when you are defending a design.

4.6 Where design effects come from when you have none

You need one before the survey exists, and there is no data yet. In order of preference:

- **The previous round of the same survey**, same indicator, same area. Almost always available and almost always ignored.
- **A published survey report** for the same indicator in a comparable setting. DHS reports tabulate design effects; SMART plausibility reports state them.
- **A convention.** SMART surveys commonly assume 1.5 for anthropometry; humanitarian assessments often use 2.0 for household-level outcomes.
- **1.0** is never a defensible assumption for a cluster survey, and a protocol that uses it has under-sized the survey by half.

4.7 Report it

```python
summary = pd.DataFrame({
    "outcome": ["Food insecure", "Improved water source"],
    "estimate": ["29.1%", "69.8%"],
    "ci_95": ["25.5-32.7%", "67.0-72.5%"],
    "deff": [1.60, 0.91],
    "n": [996, 996],
    "n_effective": [623, 1098],
})
```

```r
tibble::tribble(
  ~outcome,                ~estimate, ~deff, ~n_effective,
  "Food insecure",           "29.1%",  1.60,          623,
  "Improved water source",   "69.8%",  0.91,         1098
)
```

Five columns, one row per indicator, and it belongs in the annex of every survey report. It lets a reader judge the estimate, compare it to another survey, and size the next round — none of which is possible from the percentage alone.

4.8 What comes next

You have a design effect, which means you have a standard error, which means you can stop computing these things by hand. The next lesson declares the design once to the software and lets every estimate inherit it — `survey` in R, and the equivalent arithmetic in Python.

Part III

Estimating properly

CHAPTER 5

Declare the design once

Lesson 5 of 8 · 120 min

One design object, and every estimate after it inherits the weights, the strata and the clusters. The R survey package, the equivalent arithmetic in Python, and the four mistakes that silently produce a plain average.

5.1 Stop carrying the design by hand

Everything so far has been computed explicitly, which was the point — you should know what the arithmetic is. From here it should be declared once and inherited, because the alternative is remembering to apply weights, strata and clusters to every estimate in a fifty-line script, and the estimate you forget is the one that goes in the summary.

5.2 The design object in R

```r
library(survey)

design <- svydesign(
  ids     = ~ea_id,      # the primary sampling unit
  strata  = ~stratum,    # the stratification
  weights = ~weight,     # the weight built in lesson 2
  data    = survey,
  nest    = TRUE         # cluster ids are nested within strata
)

design
```

Four arguments and every one is load-bearing.

- `ids` is the *primary* sampling unit — the enumeration area, not the household. Passing `~household_id` here is the commonest error in this course and it silently produces a simple-random-sample variance.
- `strata` removes the between-stratum variance, which is the gain the previous lesson measured.
- `weights` makes the point estimate unbiased.
- `nest = TRUE` tells the package that area identifiers are unique only within a stratum. Omit it where they are reused across strata and the areas get merged.

Then every estimate comes off the design.

```r
svymean(~I(food_insecure == "true"), design, deff = TRUE)
svytotal(~I(food_insecure == "true"), design)
svyby(~I(food_insecure == "true"), ~stratum, design, svymean, vartype = c("se", "ci"))
svyciprop(~I(food_insecure == "true"), design, method = "logit")
```

`svyby` is the workhorse: any estimate, by any grouping, with the design carried through. And `svyciprop` is what you want for a proportion — the next lesson explains why the interval from `svymean` is the wrong shape near 0 and 1.

5.3 The same thing in Python

Python has no equivalent of `survey` that this sector has settled on, so the examples compute the estimator directly. That is a cost and also a benefit: the arithmetic is visible.

```python
import numpy as np
import pandas as pd

class SurveyDesign:
    """Stratified two-stage design, ultimate-cluster variance."""

    def __init__(self, data, ids, strata, weights):
        self.data, self.ids, self.strata, self.weights = data, ids, strata, weights

    def mean(self, indicator):
        d = self.data
        y = indicator(d).astype(float)
        w = d[self.weights]
        p = np.average(y, weights=w)

        variance, total_w = 0.0, w.sum()
        for _, stratum in d.groupby(self.strata):
            residual = stratum[self.weights] * (indicator(stratum).astype(float) - p)
            totals = residual.groupby(stratum[self.ids]).sum()
            n = len(totals)
            if n < 2:
                continue
            variance += n / (n - 1) * ((totals - totals.mean()) ** 2).sum()
        se = np.sqrt(variance) / total_w

        srs = np.sqrt(p * (1 - p) / len(d))
        return {"estimate": p, "se": se, "deff": (se / srs) ** 2,
                "ci_low": p - 1.96 * se, "ci_high": p + 1.96 * se}

design = SurveyDesign(survey, ids="ea_id", strata="stratum", weights="weight")
print(design.mean(lambda d: d["food_insecure"] == "true"))
```

```r
# The R equivalent of the whole class above:
svymean(~I(food_insecure == "true"), design, deff = TRUE)
```

The Python class is thirty lines and the R call is one. That asymmetry is real and it is why survey analysis in this sector is usually done in R. **Use R for survey estimation where you have the choice**; the Python path is for teams that do not, and for understanding what the R call is doing.

5.4 The ultimate cluster approximation

Both implementations use the same shortcut, and it is worth naming because it looks like a simplification and is not.

The variance is computed from the **totals of the primary sampling units**, ignoring the second stage entirely. That is exact when the first stage is sampled with replacement, and

slightly conservative otherwise — it very slightly overstates the variance because it ignores the finite population correction at stage two.

Slightly conservative is the right direction to be wrong in, which is why every major package does this by default. `survey` calls it the ultimate cluster approximation and so does every DHS report.

5.5 The four ways to get a plain average by accident

Each of these produces a number that looks fine and is a simple random sample estimate.

Passing the household as the cluster. `ids = ~household_id` says each household is its own PSU, so there is no clustering to account for and the standard error collapses.

Forgetting the weights. `svydesign(..., weights = NULL)` gives every household equal weight, which is the 32.8% from lesson 1.

Filtering the data frame instead of subsetting the design. This one is subtle and common:

```r
# WRONG: the design object no longer knows about the dropped strata
rural <- svydesign(ids = ~ea_id, strata = ~stratum, weights = ~weight,
                   data = filter(survey, stratum != "urban"), nest = TRUE)

# RIGHT
rural <- subset(design, stratum != "urban")
```

```python
# Same rule: subset the design, keeping the full strata structure in view
rural = SurveyDesign(survey[survey["stratum"] != "urban"],
                     ids="ea_id", strata="stratum", weights="weight")
```

Subsetting a design keeps the strata and cluster structure intact for variance estimation. Rebuilding it from a filtered frame throws away areas with no remaining households, which changes the degrees of freedom and can silently produce a stratum with one cluster in it — where the variance contribution is undefined and quietly dropped.

Computing on a merged frame. Join the survey to anything one-to-many and the weights are now wrong, because a household appears several times. Aggregate to the household before estimating, which is the grain lesson from the joining course arriving in a new setting.

5.6 Estimates other than proportions

```r
svymean(~household_size, design, na.rm = TRUE)
svytotal(~I(food_insecure == "true"), design)
svyquantile(~minutes_to_water, design, quantiles = c(0.5, 0.9), na.rm = TRUE)
svyratio(~I(food_insecure == "true"), ~I(children_under5 > 0), design)
```

```python
print(design.mean(lambda d: d["household_size"] > 7))
```

Two notes. **A weighted total is the estimate people most often want and most often cannot get**, because it needs population-scaled weights — the normalised weights from lesson 2 will give a total of 996. And `na.rm = TRUE` on a survey mean is a decision, not a convenience: it assumes the missing households resemble the observed ones within their

stratum, which is the same assumption the non-response adjustment makes and should be stated once for both.

5.7 The sub-sample design

The measured children are a sub-sample with their own weight, and they need their own design object — not a filter on the household one.

```r
children <- subset(design, !is.na(child_muac_mm))
children <- update(children, child_weight = weight * children_under5)

child_design <- svydesign(ids = ~ea_id, strata = ~stratum,
                          weights = ~child_weight,
                          data = subset(survey, !is.na(child_muac_mm)), nest = TRUE)

svyciprop(~I(child_muac_mm < 125), child_design, method = "logit")
```

```python
measured = survey[survey["child_muac_mm"].notna()].copy()
measured["child_weight"] = measured["weight"] * measured["children_under5"]

child_design = SurveyDesign(measured, ids="ea_id", strata="stratum",
                            weights="child_weight")
print(child_design.mean(lambda d: d["child_muac_mm"] < 125))
```

The clusters are the same areas, so the clustering still applies. Only the weight and the population change.

5.8 Write the design down beside the estimates

```
Design          Stratified two-stage cluster
Strata          urban, rural-accessible, rural-remote
PSU             enumeration area (75 sampled, 25 per stratum)
Stage 1         PPS, size measure = households at frame listing
Stage 2         SRS, 14 households per area
Weights         base = M_h / (25 * 14), non-response adjusted within area
Variance        ultimate cluster, Taylor linearisation
Software        R survey 4.x / equivalent direct computation in Python
```

Seven lines. Any reader can now reproduce your standard errors, and any reviewer can tell in ten seconds whether you did it properly — which is more than can be said for most survey annexes.

5.9 What comes next

The design object assumes everyone you selected was interviewed and every area you drew was visited. Neither was true here. The next lesson handles the 54 households that were not interviewed and the two areas that had to be replaced.

CHAPTER 6

The households you did not get

Lesson 6 of 8 · 120 min

Fifty-four missing interviews concentrated in the stratum least like the others, two replacement clusters that change a probability nobody recomputes, and the assumption every adjustment rests on.

6.1 Non-response is not missing data

A missing value is a question that was not answered. Non-response is a **household that was selected and never interviewed**, and the difference matters because non-response leaves no row at all — the cleaning course's hardest case, arriving in a setting where the frame tells you exactly how many rows should be there.

```python
selected = frame[frame["selected"] == True]

response = selected.groupby("stratum").agg(
    selected=("households_selected", "sum"),
    interviewed=("households_interviewed", "sum"),
)
response["rate"] = response["interviewed"] / response["selected"]
print(response)
```

```r
frame |>
  filter(selected) |>
  summarise(selected = sum(households_selected),
            interviewed = sum(households_interviewed), .by = stratum) |>
  mutate(rate = interviewed / selected)
```

Stratum	Selected	Interviewed	Response
Urban	350	316	90.3%
Rural accessible	350	337	96.3%
Rural remote	350	343	98.0%

996 of 1,050. A 95% overall response rate reads as excellent, and the useful fact is not the average — it is that **the shortfall is concentrated in urban areas**, which are also the least food insecure by twenty points.

An unadjusted analysis therefore under-represents exactly the stratum that would pull the estimate down. The direction of the bias is predictable from that table alone, before any modelling.

6.2 The adjustment, and the assumption under it

The adjustment from lesson 2 inflates each responding household's weight to carry the non-respondents in its own area.

```python
survey["nr_adjustment"] = 14 / survey["ea_id"].map(
    selected.set_index("ea_id")["households_interviewed"]
)
print(survey.groupby("stratum")["nr_adjustment"].mean().round(3))
```

```r
survey |> summarise(mean_adjustment = mean(14 / households_interviewed), .by = stratum)
```

The assumption is that non-respondents resemble respondents in the same area. It is not free and it is not testable from the survey, so it belongs in the limitations section in words:

> Non-response was adjusted within enumeration area. Urban response was 90.3% against 96–98% rural. If urban non-respondents are systematically better off than urban respondents — locked compounds and daytime absence both suggest they may be — the adjusted estimate remains slightly too high.

Notice what that paragraph does. It names the direction of the residual bias after adjustment, which is the most a survey can honestly offer.

6.3 Adjust at the right level

Three levels are available and they are not equivalent.

- **Within area.** What this survey does. Best when response varies between areas, which it always does.
- **Within stratum.** Cruder, and it throws away the information that one area had four refusals and its neighbour none.
- **Within a response class.** Group areas by something that predicts response — urban density, market day, security — and adjust within the group. Better than stratum where areas are small, because an area with three interviews gives an unstable adjustment factor.

The last one matters here: an area with very few completed interviews gets a large adjustment from a small denominator, and one such area can move a stratum estimate. Check for it.

```python
weights_by_area = survey.groupby("ea_id")["weight"].first().sort_values()
print(weights_by_area.tail(3))
print(f"largest weight / smallest: {weights_by_area.max() / weights_by_area.min():.1f}x")
```

```r
survey |>
  summarise(weight = first(weight), n = n(), .by = ea_id) |>
  arrange(desc(weight)) |>
  head(3)
```

Trim or flag any weight far outside the rest. A weight three times its stratum's base is one area doing the work of three, and it inflates variance without adding information. Trimming is a judgement call with a cost — it introduces a small bias to remove a large variance — and like every other judgement in this course it goes in writing with its threshold.

6.4 Replacement clusters

Two rural remote areas could not be reached and were replaced. The frame records it.

```python
replacements = frame[frame["replacement_for"].notna()]
print(replacements[["ea_id", "stratum", "households", "replacement_for"]])
```

```r
frame |> filter(!is.na(replacement_for)) |>
  select(ea_id, stratum, households, replacement_for)
```

This is where a design quietly stops being the design that was documented, and there are two defensible treatments.

Treat the replacement as the original. The pragmatic choice, and what almost everyone does. It assumes the replacement is exchangeable with the area it replaced — reasonable when replacement was random within stratum, which the protocol should say and often does not.

Treat the original as non-response at the cluster level. More honest and harder: the stratum effectively has 23 responding clusters, not 25, and the weights for that stratum inflate accordingly.

```python
remote_areas = (frame["stratum"] == "rural-remote") & (frame["selected"] == True)
responding = int(remote_areas.sum()) - len(replacements)
print(f"25 selected, {responding} reached, cluster-level adjustment "
      f"{25 / responding:.3f}")
```

```r
c(selected = 25, reached = 23, adjustment = 25 / 23)
```

The difference on this survey is small. **The reason to do it explicitly is not the size of the correction; it is that inaccessible areas are never a random subset.** An area you cannot reach in February is remote, or insecure, or flooded, and those are precisely the conditions associated with the outcomes being measured. Replacing it with an accessible area systematically removes the worst cases.

> Report replacement even when you treat it as exchangeable. "Two of 25 rural remote clusters were replaced for inaccessibility; both replacements were accessible areas, so the rural remote estimate is likely to be conservative" is a sentence a reader needs and cannot reconstruct.

6.5 Response rates belong in the report

```python
report = response.reset_index()
report["non_response_bias_risk"] = ["high", "low", "low"]
report["adjustment"] = "within enumeration area"
print(report)
```

```r
tibble::tribble(
  ~stratum,            ~selected, ~interviewed, ~rate,  ~risk,
  "urban",                   350,          316, 0.903, "high",
  "rural-accessible",        350,          337, 0.963, "low",
  "rural-remote",            350,          343, 0.980, "low"
```

```
)
```

Four columns and a risk assessment. A survey report that gives one overall response rate has hidden the only thing about non-response that affects the estimate — where it fell.

6.6 When response is bad enough to stop

There is no universal threshold, and the honest guidance is comparative rather than absolute.

- **Above 90%** — adjust and report. Residual bias is usually small relative to sampling error.
- **80 to 90%** — adjust, report, and describe the likely direction of residual bias. Do not compare against a previous round with a very different response rate without saying so.
- **Below 80%** — the adjustment is carrying too much. Report the estimate with a prominent caveat, or report by stratum only where response was adequate.
- **Below 60% in a stratum** — that stratum's estimate is not usable and saying so is the correct finding.

The comparison that matters most is against the **previous round**. An estimate that moved five points between rounds, where response also moved fifteen points, has an obvious alternative explanation that must be addressed before any programmatic one.

6.7 What comes next

You have an estimate, a design and an honest account of what is missing from it. The next lesson turns that into the thing that gets published: an interval, of the right shape, with the right number of decimal places, and the sentence that goes around it.

Part IV

What you may publish

CHAPTER 7

The interval, and the sentence around it

Lesson 7 of 8 · 120 min

Why the textbook interval gives 0.88% for a proportion that cannot go below zero, the logit interval that fixes it, degrees of freedom you have 72 of, and why two overlapping intervals do not mean no difference.

7.1 What the interval is for

A survey estimate is a guess about a population, made from a sample. The confidence interval is the honest width of that guess, and publishing the point estimate without it is the single most common misuse of survey data in this sector.

The formal reading — "95% of intervals constructed this way would contain the true value" — is correct and unhelpful in a meeting. The operational reading is what you need: **the interval is the set of population values that would not be surprising given this sample.** If a threshold sits inside your interval, your survey cannot say which side of it the population is on.

7.2 The textbook interval, and where it fails

```
p +/- z * se
```

Fine in the middle of the range and wrong at the ends. Take severe acute malnutrition among the measured children:

```
p, se = 0.0217, 0.0066
print(f"Wald: {p - 1.96 * se:.2%} to {p + 1.96 * se:.2%}")
```

```
p <- 0.0217; se <- 0.0066
c(p - 1.96 * se, p + 1.96 * se)
```

2.17%, interval **0.88% to 3.46%**. Symmetric, and the symmetry is the problem: a proportion cannot be negative, and with a slightly smaller estimate this interval would extend below zero and be published anyway.

7.3 The logit interval

Transform to the log-odds scale, build the interval there, transform back.

```
import math

logit = math.log(p / (1 - p))
se_logit = se / (p * (1 - p))
low, high = (logit - 1.96 * se_logit, logit + 1.96 * se_logit)

print(f"logit: {1 / (1 + math.exp(-low)):.2%} to {1 / (1 + math.exp(-high)):.2%}")
```

```r
library(survey)
svyciprop(~I(child_muac_mm < 115), child_design, method = "logit")
```

1.19% to 3.92%. Asymmetric — further above the estimate than below — which is the correct shape for a small proportion, and it cannot escape the 0–1 range whatever the estimate.

Use the logit interval for any proportion below about 10% or above 90%. In the middle the two agree to a decimal place and it does not matter. `svyciprop(..., method = "logit")` in R does it directly; the Python examples in this course compute it as above.

7.4 Degrees of freedom you actually have

The 1.96 assumes a normal distribution, which assumes plenty of independent observations. In a cluster survey the independent units are the **clusters**, not the households.

```
df = number of PSUs - number of strata
```

```python
clusters = survey["ea_id"].nunique()
strata = survey["stratum"].nunique()
print(f"{clusters} clusters, {strata} strata, df = {clusters - strata}")
```

```r
c(clusters = n_distinct(survey$ea_id), strata = n_distinct(survey$stratum))
degf(design)
```

Seventy-five clusters, three strata, **72 degrees of freedom**, and t(0.975, 72) is 1.993 rather than 1.96. A difference of 2% in the interval width, which here is irrelevant and is exactly the kind of thing that stops being irrelevant fast: a survey with twelve clusters has nine degrees of freedom and a multiplier of 2.26, which is 15% wider than the normal approximation.

Use the t multiplier, and report the degrees of freedom. It costs nothing and it tells a reader how many clusters you had without them having to ask.

7.5 Overlapping intervals are not a test

The most consequential misreading in this lesson.

```python
for stratum in ["urban", "rural-accessible", "rural-remote"]:
    print(stratum)   # displaced households
```

Stratum	Displaced	95% interval
Urban	15.6%	12.0 – 19.2%
Rural accessible	8.0%	4.7 – 11.3%
Rural remote	15.7%	11.0 – 20.4%

Urban and rural accessible intervals do not overlap, so those two differ. Urban and rural remote overlap almost entirely, so those two do not differ detectably.

Now the trap: **two intervals that overlap slightly can still be significantly different.** Comparing intervals is a cruder test than comparing the difference, because the difference has its own standard error which is smaller than the sum of the two.

```
svyttest(I(displacement_status == "displaced") ~ I(stratum == "urban"),
         subset(design, stratum != "rural-remote"))
```

```
# The difference of two estimates, with its own standard error
diff = p_urban - p_rural
se_diff = math.sqrt(se_urban**2 + se_rural**2)
print(f"difference {diff:.1%}, 95% CI {diff - 1.99 * se_diff:.1%} to "
      f"{diff + 1.99 * se_diff:.1%}")
```

The rule: **to compare two estimates, estimate the difference and put an interval on that.** Reading two intervals is a screening step, not a conclusion.

Note also that the Python line above assumes the two estimates are independent, which is true for separate strata and false for two subgroups within a stratum — in that case the covariance term matters and `svyttest` or `svycontrast` handles it properly.

7.6 Rounding, and false precision

An interval of ±3.6 points does not support three decimal places.

```
print(f"{p:.1%} (95% CI {low:.1%} to {high:.1%})")
```

```
sprintf("%.1f%% (95%% CI %.1f–%.1f)", 100 * p, 100 * low, 100 * high)
```

- **One decimal place** for a percentage from a survey of this size. Two implies a precision of 0.01 points, which is a hundred times narrower than your interval.
- **Round the interval outwards**, never inwards. 25.47–32.71 becomes 25.4–32.8.
- **Never round the point estimate to a threshold.** An estimate of 14.96% reported as "15%" beside a 15% emergency threshold has made a classification decision by rounding.

7.7 The sentence

The number, the interval, the denominator, the design. One sentence, and it is the deliverable of this whole course.

> Household food insecurity was estimated at **29.1%** (95% CI 25.5–32.7), from 996 households in 75 clusters, weighted to the sampling frame and adjusted for non-response; design effect 1.60, effective sample 623.

Compare that with what usually gets published:

> Food insecurity affects 32.8% of households.

The second sentence is unweighted, has no interval, no denominator and no design. It is also 3.7 points higher, and it is the one that will be quoted.

7.8 Reading an interval against a threshold

The question this sector asks most often, and the one the interval exists for.

```
THRESHOLD = 0.15

if high < THRESHOLD:
    verdict = "below the threshold"
elif low > THRESHOLD:
    verdict = "above the threshold"
else:
    verdict = "cannot be distinguished from the threshold"
print(verdict)
```

```
dplyr::case_when(
  high < 0.15 ~ "below the threshold",
  low  > 0.15 ~ "above the threshold",
  TRUE        ~ "cannot be distinguished from the threshold"
)
```

The third branch is the one people delete, and it is the most common honest answer. **An estimate of 14.2% with an interval of 11.0–18.1 does not say the situation is below the emergency threshold**; it says the survey cannot tell. That sentence is unwelcome, correct, and considerably cheaper than a response scaled on a number that could not support the decision.

7.9 What comes next

The interval tells you what your whole sample supports. Cut the sample into districts, age groups and sexes and each piece supports far less. The last lesson is where to stop cutting, and how to say so in a table.

CHAPTER 8

Where to stop cutting

Lesson 8 of 8 · 120 min

Three strata support an estimate. Stratum by displacement status gives nine cells with eighteen households in the smallest. A rule set before you look, and the table that shows a reader where the survey ran out.

8.1 The request that always arrives

The survey report goes out with three stratum estimates. Within a week somebody asks for it by displacement status. Then by sex of head of household. Then for displaced female-headed households in rural remote areas, because that is the group the next proposal is about.

Each request is reasonable and the last one is unanswerable, and the difficulty is that **nothing in the software refuses.** Every cut produces a percentage.

8.2 What the cutting does

```python
for keys in [["stratum"],
             ["stratum", "displacement_status"],
             ["stratum", "main_livelihood"]]:
    cells = survey.groupby(keys).size()
    print(f"{' x '.join(keys):40} {len(cells):>3} cells, "
          f"smallest {cells.min():>3}, {(cells < 50).sum()} below 50")
```

```sql
survey |> count(stratum) |> summarise(cells = n(), smallest = min(n))
survey |> count(stratum, displacement_status) |> summarise(cells = n(), smallest = min(n))
survey |> count(stratum, main_livelihood) |> summarise(cells = n(), smallest = min(n))
```

Disaggregation	Cells	Smallest	Below 50
Stratum	3	316	0
Stratum × displacement	9	18	5
Stratum × livelihood	18	6	9

One additional variable takes the smallest cell from 316 households to 18. A second takes it to 6.

And the household count is the *optimistic* view, because these are clustered observations. Eighteen households in a cell may come from four enumeration areas, which is four independent units and about nine degrees of freedom — not eighteen.

8.3 Count clusters, not just households

```python
cells = survey.groupby(["stratum", "displacement_status"]).agg(
    households=("household_id", "size"),
    clusters=("ea_id", "nunique"),
```

```
)
print(cells.sort_values("clusters").head())
```

```
survey |>
  summarise(households = n(), clusters = n_distinct(ea_id),
            .by = c(stratum, displacement_status)) |>
  arrange(clusters)
```

A cell drawn from fewer than about ten clusters cannot support a variance estimate you would want to defend, however many households are in it. This is the survey-specific version of the minimum cell size the indicator course introduced, and it is stricter, because clustering means the effective sample is smaller than the count.

Two cells in this survey come from a single cluster. A variance estimate from one cluster is undefined, and most software will either drop the cell silently or return zero — a standard error of zero on a subgroup estimate is always a bug, and it is always this one.

8.4 Set the rule before you look

Three thresholds, decided in advance and written into the analysis plan.

```
MIN_HOUSEHOLDS = 50
MIN_CLUSTERS = 10
MAX_CI_WIDTH = 0.20      # 20 percentage points

def reportable(cell):
    return (cell["households"] >= MIN_HOUSEHOLDS
            and cell["clusters"] >= MIN_CLUSTERS
            and cell["ci_width"] <= MAX_CI_WIDTH)
```

```
MIN_HOUSEHOLDS <- 50; MIN_CLUSTERS <- 10; MAX_CI_WIDTH <- 0.20

reportable <- function(d) {
  d$households >= MIN_HOUSEHOLDS & d$clusters >= MIN_CLUSTERS &
    d$ci_width <= MAX_CI_WIDTH
}
```

The third condition is the one that does the real work, and it is better than the first two because it is about the answer rather than about the input. **An interval wider than 20 points cannot distinguish "a serious problem" from "not a problem"**, so publishing the midpoint invites a decision the number cannot support.

Where a threshold in the sector is what matters, set the width against the threshold instead: an interval that has to sit on one side of 15% needs to be narrower than the distance from the estimate to 15%.

8.5 Suppress, but show the cell

```
by_cell = svyby_equivalent(survey, ["stratum", "displacement_status"], "food_insecure")
by_cell["reported"] = by_cell.apply(reportable, axis=1)
by_cell.loc[~by_cell["reported"], ["estimate", "ci_low", "ci_high"]] = None
by_cell["note"] = by_cell["reported"].map(
    {False: "too few clusters to estimate", True: ""}
)
print(by_cell)
```

```
by_cell <- svyby(~I(food_insecure == "true"), ~stratum + displacement_status,
                 design, svymean, vartype = "ci") |>
  mutate(reported = reportable(cur_data()),
         note = if_else(reported, "", "too few clusters to estimate"))
```

Never delete the row. A suppressed cell that still shows its household count and a reason tells the reader the group was surveyed and the survey could not answer for it. A deleted row reads as though the group does not exist, and the next person to ask will ask again.

This is the same rule the DQA course applied to unmeasurable dimensions and the indicator course applied to small cells. It keeps arriving because it is the same underlying discipline: absence of evidence has to look different from absence.

8.6 Domain estimation, and the thing that looks like a filter

One technical point that changes the answer.

```
# WRONG: rebuilding the design from a filtered frame
displaced <- svydesign(ids = ~ea_id, strata = ~stratum, weights = ~weight,
                       data = filter(survey, displacement_status == "displaced"),
                       nest = TRUE)

# RIGHT: a domain estimate keeps the full design in view
svyby(~I(food_insecure == "true"), ~displacement_status, design, svymean)
```

```
# The same rule: the variance calculation must still see every cluster,
# including those with no displaced households in them.
```

A subgroup that does not span every cluster is a **domain**, not a sub-population with its own design. Estimating it from a filtered frame loses the clusters that contain none of the subgroup, and those empty clusters carry real information about the variance of the subgroup's size. The R `survey` package handles this correctly through `svyby` on the full design; a filtered rebuild does not.

The effect is usually modest and always in the same direction: **the filtered version understates the standard error**, which is the direction that makes you overconfident.

8.7 Plan the disaggregation before the survey

The honest fix for all of this is upstream. The indicator course made the point in general; here it has a number attached.

If a subgroup is 12% of the population and you need ±10 points on it, the sample size formula run on that subgroup gives what you need — and it is usually far more than the survey was sized for. That is a design conversation, and the options are real:

- **Oversample the subgroup**, which is what stratification exists for. This survey oversampled rural remote precisely so it could be reported on.
- **Accept a wider interval** for that subgroup, stated in advance.
- **Drop the requirement**, and say in the protocol that the survey will not report on it.

What you cannot do is decide afterwards. By the time the data exists the cells are whatever they are, and a table showing eight suppressed cells out of eighteen is the visible

form of a design conversation that did not happen.

8.8 The table to publish

```
final = by_cell[["stratum", "displacement_status", "households", "clusters",
                 "estimate", "ci_low", "ci_high", "note"]]
final.to_csv("outputs/tables/food_insecurity_by_stratum_displacement.csv", index=False)
```

```
readr::write_csv(by_cell,
  here::here("outputs", "tables", "food_insecurity_by_stratum_displacement.csv"))
```

Stratum	Status	Households	Clusters	Estimate	95% CI	Note
Urban	Resident	246	25	13.9%	10.2–18.7	
Urban	Displaced	49	18	—	—	too few households
Rural remote	Resident	279	25	46.1%	40.0–52.3	
Rural remote	Returnee	24	12	—	—	too few households

Seven columns, and the two empty ones are as informative as the four filled ones. A reader can see exactly where the survey ran out, which is the difference between a table that answers questions and one that generates them.

8.9 Where this course leaves you

You can reconstruct weights from a frame and prove they sum to the population, size a survey against a precision requirement and read that requirement backwards from a survey you inherited, measure a design effect rather than assuming one, estimate with the design carried through, handle non-response and replacement explicitly, publish an interval of the right shape, and say where the sample stops supporting a cut.

That is the whole of what a survey analyst is judged on, and it is most of module

1. The last course in the module, *Routine Data and DHIS2*, goes back to the

aggregate reporting systems this programme has been borrowing from since module 2 — data elements, category combinations, the org unit hierarchy — and answers the question a coverage figure always provokes: what exactly was counted, and by whom?

About this edition

This PDF is generated from the same source as the web version of the course, so the two cannot drift. The web edition carries the runnable code, the downloadable datasets and any corrections issued since this file was produced.

> Every dataset referenced in this course is synthetic or fully de-identified. No record traces to a real person.

This course is published in English and in French. The French edition is a professional adaptation using the terminology UNICEF, WHO, WFP, OCHA and national ministries publish in — not a literal translation.

Read and run the course online at data-analysis.cassion.dev/courses/survey-analysis-sampling

Generated July 28, 2026.