Simulating survey studies

Carlo Knotz

Table of contents

1 Introduction

This tutorial shows how you can simulate many, many different iterations of a fictional survey study – and how the different results are, collectively, distributed. In other words, it shows you the main idea behind the *Central Limit Theorem* and how it is used in applied social research to learn about a population based on small random samples.

Important: Some of these simulations take a few minutes to run. Be patient.

1.1 Setup

These are the packages you'll need (use `install.packages()` to install anything that is missing, which is likely `ggpubr`):

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   4.0.0     v tibble    3.2.1
v lubridate 1.9.4     v tidyr     1.3.1
v purrr     1.2.0
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becor
```

```
theme_set(theme_classic())
library(ggpubr)
```

1.2 Simulating our fictional population

Imagine that we want to figure out the average level of happiness in a country similar to Norway, with a population of 5.5 million people. We measure happiness on a 0-10 scale.

In R, we can create or simulate a such a population of 5.5 million individuals and their levels of happiness with a few lines of code, as shown below. The main tool we use here is the `sample()` function, which – as the same suggests – samples randomly from a pre-specified set of numbers.[1] In our case, we want to sample from the numbers 0 to 10, which we specify as `seq(0,10,1)` ("from 0 to 10, in steps of 1"), we want to do that 5.5 million times (`size = 5.5*10^6`), and we want the different numbers to have certain probabilities to be chosen (which all need to add up to 1, or 100%).

We run the `sample()` function within a function to create a `data.frame` and save the result as the variable `happy`. We also create another variable called `idno`, which is just a unique ID for each of the 5.5 million "individuals". The resulting `data.frame` – our "population" – gets saved as `pop`.

Note that we also set a "seed" number so that we can always exactly reproduce the same results later; otherwise there will be small random variations in the results. You always need to run the `set.seed()` function together with the code immediately below for results to stay the same every time you run the analysis:

```
set.seed(42)
pop <- data.frame(idno = seq(1,5.5*10^6,1),
                  happy = sample(seq(0,10,1),
                                 size = 5.5*10^6,
                                 replace = T,
                                 prob = c(0.01,.01,.02,.03,.04,.09,.10,.13,.20,.29,0.05)))
```

The "population" will now appear as `pop` in your *Environment* tab.

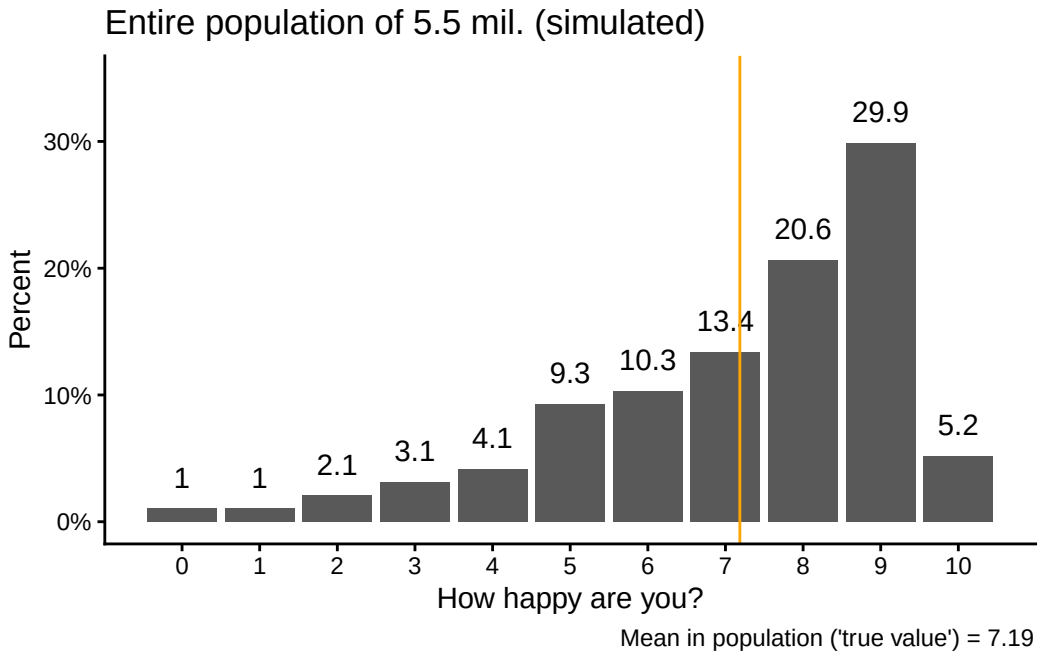
We can then calculate the "true" average level of happiness in our population. This is the "target" value we are after. In a real social scientific study, this is the value that we do not know (because we very rarely have data for an entire population) but want to measure with a social survey:

```
popmean <- mean(pop$happy)
```

We can also visualize the distribution of happiness in the population plus the average value as a vertical line:

[1] See also `?sample` for details.

```r
pop |>
  group_by(happy) |>
  summarize(obs = n()) |>
  mutate(share = obs/sum(obs)) |>
  ggplot(aes(x = happy, y = share)) +
  geom_bar(stat = "identity") +
  geom_text(aes(label = round(100*share, digits = 1)), vjust = -1) +
  geom_vline(xintercept = popmean, color = "orange") +
  scale_x_continuous(breaks = seq(0,10,1)) +
  scale_y_continuous(labels = scales::percent,
                     limits = c(0,.35)) +
  labs(y = "Percent", x = "How happy are you?",
       title = "Entire population of 5.5 mil. (simulated)",
       caption = paste0("Mean in population ('true value') = ",round(popmean, digits = 2))) +

popvis
```



1.3 Taking a first sample

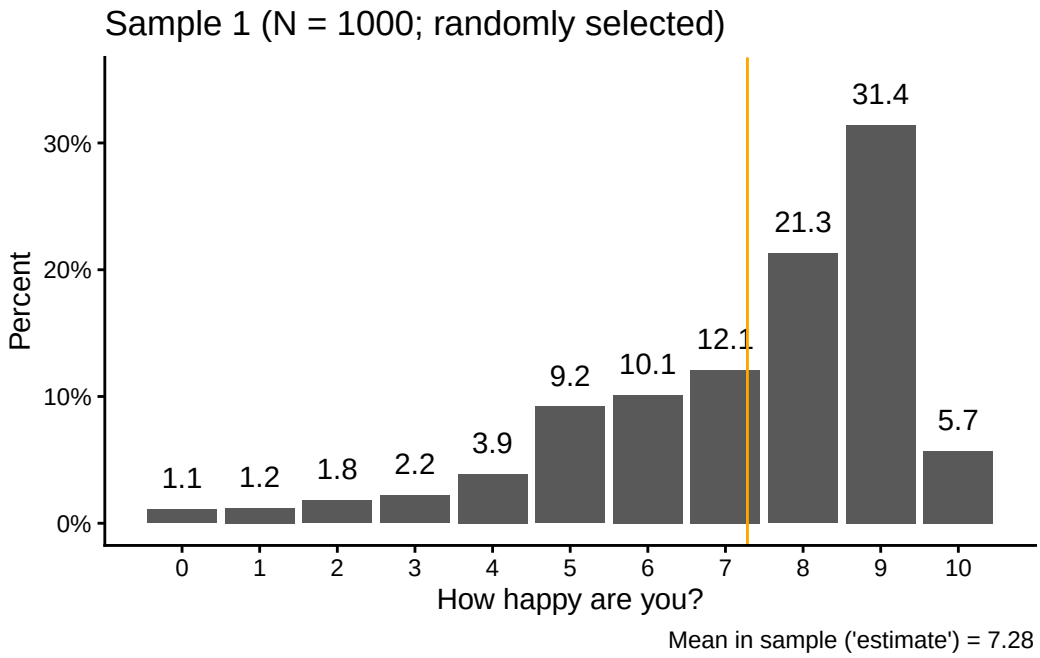
Now we pretend that we are doing a social science survey by taking a random sample of 1000 "respondents" from the population with `slice_sample()`. Here again, we set a seed number so that we get the same results every time.

```
set.seed(42)
sam1 <- pop |>
  slice_sample(n = 1000)
```

If you take a look at your *Environment* tab, you see the sample data as `sam1`: 1000 random picks from the "population" (`pop`) and their levels of happiness.

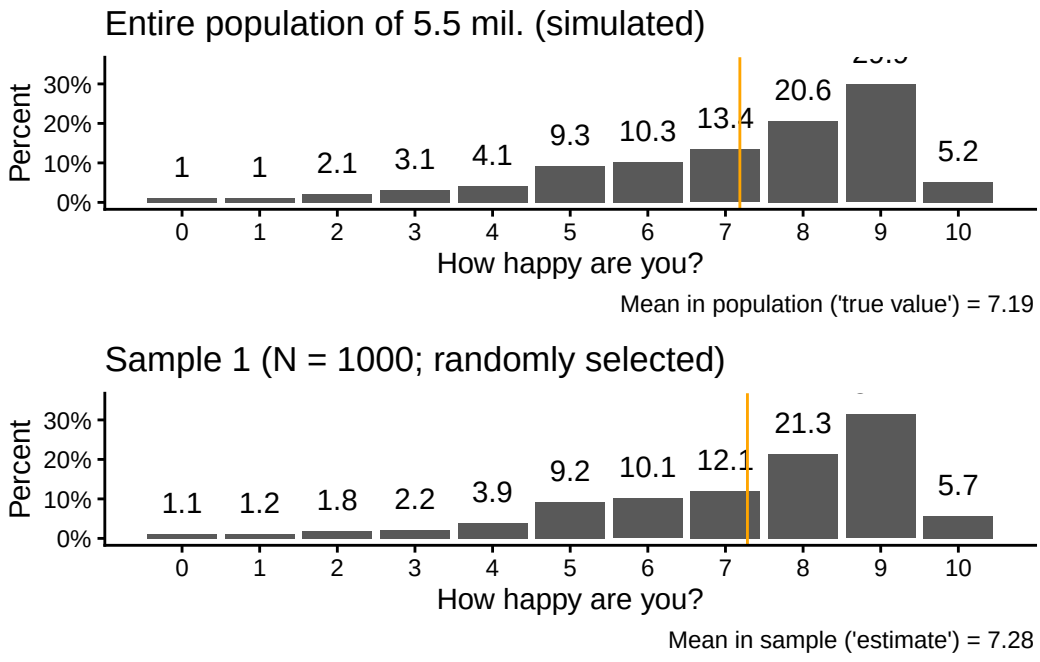
Then we also calculate the average level of happiness in the sample and visualize the sample data:

```
sam1mean <- mean(sam1$happy)

sam1 |>
  group_by(happy) |>
  summarize(obs = n()) |>
  mutate(share = obs/sum(obs)) |>
  ggplot(aes(x = happy, y = share)) +
  geom_bar(stat = "identity") +
  geom_text(aes(label = round(100*share, digits = 1)), vjust = -1) +
  geom_vline(xintercept = sam1mean, color = "orange") +
  scale_x_continuous(breaks = seq(0,10,1)) +
  scale_y_continuous(labels = scales::percent,
                     limits = c(0,.35)) +
  labs(y = "Percent", x = "How happy are you?",
       title = "Sample 1 (N = 1000; randomly selected)",
       caption = paste0("Mean in sample ('estimate') = ",round(sam1mean, digits = 2))) -> sa

sam1vis
```



We can also use `ggarrange()` from the `ggpubr` package to directly compare the sample and population data:

```
ggpubr::ggarrange(popvis,sam1vis, nrow = 2)
```



If you take a careful look, you see that the overall pattern is the same, but there are small variations in the percentages and in the average value.

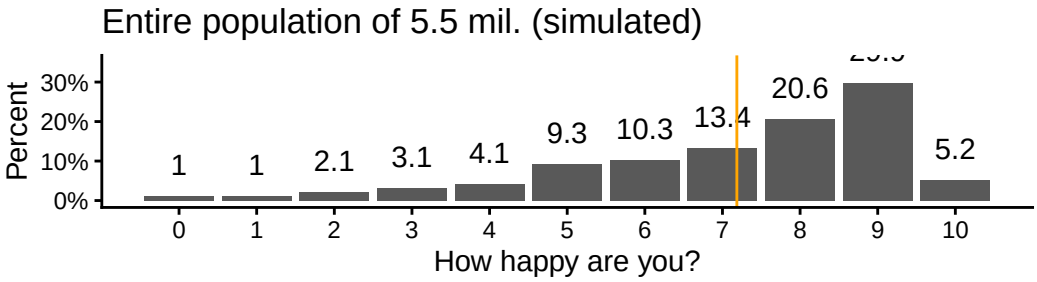
1.4 Taking a second sample

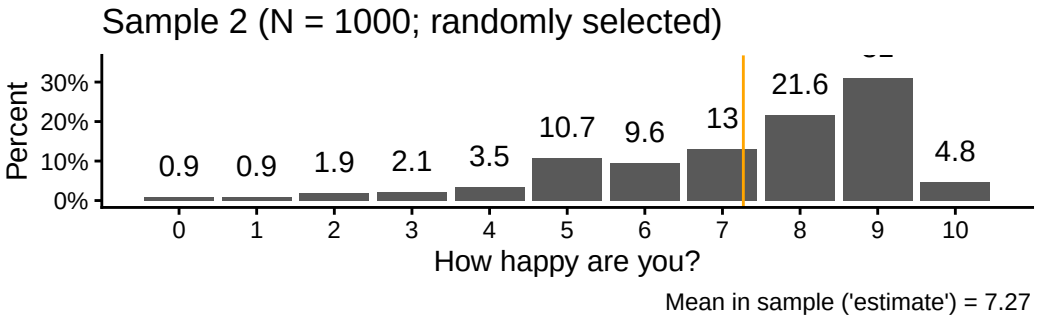
Let's now take a second sample. Here, we specicy a different seed number so that we don't get the same result again (but so that the entire analysis is still reproducible):

```
set.seed(17)
sam2 <- pop |>
  slice_sample(n = 1000)
```

Then we again calculate the sample mean in the second sample and visualize the new sample data next to the original population data:

```
sam2mean <- mean(sam2$happy)

sam2 |>
  group_by(happy) |>
  summarize(obs = n()) |>
  mutate(share = obs/sum(obs)) |>
  ggplot(aes(x = happy, y = share)) +
  geom_bar(stat = "identity") +
  geom_text(aes(label = round(100*share, digits = 1)), vjust = -1) +
  geom_vline(xintercept = sam2mean, color = "orange") +
  scale_x_continuous(breaks = seq(0,10,1)) +
  scale_y_continuous(labels = scales::percent,
                     limits = c(0,.35)) +
  labs(y = "Percent", x = "How happy are you?",
       title = "Sample 2 (N = 1000; randomly selected)",
       caption = paste0("Mean in sample ('estimate') = ",round(sam2mean, digits = 2))) -> sa

ggpubr::ggarrange(popvis,sam2vis, nrow = 2)
```



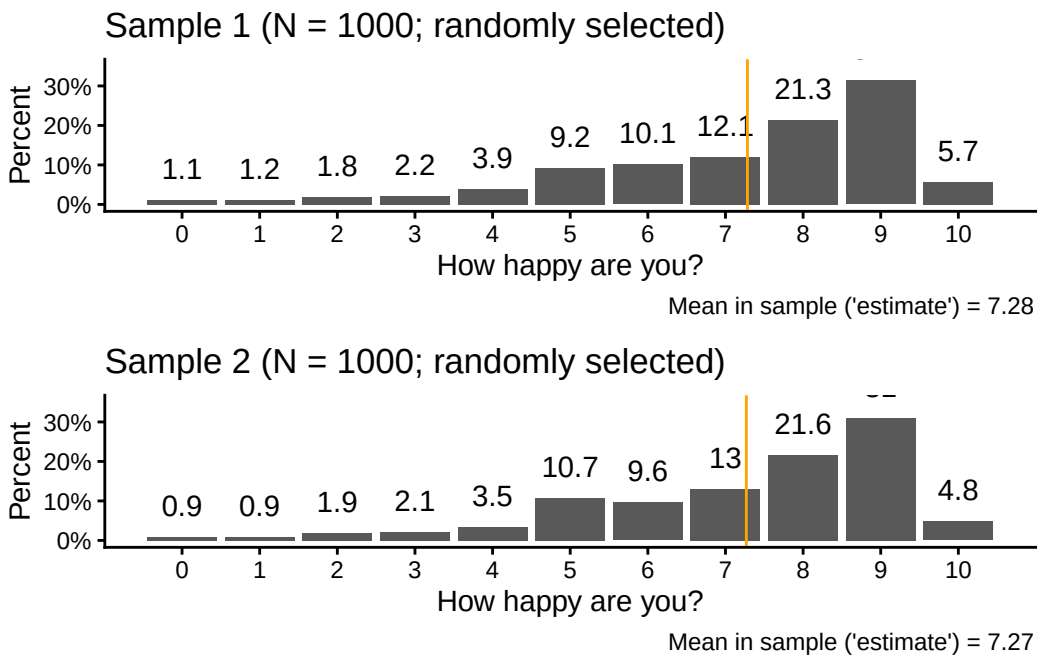


You should see again that there are small differences between the sample and the population, but the overall patterns are very similar.

1.5 Comparing the samples

You can see these small differences also when we compare the two samples directly with each other:

```
ggpubr::ggarrange(sam1vis,sam2vis, nrow = 2)
```



1.6 Scaling it up to 1000 samples

So far, we took only two random samples with 1000 observations ("respondents") per sample. In a real-life scenario, this is about as much as we could realistically do.

Within this simulation, however, we can use R to take many, many samples from our "population" (almost) simultaneously and then see how the main statistic we are interested in, the sample average, varies between the different samples.

To do this, we first create a `data.frame` in which we later save the different simulation results: **a dataset of results**. We call this `sampdist`:

```
sampdist <- data.frame(sample = seq(1,1000,1),
                       result = c(c(sam1mean,sam2mean),rep(NA,998)))
```

This dataset includes two variables, one is `sample` which simply tells us the number of each sample from 1 to 1000. The other variable (called `result`) stores the average happiness value from each sample. To create the `results` variable, we start with the first two sample means and then add 998 `NA` values to get a total of 1000.

Next, we use a **loop** to simulate 1000 samples. A loop is a fundamental concept in programming (also outside of R). It is, very simply put, an instruction to the computer to repeat a certain operation a certain number of times, one after the other.

In R, you can specify loops with the `for()` function, as shown below. What we are tell R with this code is, translated to human language:

1. For all values of `k`, which we define to be the numbers between 3 and 1000…
2. …go the `pop` dataset, draw a random sample from it, and store that sample as `loopsam`…
3. …calculate the mean of the `happy` variable in that sample…
4. …and then write the result, the mean of sample `k`, into the `k`-th row and the second column of our results dataset.

Notice that we put the code after the `for()` function within curly braces (`{}`) to show R that this all fits together and belows to the loop. We start at 3 because we already have values for the first two rows of the results dataset.

Be aware: Loops are not very efficient and this calculation can take a few moments.

```
for(k in 3:1000) {

  loopsam <- pop |>
    slice_sample(n = 1000)

  sampdist[k,2] <- mean(loopsam$happy)
}
```

Let's have a look at the first few observations in our results dataset:

```
head(sampdist)
```

```
  sample result
1      1  7.283
2      2  7.269
3      3  7.158
4      4  7.251
5      5  7.348
6      6  7.148
```

What you see here is the first 6 samples and their sample means. The first two are the same as above, and the other four were generated by the loop. You also see, and this is important, that the sample means vary *slightly*, but they are typically somewhere around 7.2.

Let's now calculate the average over all of these averages – the "average result" – and compare that to the "true" population mean from earlier:

```
sampdistmean <- mean(sampdist$result)
sampdistmean
```

```
[1] 7.182107
```

```
popmean
```

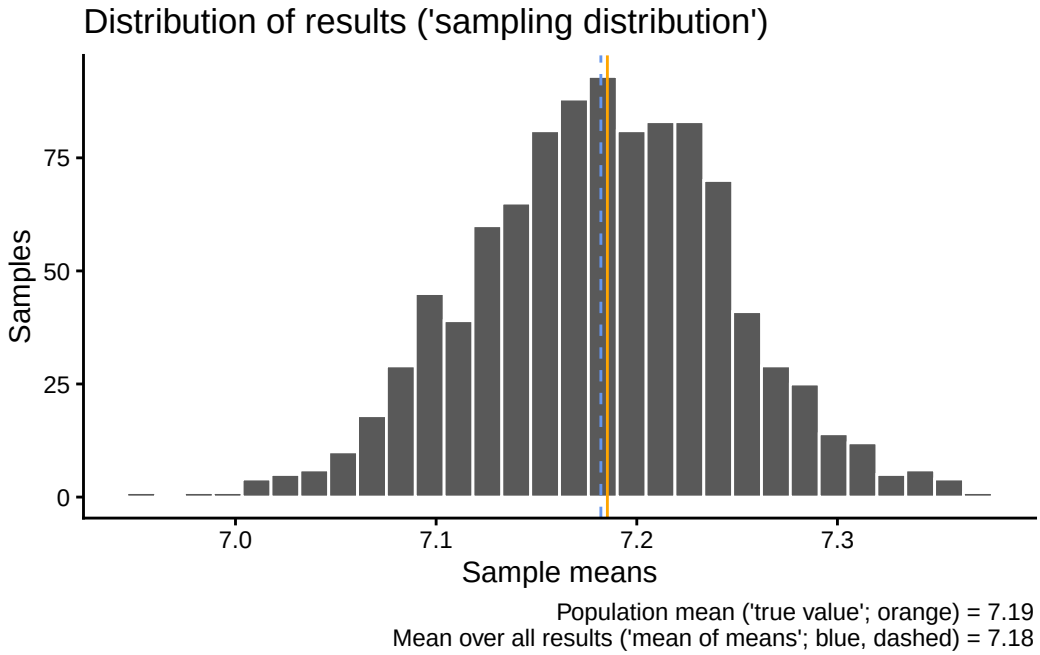
```
[1] 7.185346
```

The average result – the "mean of all means" – is 7.18.

Then we can also visualize the distribution of all our different test results together with the "true" population mean from earlier and the average result:

```
sampdist |>
  ggplot(aes(x = result)) +
  geom_histogram(color = "white") +
  geom_vline(xintercept = sampdistmean, color = "cornflowerblue", linetype = "dashed") +
  geom_vline(xintercept = popmean, color = "orange") +
  labs(x = "Sample means", y = "Samples",
       title = "Distribution of results ('sampling distribution')",
       caption = paste0("Population mean ('true value'; orange) = ",round(popmean, digits = 2
                        "\n Mean over all results ('mean of means'; blue, dashed) = ",round(s
```

```
`stat_bin()` using `bins = 30`. Pick better value `binwidth`.
```



Do you notice something here? What does this distribution remind you of, and why do you think do we get this shape (especially since the original distribution does not at all look like that? If you can answer this question, you have understood the main idea behind the *Central Limit Theorem*).

1.7 Then we do 10'000 surveys

1000 simulated results are nice, but we *can* also go higher, let's say to 10'000. To do that, we create a new results `data.frame` (which we save under the same name as before), but now with 10'000 rows instead of 1000:

```
set.seed(42)
sampdist <- data.frame(sample = seq(1,10000,1),
                       result = c(c(sam1mean,sam2mean),rep(NA,9998)))
```

Once we have that, we again run a loop, but from numbers 3 to 10'000. Caution, this now takes a bit longer still:

```
for(k in 3:10000) {

  loopsam <- pop |>
    slice_sample(n = 1000)

  sampdist[k,2] <- mean(loopsam$happy)
}
```

Let's have a look at new "mean of means" of the 10'000 new samples and compare it to the overall population mean:

```
sampdistmean <- mean(sampdist$result)
sampdistmean
```

```
[1] 7.185184
```

```
popmean
```

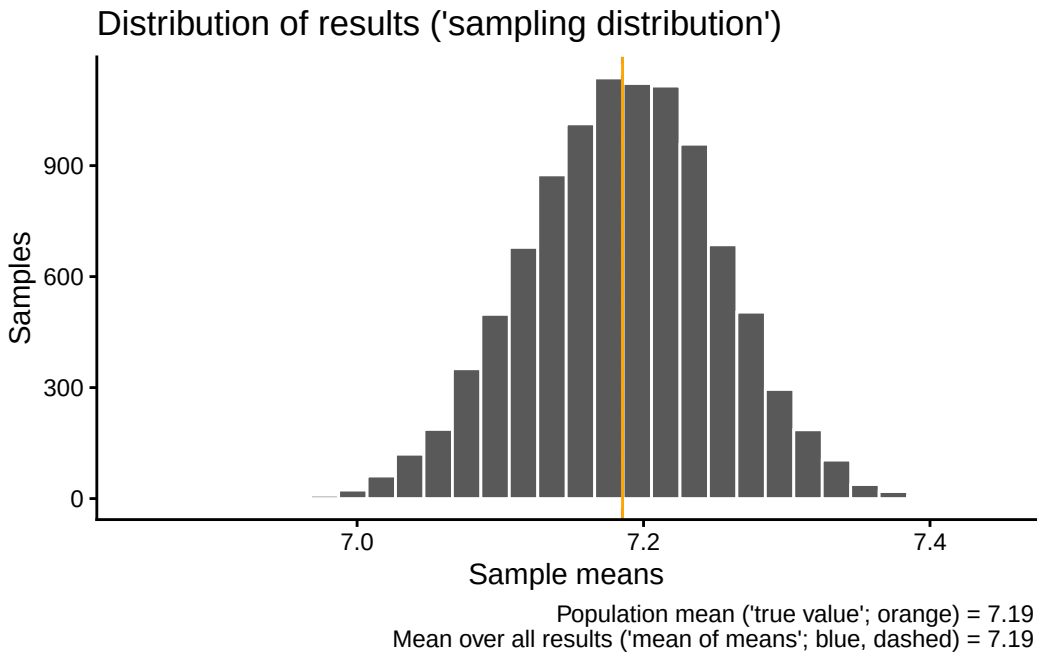
```
[1] 7.185346
```

Notice how they are very close to each other, but still not exactly identical?

Then we also visualize the new distribution of all 10'000 sample means:

```
sampdist |>
  ggplot(aes(x = result)) +
  geom_histogram(color = "white") +
  geom_vline(xintercept = sampdistmean, color = "cornflowerblue", linetype = "dashed") +
  geom_vline(xintercept = popmean, color = "orange") +
  labs(x = "Sample means", y = "Samples",
       title = "Distribution of results ('sampling distribution')",
       caption = paste0("Population mean ('true value'; orange) = ",round(popmean, digits = 2
                        "\n Mean over all results ('mean of means'; blue, dashed) = ",round(s
```

```
`stat_bin()` using `bins = 30`. Pick better value `binwidth`.
```



Do you notice what changed compared to the 1000 results earlier? Why would that be?

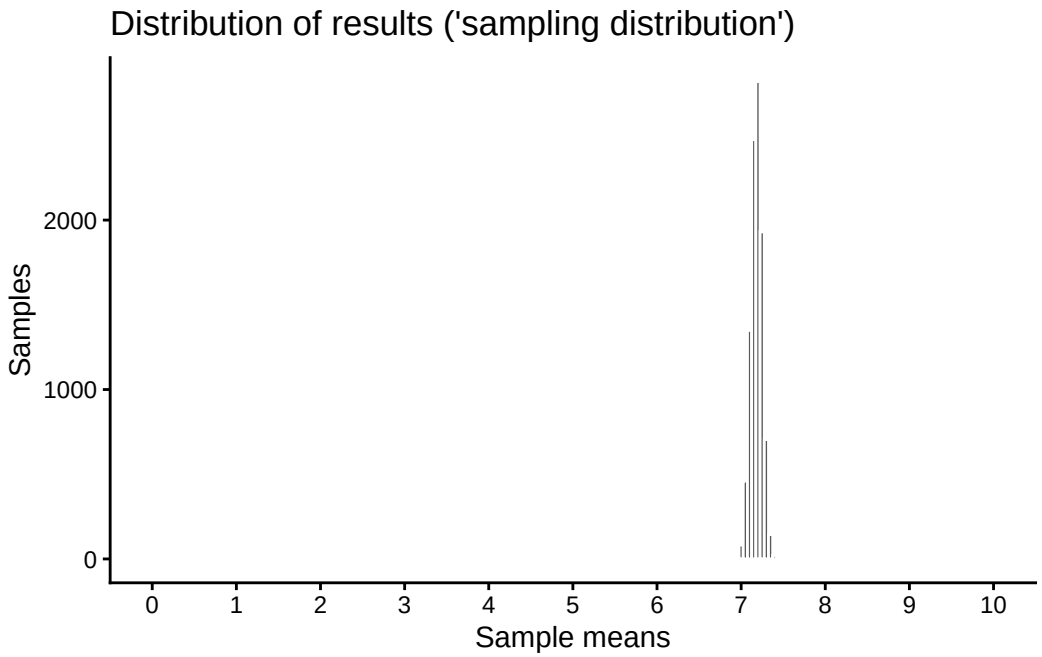
1.8 The results in context

You might now get the impression that there is quite a bit of variation between the different samples, that the result "jumps" quite a bit between different repetitions of our fictional survey. However, notice also that the scale of the above graph only covers a small part of the entire scale of our original happiness variable, which goes from 0 to 10.

To show the "sampling variation", the random differences between our 10'000 samples, in context, we expand the scale of our x-axis to the true scale of the original variable with the `limits` option in the code below:

```
sampdist |>
  ggplot(aes(x = result)) +
  geom_histogram(color = "white", binwidth = 0.05) +
  scale_x_continuous(limits = c(0,10),
                     breaks = seq(0,10,1)) +
  labs(x = "Sample means", y = "Samples",
       title = "Distribution of results ('sampling distribution')")
```

```
Warning: Removed 2 rows containing missing values or values outside the scale range
(`geom_bar()`).
```

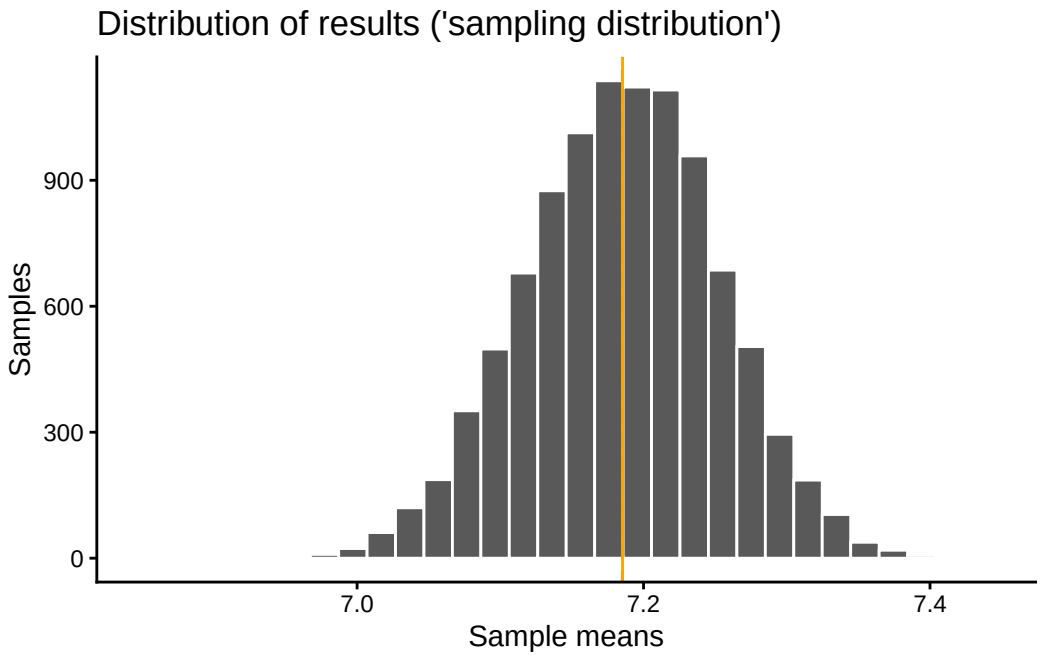


1.9 Comparing our results distribution ("sampling distribution") to the Normal distribution

You might (should) have already noticed what how our different "survey" results are distributed, but just to make sure: Here is a comparison of our sampling distribution to the normal distribution:

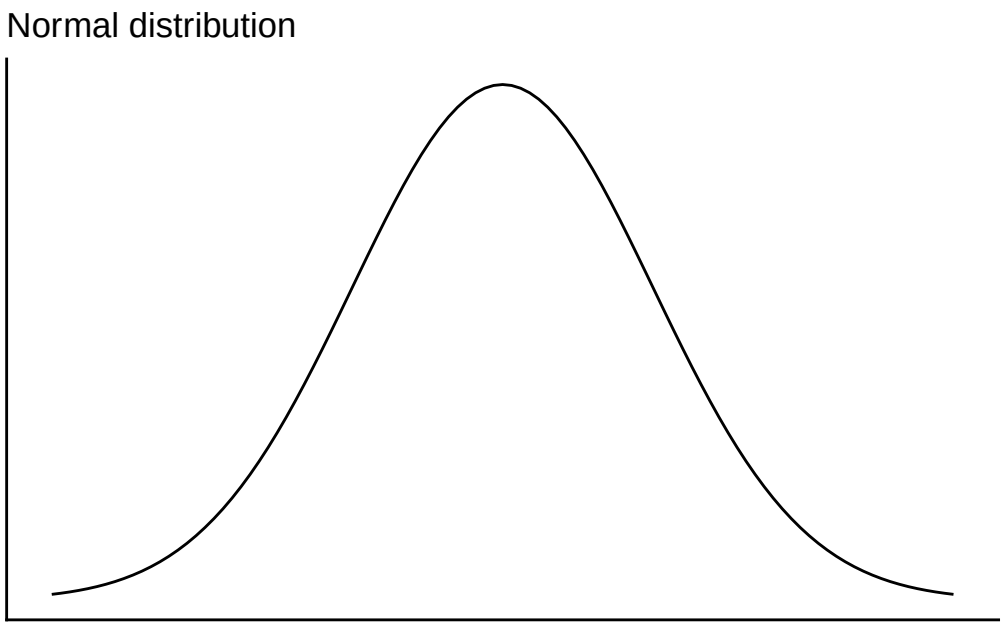
```
sampdist |>
  ggplot(aes(x = result)) +
  geom_histogram(color = "white") +
  geom_vline(xintercept = sampdistmean, color = "cornflowerblue", linetype = "dashed") +
  geom_vline(xintercept = popmean, color = "orange") +
  labs(x = "Sample means", y = "Samples",
       title = "Distribution of results ('sampling distribution')") -> samdist10k
samdist10k
```

```
`stat_bin()` using `bins = 30`. Pick better value `binwidth`.
```



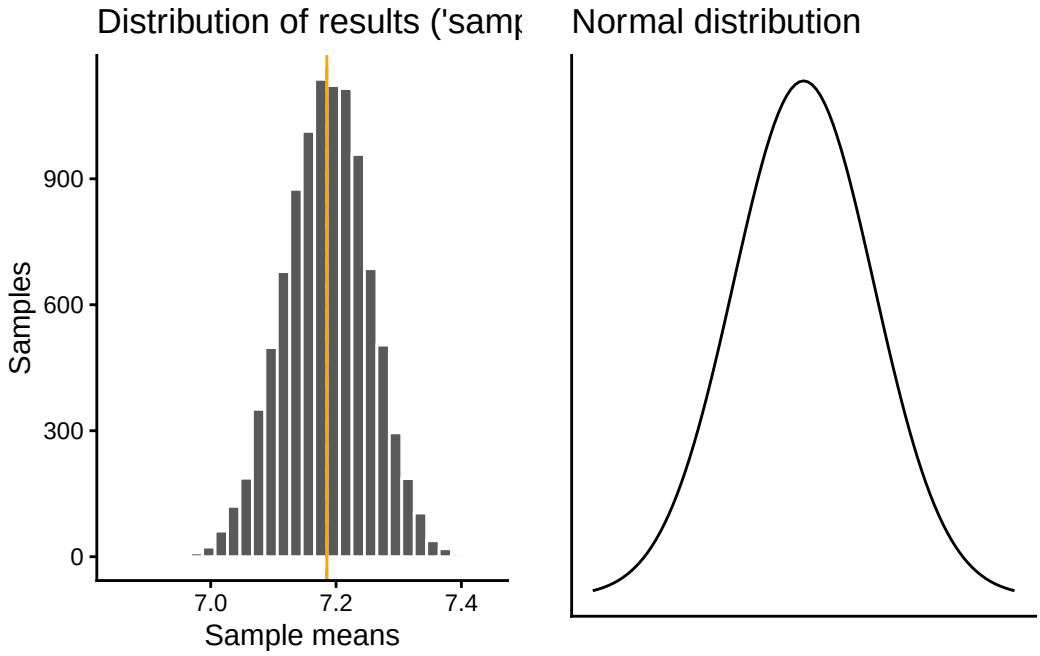

```
p1 <- ggplot(data = data.frame(x = c(-3, 3)), aes(x)) +
  stat_function(fun = dnorm, n = 101, args = list(mean = 0, sd = 1)) + ylab("") +
  scale_y_continuous(breaks = NULL) +
  scale_x_continuous(breaks = NULL) +
```

```
  labs(x = "", title = "Normal distribution")
p1
```




```
ggpubr::ggarrange(samdist10k,p1, ncol = 2)
```

```
`stat_bin()` using `bins = 30`. Pick better value `binwidth`.
```


Distribution of results ('samp
Samples
900
600
300
0
7.0
7.2
7.4
Sample means
Normal distribution

1.10 Now 10'000 new samples, but with a smaller sample size (200 instead of 1'000)

In the previous simulations, we always worked with a sample size of 1000. First, we took 1000 samples of 1000 "respondents" each, then we took 10'000 samples of 1000 respondents.

Let's see what happens when we take 10'000 new samples, but now with a size of 200 "respondents" per sample:

```
sampdist_200 <- data.frame(sample = seq(1,10000,1),
                           result = rep(NA,10000))

set.seed(42)
for(k in 1:10000) {

  loopsam <- pop |>
    slice_sample(n = 200)

  sampdist_200[k,2] <- mean(loopsam$happy)
}

sampdist_200mean <- mean(sampdist_200$result)
sampdist_200mean
```

```
[1] 7.184244
```

```
popmean
```

```
[1] 7.185346
```

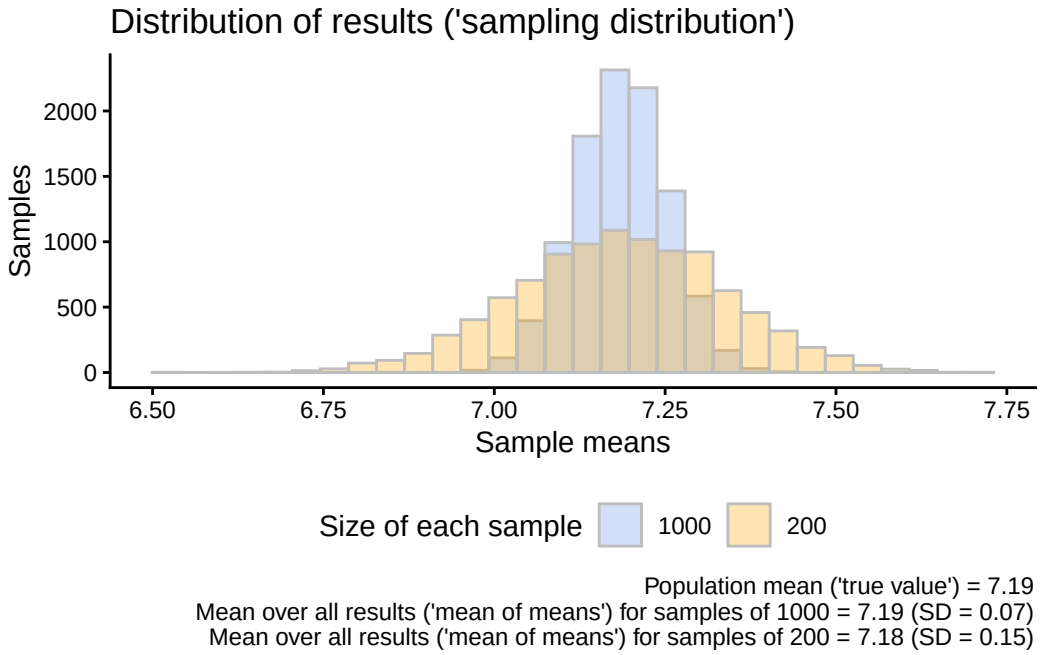
The new "mean of means", now based on 200 "respondents" per sample, is still very close to the "true" population mean.

To see what changes when we use smaller samples, we show both sampling distributions, the one for the samples with $N = 1000$ and the new one for the samples with $N = 200$, in the same graph. We also calculate the standard deviation of each of the distributions:

```
sampdist_SD <- sd(sampdist$result)
sampdist_200_SD <- sd(sampdist_200$result)

sampdist |>
  left_join(sampdist_200, by = "sample",
```

```
           suffix = c("1000","200")) |>
  pivot_longer(cols = -1,
               names_to = "size",
               values_to = "meanval") |>
  mutate(size = gsub("result","",size)) |>
  ggplot(aes(x = meanval,fill = size)) +
  geom_histogram(alpha=0.3, position="identity",
                 color = "grey") +
  scale_fill_manual(values = c("cornflowerblue","orange")) +
  labs(x = "Sample means", y = "Samples",
       fill = "Size of each sample",
       title = "Distribution of results ('sampling distribution')",
       caption = paste0("Population mean ('true value') = ",round(popmean, digits = 2),
                        "\n Mean over all results ('mean of means') for samples of 1000 = ",
                        round(sampdistmean, digits = 2)," (SD = ",round(sampdist_SD, digits =
                        "\n Mean over all results ('mean of means') for samples of 200 = ",
                        round(sampdist_200mean, digits = 2)," (SD = ",round(sampdist_200_SD,
  theme(legend.position = "bottom")
```

```
`stat_bin()` using `bins = 30`. Pick better value `binwidth`.
```



Do you notice where the main difference is between the sampling distribution based on $N = 1000$ and the one based on $N = 200$?

If you had to design a social survey to measure happiness (or something else) in the Norwegian population, how could you use the results of this simulation to guide your study design?