Probability Sampling

Corinne Riddell (Instructor: Tomer Altman)

September 17, 2025

Learning objectives

- Learn how to take a random sample (i.e., a probability sample) in `R`
- Know the difference between a simple random sample, a proportionate stratified sample, a disproportionate stratified sample, and a multi-stage sample
- Know how to take a simple random sample and a proportionate stratified sample in `R`
- Introduce the idea of probability
- See how the quality of a probability estimate is related to the size of the sample chosen from the underlying population

Recall from last class

- Last class we discussed non-probability studies (e.g., convenience samples) and how they are not necessarily representative of an underlying population from which they were sampled
- To obtain a more representative sample, we can take a **probability sample**
- The most common type of probability sample is a **simple random sample** (SRS)

Simple random sample (SRS)

- Simple random sample (SRS): A sample chosen by chance, where each individual in the data set has the same chance of being selected
- We can easily choose an SRS from a data frame in `R`

Example of SRS in R

- First read in the hospital cesarean data

```r
CS_data <- read_xlsx("./data/kozhimannil.xlsx", sheet = 1)
CS_data <- CS_data %>% mutate(ID = row_number())
head(CS_data)
```

```
## # A tibble: 6 x 7
##   Births HOSP_BEDSIZE cesarean_rate lowrisk_cesarean_rate `Cesarean rate *100`
##    <dbl>        <dbl>         <dbl>                 <dbl>                <dbl>
## 1    767            1         0.344                0.107                  34.4
## 2    183            1         0.454                0.186                  45.4
## 3    668            1         0.430                0.195                  43.0
## 4    154            1         0.279                0.0844                 27.9
## 5    327            1         0.306                0.119                  30.6
## 6   2356            1         0.301                0.0662                 30.1
## # i 2 more variables: `Low Risk Cesarean rate*100` <dbl>, ID <int>
```

```r
dim(CS_data)
```

```
## [1] 580   7
```

Example of SRS in R

In the following code chunk, we use `slice_sample(n = 100)` to take an SRS of 100 records (here, hospitals) from `CS_data`:

```
CS_100_1 <- CS_data %>% slice_sample(n=100)
```

Suppose we took a second sample…

```
CS_100_2 <- CS_data %>% slice_sample(n=100)
```

Do you expect `head(CS_100_1)` to equal `head(CS_100_2)`?

Example of SRS in R

```
head(CS_100_1 %>% select(Births, HOSP_BEDSIZE, cesarean_rate, ID))
```

```
## # A tibble: 6 x 4
##   Births HOSP_BEDSIZE cesarean_rate    ID
##    <dbl>        <dbl>         <dbl> <int>
## 1   6857            2         0.418   156
## 2    343            1         0.292   112
## 3    731            3         0.269   330
## 4   1136            3         0.298   369
## 5    928            2         0.356   286
## 6    282            3         0.294   580
```

Example of SRS in R

```
head(CS_100_2 %>% select(Births, HOSP_BEDSIZE, cesarean_rate, ID))
```

```
## # A tibble: 6 x 4
##   Births HOSP_BEDSIZE cesarean_rate    ID
##    <dbl>        <dbl>         <dbl> <int>
## 1   1793            3         0.392   542
## 2    958            3         0.399   454
## 3    587            1         0.358    26
## 4    619            2         0.292   285
## 5   2399            3         0.274   350
## 6    274            2         0.354   176
```

Example of SRS in R

```
identical(CS_100_1, CS_100_2)
```

```
## [1] FALSE
```

Example of SRS in R

Question: Why are the first six lines different, when the `slice_sample(n=100)` code is the same? So far with the functions we've learned in class, if we wrote the same line of code, we would generate the same result. What is going on?

Answer: Anytime you do something in R that involves a *random* process, the results will be different (i.e., non-deterministic). This is a good thing! This allows you to pick many different random samples. In future weeks we will do this many times.

Check your understanding!

Example of SRS in R

What if you want to ensure that you pick the same SRS as a friend? Or you want to make sure that you can reproduce a great result that you found before?

Then you need to use `set.seed()`. Here we set the seed to `123` before running the sampling code. This ensures that the random samples generated are the same.

Try to run this code again with a number other than `123`:

```
set.seed(123)
CS_100_1 <- CS_data %>% slice_sample(n=100)

set.seed(123)
CS_100_2 <- CS_data %>% slice_sample(n=100)

identical(CS_100_1, CS_100_2)
```

```
## [1] TRUE
```

SRS by Proportion in R

Another way to take a random sample is to specify the **prop**ortion of the dataset that you'd like to include in your sample using `slice_sample(prop = 0.05)`. Here, we sample 5% of the records (here, hospitals):

```
CS_5percent <- CS_data %>% slice_sample(prop = 0.05)
```

Proportionate Stratified Sampling in R

Suppose you want to take a 10% sample of individuals from within each county in California. This is called a "proportionate stratified sample". To do this you would use the following code:

```
CA_data %>% group_by(county) %>% slice_sample(prop = 0.1)
```

Here is an example using the `CS_data`:

```
## Perform a proportionate stratified sampling of the CS_data by HOSP_BEDSIZE
CS_10percent_grouped <- CS_data %>%
  group_by(HOSP_BEDSIZE) %>%
  slice_sample(prop = 0.1)

dim(CS_10percent_grouped)
```

```
## [1] 57  7
```

Proportionate Stratified sampling in R

How to check you really did sample 10% of each `HOSP_BEDSIZE` group?

First see how many hospitals fall into each category in the original data

```
CS_data %>% group_by(HOSP_BEDSIZE) %>% summarize(counts = n())
```

```
## # A tibble: 3 x 2
##   HOSP_BEDSIZE counts
##          <dbl>  <int>
## 1            1    131
## 2            2    179
## 3            3    270
```

Then in the sample:

```
CS_10percent_grouped%>%group_by(HOSP_BEDSIZE) %>% count()
```

```
## # A tibble: 3 x 2
## # Groups:   HOSP_BEDSIZE [3]
##   HOSP_BEDSIZE     n
##          <dbl> <int>
## 1            1    13
## 2            2    17
## 3            3    27
```

(`summarize(counts = n())` is almost the same as `count()`)

In this example, proportionate stratified SRS assembles a sample that maintains the relative proportions of `HOSP_BEDSIZE` in the chosen sample compared to the population.

Disproportionate Stratified Sampling in R

- When might you want to over-represent certain groups?
- Example: Estimating infant mortality by race/ethnicity when some race/ethnic groups are very small (e.g., indigenous groups in U.S./Canada)
- Then, you may want to over-sample certain groups so you can better estimate infant mortality in those groups than if you sampled proportionately
- In this case, you could filter your sample into different race/ethnic groups and take samples of different sizes (or fractions) from each group. You won't be asked how to do this in R, but rather, know what a disproportionate stratified sample is and why you might want to take this kind of sample.

Multi-Stage Sampling

- **Multi-stage sampling** occurs when you first sample a clustering unit and then within the *clustering unit* select individuals
- Examples of **clustering units** include hospitals, schools, counties, etc.
- For example, sampling schools using an SRS, and then sampling students within those schools.
- Think about the differences between taking a multi-stage sample of students within schools vs. an SRS of students across a set of schools. How would these samples look different?
- Suppose you are taking measurements from blood tests from the students, which sample is more practical to conduct? Which one allows you to estimate some quantities for all the schools?

Applied Example: Using a Sample to Estimate Low Birthweight in U.S. Territories

Description of the Data

- These data are births by place of occurrence for U.S. territories (American Samoa, Guam, North Mariana Islands, Puerto Rico, and U.S. Virgin Islands) from the year 2015

- This is a subset of the data downloaded from the National Bureau of Economic Research

- You can find more information about the data set in the User Guide

Data Dictionary

Here is the data dictionary for this dataset:

Variable	Description
babyID	Unique identifier: row number
dbwt	Birth weight in Grams: 227-8165 grams
combgest	Combined gestation, in weeks: 17th to 47th week of gestation

Variable	Description
sex	Assigned sex at birth: M (Male) or F (Female)
dob_mm	Birth month
cig_rec	If the mother reports smoking in any of the three trimesters of pregnancy she is classified as a smoker: (Y) Yes, (N) No, or (U) Unknown

Import the Data into R

```r
library(tidyverse)
birth_data <- read_csv(file = "./data/L03_US-territories-births.csv")
birth_data <- birth_data%>% select(-...1)
```

```r
head(birth_data)
```

```
## # A tibble: 6 x 6
##   babyID  dbwt combgest sex   dob_mm cig_rec
##    <dbl> <dbl>    <dbl> <chr>  <dbl> <chr>
## 1      1  2977       37 M          1 N
## 2      2  3191       41 M          1 Y
## 3      3  1786       32 F          1 N
## 4      4  4489       39 M          1 N
## 5      5  3203       38 M          1 N
## 6      6  3203       39 F          1 N
```

Overview of the Applied Example

- Take a **simple random sample** from the **population** of births. We will use this sample to *estimate* the **sample proportion** of babies with low birth weight in the population.
- To know how close our **estimate** is to the **true proportion**, we will first calculate the **population proportion** of an infant being born less that 5 pounds 8 ounces, or 2,500 grams, which is the traditional cutoff used to classify an infant as having low birth weight.
- In real life settings, we would not know the **true** proportion, we would only know the value we estimate from our sample. But here, we can investigate how close the sample estimate is to the true parameter and see what we can do to make our estimate even better.
- We will see how well we can estimate the true probability based on random samples of varying sizes.

Step 1: Add a Variable to the Dataset for Low Birth Weight (LBW)

```r
birth_data <- birth_data %>% mutate(lbw = dbwt < 2500)
```

- What does the variable `lbw` store?
- `lbw` stores "logical" values, which means it is either equal to `TRUE` or `FALSE`
- Variables that store only two values are called **binary** variables. They are most commonly stored as logical data (TRUE/FALSE), numeric (0/1) or categorical ("Yes"/"No").
- In the above code, `R` evaluates whether birth weight (`dbwt`) is less than 2,500 grams for each birth. If it is, then `lbw = TRUE` for that birth, and if not then `lbw = FALSE` for that birth.

Step 2: Calculate the Proportion of Low Birth Weight Infants in Population Across All U.S. Territories

```r
lbw_population <- birth_data %>% summarize(true_prob_lbw = mean(lbw))
lbw_population
```

```
## # A tibble: 1 x 1
##   true_prob_lbw
##           <dbl>
## 1         0.102
```

- Question: How did we take a mean of values equal to `TRUE` or `FALSE`?
- Answer: `R` treats `TRUE` as equivalent to `1` and `FALSE` as equivalent to `0`. The mean of a variable coded as 0 or 1 is the **proportion** (or **probability**) of individuals who have low birth weight.
- Remember: We do not usually know the true value because we rarely have data on every individual in a population

Step 3: Take a Random Sample of Size n=10

```r
random_sample_n10 <- birth_data %>%
  slice_sample(n = 10) %>%
  mutate(sample_size = n())
#the last line of code adds the sample size to every row of the new data frame
#we will want to reference this information later
```

Step 4: Progressively Increase the Sample Size and Store those Samples

Sample the rows of data using the following sample sizes. Assign each of your samples to a different R object.

```
1. 10
2. 50
3. 100
4. 200
5. 500
6. 1000
7. 5000
8. 10000
9. 36724 (i.e, the entire target population)
```

Step 4 Code

```r
random_sample_n50 <- birth_data %>%   slice_sample(n = 50) %>% mutate(sample_size = n())

random_sample_n100 <- birth_data %>% slice_sample(n = 100) %>% mutate(sample_size = n())

random_sample_n200 <- birth_data %>% slice_sample(n = 200) %>% mutate(sample_size = n())

random_sample_n500 <- birth_data %>% slice_sample(n = 500) %>% mutate(sample_size = n())

random_sample_n1000 <- birth_data %>% slice_sample(n = 1000) %>% mutate(sample_size = n())

random_sample_n5000 <- birth_data %>% slice_sample(n = 5000) %>% mutate(sample_size = n())

random_sample_n10000 <- birth_data %>% slice_sample(n = 10000) %>% mutate(sample_size = n())

whole_pop <- birth_data %>% slice_sample(n = nrow(birth_data)) %>% mutate(sample_size = n())
```

Step 4 Side Note

By default `slice_sample(n=100)` takes a sample of size 100 **without replacement**. This means that each individual can only be included in the sample at most one time.

In future classes, we will introduce methods where we select a sample **with replacement**. We will talk more about how this works in the coming weeks.

Step 5: Calculate the Estimate of the Proportion of LBW for Each Random Sample

- For each sample, we want to estimate the proportion of `LBW` babies to see how much it differs from the true proportion in the entire population
- We could do this using `summarize` for each sample and by writing that code ten times, but there is an easier way

Step 5: Code to Estimate the Proportions More Efficiently

The function `bind_rows(df1, df2, df3, ...)` can be used to stack multiple data frames (e.g. df1, df2, df3, . . .) on top of each other when they all contain the same variables. Bind together the nine data frames created in the previous code chunk using `bind_rows()`and assign the stacked data frame the name `stacked_samples`:

```
stacked_samples <- bind_rows(random_sample_n10, random_sample_n50,
                             random_sample_n100, random_sample_n200,
                             random_sample_n500, random_sample_n1000,
                             random_sample_n5000, random_sample_n10000,
                             whole_pop)
```

Step 5: Code to Estimate the Proportions More Efficiently

Estimate the proportion of babies with low birth weight using each of your samples in `stacked_samples`. Hint: `group_by()` and `summarize()` will come in handy! Assign the output to a data frame called `sample_estimates`

```
sample_estimates <- stacked_samples %>%
  group_by(sample_size) %>%
  summarize(estimated_proportion_lbw = mean(lbw))

sample_estimates
```

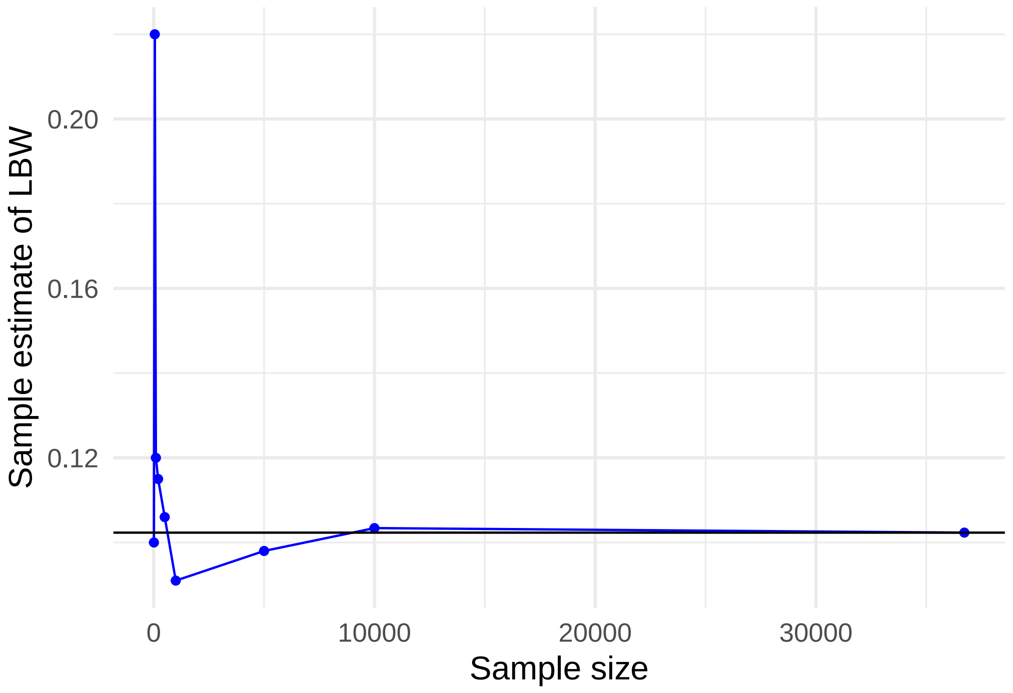
```
## # A tibble: 9 x 2
##   sample_size estimated_proportion_lbw
##         <int>                    <dbl>
## 1          10                    0.1
## 2          50                    0.22
## 3         100                    0.12
## 4         200                    0.115
## 5         500                    0.106
## 6        1000                    0.091
## 7        5000                    0.098
## 8       10000                    0.103
## 9       36724                    0.102
```

Step 6: Visualize the Results

- Make a line plot of the estimates of the proportions versus the sample size
- Add a horizontal line to the line plot at the **population proportion** that we are striving to estimate
- We also add points on top of the line to see exactly where the estimates are

```
ggplot(sample_estimates, aes(x = sample_size, y = estimated_proportion_lbw)) +
  geom_line(col = "blue") +
  geom_point(col = "blue") +
  geom_hline(yintercept = lbw_population %>% pull(true_prob_lbw)) +
```

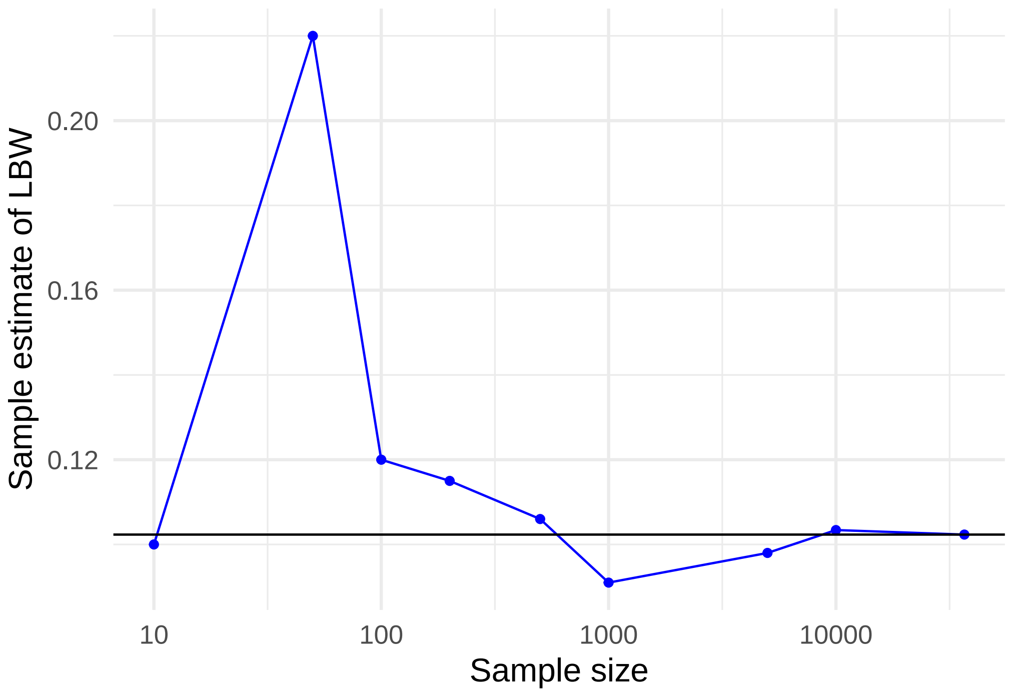
```r
  labs(y = "Sample estimate of LBW", x = "Sample size") +
  theme_minimal(base_size = 15)
```



Step 6: Visualize the Results

- Because the scale of the x axis is so large, try using `scale_x_log10()` to convert the x-axis to a logarithmic scale

```r
ggplot(sample_estimates, aes(x = sample_size, y = estimated_proportion_lbw)) +
  geom_line(col = "blue") +
  geom_point(col = "blue") +
  geom_hline(yintercept = lbw_population %>% pull(true_prob_lbw)) +
  scale_x_log10() +
  labs(y = "Sample estimate of LBW", x = "Sample size") +
  theme_minimal(base_size = 15)
```



Check Your Understanding!

1) What happens as the sample size increases?
2) Will it **always** be the case that a higher sample size produces an estimate closer to the true value than a lower sample size?