Discussion 5

Sampling and distribution plots

Jingwei Xiong

UCD statistics

2023-01-31

Case study

What was covered in this section:

- Random variable generator in R
- Sample in R
- `bind_rows` in `tidyverse` for binding datasets rows
- GGplot histogram, curves
- ggpubr for combining plots

Remember: If there is an error: cannot find function xxx, you forget to library some package!

Sampling

- In the following scenarios we'll explore through simulation from datasets.
- Nonrandom sampling can produce datasets with statistical properties that are distored relative to the population that the sample was drawn from. This kind of distortion is known as **bias**.
- biased sampling favors certain observational units over others, and biased estimates are estimates that favor larger or smaller values than the truth.
- Today we will explore sampling bias, and learn distribution plots from the case study.

Background

- Sampling design

The **sampling design** of a study refers to **the way observational units are selected** from the sampling frame (the collection of all observable units).

> Any design can be expressed by the probability that each unit is included in the sample. In a random sample, all units are equally likely to be included.

- Bias

Formally, **bias** describes **the 'typical' deviation of a sample statistic (observed) from its population counterpart (unobserved)**.

For example, if a particular sampling design tends to produce an average measurement around 1.5 units, but the true average in the population is 2 units, then the estimate has a bias of -0.5 units.

- Simulated data

Simulation is a great means of exploration for the present topic because you **can control the population properties**.

> With simulated data, by contrast, you control how data are generated with exact precision -- so by extension, you know everything there is to know about the population. In addition, repeated simulation of data makes it possible to explore the typical behavior of a particular sampling design, so you can learn 'what usually happens' for a particular sampling design by direct observation.

Scenario 1: unbiased samples

Here we will compare the sample mean and the distribution of sample values for a single viariable with the population mean.

Hypothetical population

To provide a little context to this scenario, imagine that you're measuring eucalyptus seeds to determine their typical diameter.

Here we simulates diameter measurements for a population of 5000 seeds; imagine that this is the total number of seeds in a small grove at some point in time.

```
population = data.frame(
  diameter = rgamma(n = 5000, shape = 2, scale = 1/2),
  seed = 1:5000
)
```

R random variables

```
population = data.frame(
  diameter = rgamma(n = 5000, shape = 2, rate = 2),
  seed = 1:5000
)
head(population,3)
```

```
   diameter seed
1 0.4221377    1
2 1.5845365    2
3 3.0998176    3
```

Here we generate 5000 hypothetical seeds follow a gamma distribution $\Gamma(\alpha = 2, \beta = 2)$. If we run the code again we will find the result will not be the same.

```
  diameter seed
1 1.363150    1
2 1.221888    2
3 1.056683    3
```

Random seed

This is because we are generating **random numbers**. In fact, the R generate pseudo-random numbers from a given seed. We can use a specific function `set.seed` to fix the randomness. For example:

```
set.seed(2023)
population = data.frame(
  diameter = rgamma(n = 5000, shape = 2, rate = 2),
  seed = 1:5000
)
head(population,3)
```

```
   diameter seed
1 0.6995702    1
2 0.5973719    2
3 0.6403414    3
```

This ensures that every time the code following `set.seed` is run, the same results will be observed.

R Functions for Probability Distributions

Every distribution that R handles has four functions. There is a root name, for example, the root name for the normal distribution is `norm`. This root is prefixed by one of the letters.

- `p` for "probability", the cumulative distribution function (c. d. f.)
- `q` for "quantile", the inverse c. d. f.
- `d` for "density", the density function (p. m. f. or p. d. f.)
- `r` for "random", a random generator having the specified distribution.

For the normal distribution, these functions are `pnorm`, `qnorm`, `dnorm`, and `rnorm`. For the binomial distribution, these functions are `pbinom`, `qbinom`, `dbinom`, and `rbinom`. And so forth.

Here you can find the distribution and their function names. We won't have time to explain all of them, you can look up the help function **?** for their arguments.

Now back to our example.

We have already learned how to use **tidyverse** to summarize our data. Now we can use **knitr::kable** to generate table. You can use that in any Rmd file to replace the original R output.

```
library(tidyverse)
set.seed(2023)
population = data.frame(
  diameter = rgamma(n = 5000, shape = 2, rate = 2),
  seed = 1:5000
)
population |>
  summarise(mean = mean(diameter), sd = sd(diameter)) |>
  knitr::kable(digit = 4, caption = "summary statistics of populatior
```

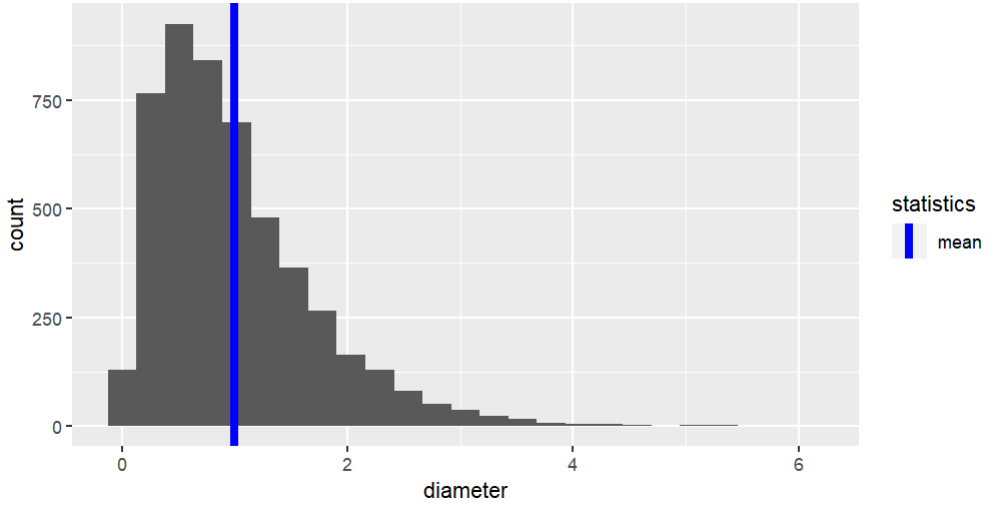
Table: summary statistics of population

mean	sd
0.9986	0.7174

Histogram

Just a summary table is far from enough. Now we can also use ggplot to generate histogram for our population diameter:

```
library(ggplot2)
population |>
  ggplot(aes(x = diameter)) +
  geom_histogram(bins = 25) +
  geom_vline(aes(xintercept = mean(population$diameter),
                 color = "mean"), size=2) +
  scale_color_manual(name = "statistics", values = c(mean = "blue"))
```



Unbiased sample design

Imagine that your sampling design involves collecting bunches of plant material from several locations in the grove until you obtaining 250 seeds.

We'll suppose that any of the 5000 seeds is equally likely to be obtained, so that your 250 seeds comprise a **random sample** of the population.

We can simulate samples by drawing values **without replacement** from the population. For a tidy dataframe, we can use **sample_n** from **tidyverse**

```
set.seed(2023)
sample = population |> sample_n(size = 250, replace = FALSE)
```

For a vector, we can use **sample**

```
sample(population$diameter, size = 250, replace = FALSE)
```

bind_rows

Now we can quickly compute the sample mean and sample standard deviation. Here we first construct sample and population inside 1 tidy dataset adding additional column **Group** indicating sample or population. Now with this dataset we can easily groupby and summarise.

```
df = bind_rows(lst(population, sample), .id = 'Group')
  # bind_rows: lst(dataset1, dataset2), .id = "name for new column"
  # After the first step, the Sample column 1
head(df, 1)
```

```
       Group  diameter seed
1 population 0.6995702    1
```

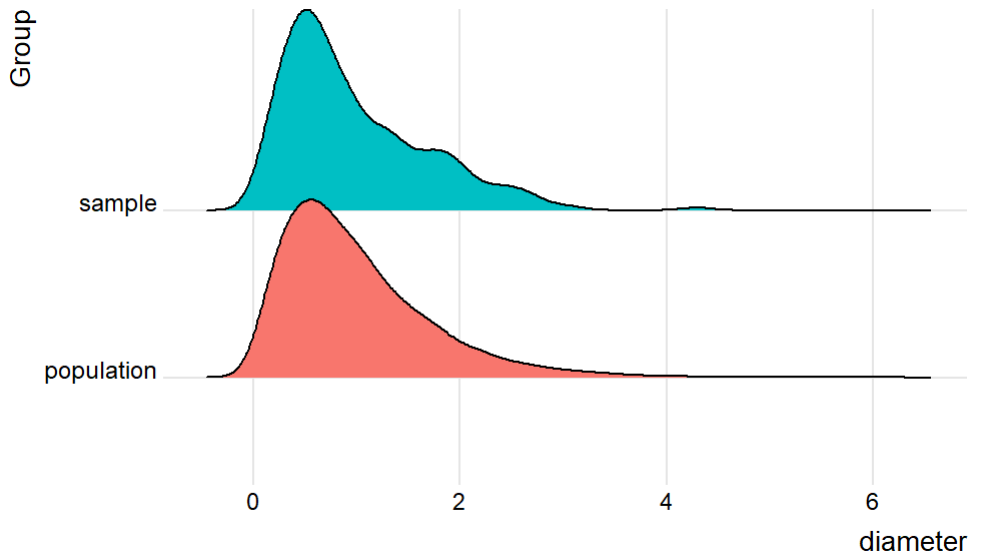
```
df |> group_by(Group) |>
  summarise(mean = mean(diameter), sd = sd(diameter)) |>
  knitr::kable(digit = 4, caption = "summary statistics")
```

Table: summary statistics

Group	mean	sd
population	0.9986	0.7174

We can also compare the distribution of sample and population using Ridge line chart. The ridge line chart requires a tidy dataset with a column indicating the group. Remember we have already get df as a tidy dataset. From the result, the sample distribution is very similar to the population.

```
library(ggridges)
df |> ggplot(aes(x = diameter,y = Group, fill = Group)) +
  geom_density_ridges(scale = 1.2) +
  theme_ridges() +
  theme(legend.position = "none")
```



From the result we find that the sample mean, sd is quite close to the population. So we may wonder: *does that happen all the time, or was this just a lucky draw?*

This question can be answered by simulating a large number of samples to see whether the undistorted representation of the population is typical for this sampling design.

Here we use the code to:

- drawing 1000 samples of size 300;
- storing the sample mean from each sample;
- computing the average difference between the sample means and the population mean.

Here we use the for loop to repetitively simulate drawing samples and record the mean and sd for each simulation. In each iteration $n = 1, 2, \ldots, 1000$, we draw the sample and compute the mean and sd, and then store the result in the result_mean and result_sd vectors and make a dataframe.

```
n = 1000
#creating storing vectors for mean and sd
result_mean = rep(0,n)
result_sd = rep(0,n)
set.seed(2023) # set random seed
for(i in 1:n){ # for loop for sampling and calculation
  sample = population |> sample_n(size = 250, replace = FALSE)
  result_mean[i] = mean(sample$diameter)
  result_sd[i] = sd(sample$diameter)
}
# construct the result dataframe
result = data.frame(mean = result_mean, sd = result_sd,
                    index = 1:1000)
```

Now, the result_mean and result_sd contains 1000 observation from 1000 simulations. We can compare this sampling distribution with our original mean and sd.

Combining 2 ggplots

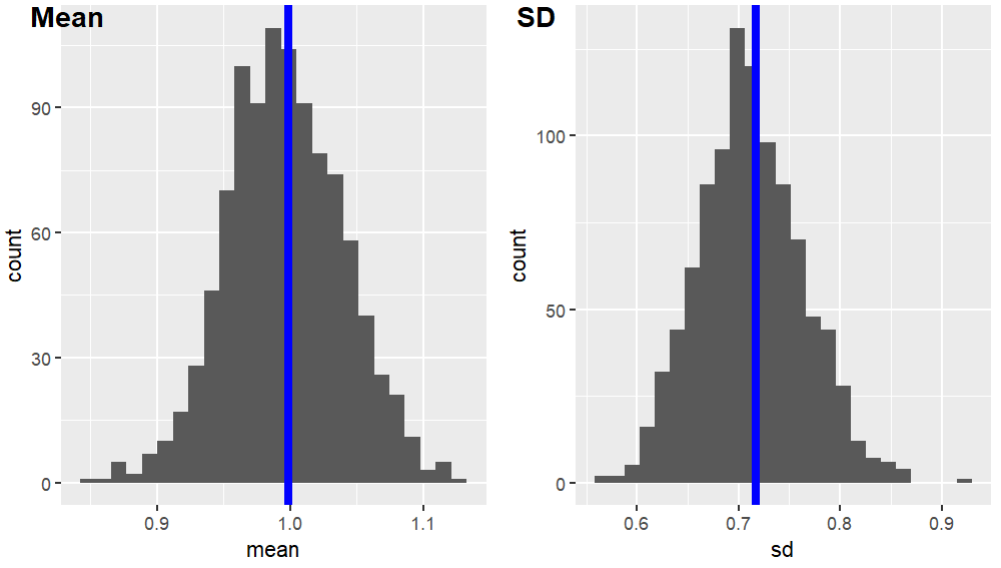
To check the sampling distribution of mean and sd, we can make 2 histogram and combine them together. To do so, we need the `ggpubr` package, `ggarange` function

```
library(ggpubr)
p1 = result |> ggplot(aes(x = mean)) +
  geom_histogram(bins = 25) +
  geom_vline(aes(xintercept = mean(population$diameter)), size=2, col

p2 = result |> ggplot(aes(x = sd)) +
  geom_histogram(bins = 25) +
  geom_vline(aes(xintercept = sd(population$diameter)), size=2, color

ggarrange(p1, p2,
          labels = c("Mean", "SD"),
          ncol = 2, nrow = 1)
```

You can arrange as much ggplots as you want in ggarange function. Just remember that the labels vector length is same with picture numbers, and ncol*nrow = that number.



The blue line indicates the true population average and population standard deviation. From these histograms we can observe the sampling distribution of mean and standard deviation is approximately normal with the mean equal to population average and standard deviation.

> Congulatulations, we have successifully simulated the **Central limit theorem**

Scenario 2: biased sampling

In the initial design, you were asked to imagine that you collected and sifted plant material to obtain seeds. Suppose you chose to use a sieve that is a little too coarse, tending only to sift out larger seeds and allowing smaller seeds to pass through because you were unaware that the typical seed has a diameter of about 1mm.

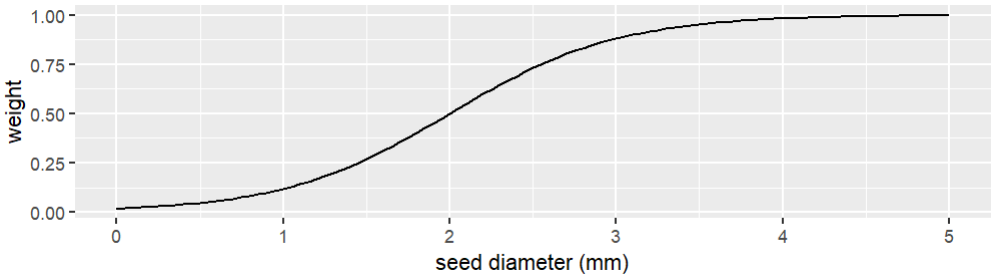
As a result, small seeds have a lower probability of being included in the sample and large seeds have a higher probability of being included in the sample.

Suppose the sampling weight (higher means higher probability) of the seed is defined as $\frac{1}{1+e^{-2(x-c)}}$, where $c = 2$.

Plotting distribution/function

To plot a given function, we can use `geom_function`. Here we use a shorthand `function(x) 1/(1 + exp(-2*(x-2) ))` to define a function that transform x into $\frac{1}{1+e^{-2(x-c)}}$. The syntax is: function(x) expression including x

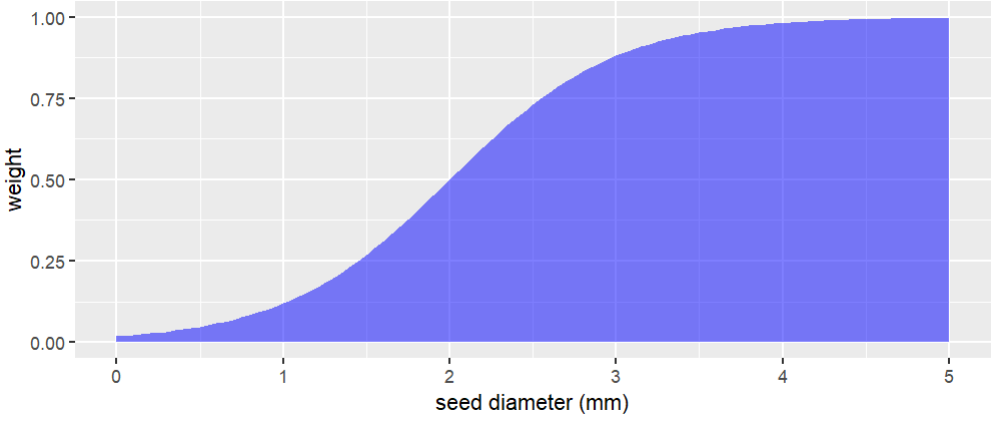
```
# To plot functions without data, specify range of x-axis
base = ggplot() + xlim(0, 5)
base + geom_function(fun =
    function(x) 1/(1 + exp(-2*(x-2) )) ) +
  ylab("weight") + xlab("seed diameter (mm)")
```



So from this plot larger diameters have larger weights and are more likely to be sampled.

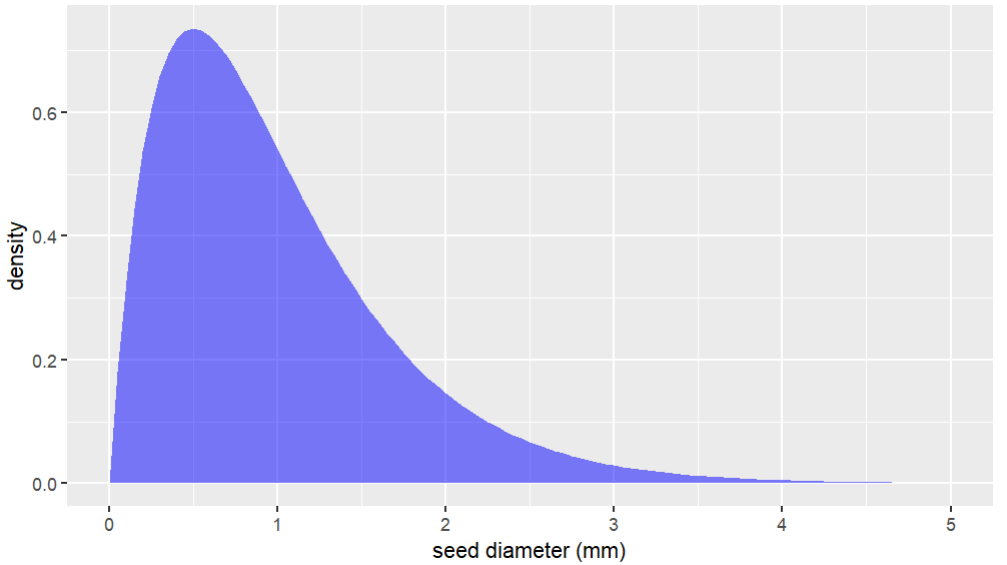
We can use this setting to make the area below the function to be filled with color:

```
# To plot functions without data, specify range of x-axis
base = ggplot() + xlim(0, 5)
base + stat_function(fun =
    function(x) 1/(1 + exp(-2*(x-2) )) ,
    geom = "area", fill = "blue", alpha = 0.5) +
  ylab("weight") + xlab("seed diameter (mm)")
```



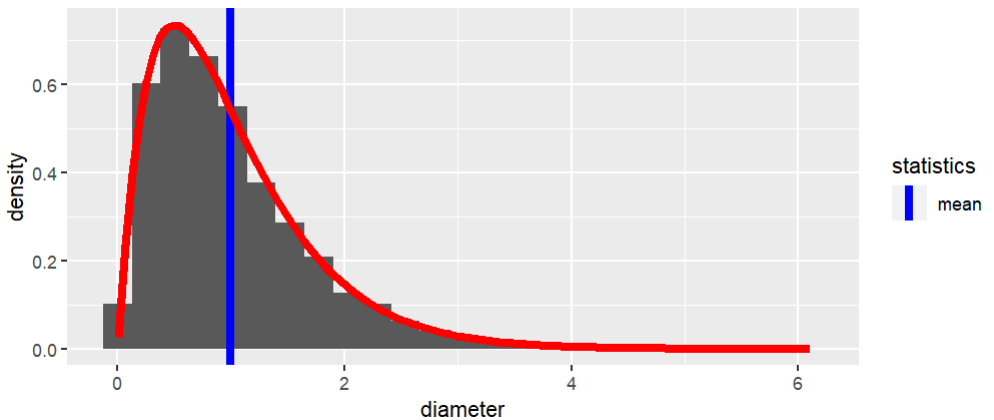
We can also use this to draw different functions, for example, the gamma distribution function with rate and shape = 2 (Our seed diameter population).

```
base = ggplot() + xlim(0, 5)
base + stat_function(fun =
    function(x) dgamma(x, shape = 2, rate = 2),
    geom = "area", fill = "blue", alpha = 0.5) +
  ylab("density") + xlab("seed diameter (mm)")
```



We can also add this layer on our population histogram:

```
#histogram for population
population |>
  ggplot(aes(x = diameter)) +
  # We need to add aes(y = ..density..) to make sure the histogram
  # and probability density function are in the same scale
  geom_histogram(aes(y = ..density..),bins = 25) +
  geom_vline(aes(xintercept = mean(population$diameter),
                 color = "mean"), size=2) +
  scale_color_manual(name = "statistics", values = c(mean = "blue"))
  # Now we add the density curve
  geom_function(fun =
    function(x) dgamma(x, shape = 2, rate = 2),
    size = 2, col = "red")
```



> Now back to the biased sampling:

The actual probability that a seed is included in the sample -- its **inclusion probability** -- is proportional to the sampling weight. These inclusion probabilities π_i can be calculated by normalizing the weights w_i:

$$\pi_i = \frac{w_i}{\sum_i w_i}$$

The following code draws a sample with replacement from the hypothetical seed population with seeds weighted according to the inclusion probability defined before as:

$\frac{1}{1+e^{-2(x-c)}}$, where $c = 2$

```
set.seed(2023)
weighted_sample = population |>
  #calculate the weight and add a new column to the population
  mutate(weight = 1/(1 + exp(-2*(diameter-2))) ) |>
  #sample 250 with replacement by the weight column
  sample_n(size = 250, replace = TRUE, weight = weight)
```

Now simulate it 1000 times:

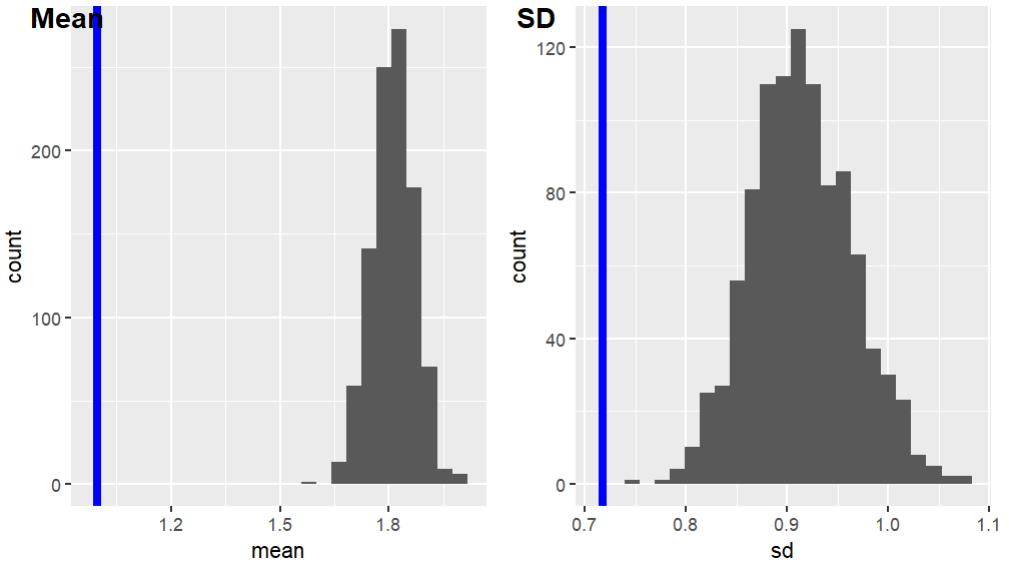
```
n = 1000
#creating storing vectors for mean and sd
result_mean = rep(0,n)
result_sd = rep(0,n)
set.seed(2023) # set random seed
for(i in 1:n){ # for loop for sampling and calculation
  weighted_sample = population |>
  mutate(weight = 1/(1 + exp(-2*(diameter-2))) ) |>
  sample_n(size = 250, replace = TRUE, weight = weight)
  result_mean[i] = mean(weighted_sample$diameter)
  result_sd[i] = sd(weighted_sample$diameter)
}
result_2 = data.frame(mean = result_mean, sd = result_sd,
                      index = 1:1000)
```

And then we draw the distribution histogram again:

```r
library(ggpubr)
p1 = result_2 |> ggplot(aes(x = mean)) +
  geom_histogram(bins = 25) +
  geom_vline(aes(xintercept = mean(population$diameter)), size=2, col

p2 = result_2 |> ggplot(aes(x = sd)) +
  geom_histogram(bins = 25) +
  geom_vline(aes(xintercept = sd(population$diameter)), size=2, color

ggarrange(p1, p2,
          labels = c("Mean", "SD"),
          ncol = 2, nrow = 1)
```



Put all 4 pictures in one plot:

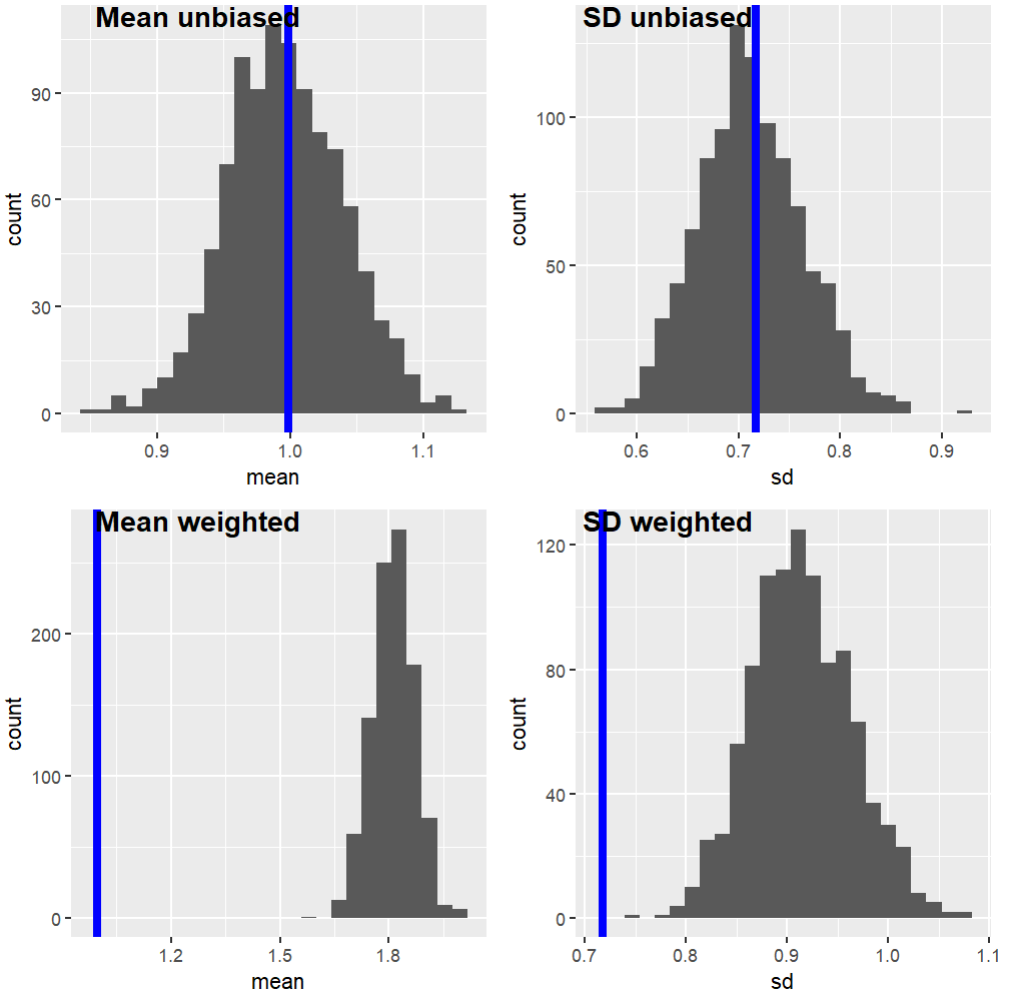
```
library(ggpubr)
p1 = result |> ggplot(aes(x = mean)) +
  geom_histogram(bins = 25) +
  geom_vline(aes(xintercept = mean(population$diameter)), size=2, col

p2 = result  |> ggplot(aes(x = sd)) +
  geom_histogram(bins = 25) +
  geom_vline(aes(xintercept = sd(population$diameter)), size=2, color

p3 = result_2 |> ggplot(aes(x = mean)) +
  geom_histogram(bins = 25) +
  geom_vline(aes(xintercept = mean(population$diameter)), size=2, col

p4 = result_2  |> ggplot(aes(x = sd)) +
  geom_histogram(bins = 25) +
  geom_vline(aes(xintercept = sd(population$diameter)), size=2, color

ggarrange(p1, p2, p3, p4,
          labels = c("Mean unbiased", "SD unbiased",
                     "Mean weighted", "SD weighted"),
          ncol = 2, nrow = 2)
```



Apparently, the weighted sampling introduce singnificant bias between sampling distribution and population mean and sd.

Sampling from 2 groups

Suppose you're interested in determining the average beak-to-tail length of red-tailed hawks to help differentiate them from other hawks by sight at a distance. Females and males differ slightly in length -- females are generally larger than males. Here we generate the length for a hypothetical population of 3000 females and 2000 males. The remaining sampling problems will in the homework.

```
set.seed(2023)
male = data.frame(
  length = rnorm(n=2000, mean = 57.5, sd = 3),
  sex = rep("Male", 2000)
)
female = data.frame(
  length = rnorm(n=3000, mean = 50.5, sd = 2.8),
  sex = rep("Female", 3000)
)
# because we already have the sex indicator column,
# bind_rows don't need the .id argument.
population_2 = bind_rows(male,female)
```

stacked histogram

If we choose aesthetic mapping of fill in geom_histogram, then we will construct stacked histogram.

```
population_2  |>
  ggplot( aes(x=length, fill=sex)) +
  geom_histogram(bin = 25,color="#e9ecef", alpha=0.6, position = 'ide
  scale_fill_manual(values=c("#69b3a2", "#404080")) +
  labs(fill="")
```