Handout 6 Video Script: Confidence Intervals for a Mean

Nov 6, 2026 · @Xinran

Chunk	Keep or blank	Note
`library()`, `data(ncbirths)`, `set.seed(123)`	Keep	Run on camera
Helper functions	Keep	Handout says "just run it"
Population + sample (n = 30)	Blank	Typed live in Part 3
CI parameters + `c(xbar - moe, xbar + moe)`	Blank	Typed live in Part 4
Capture check (`lower`, `upper`, `&`)	Blank	Typed live in Part 4
n = 100 sample + CI	Blank	Typed live in Part 5
`simulate_cis()` call	Blank	Typed live in Part 6
ggplot chunk	Blank, or keep pre-written	Keeping it saves about 3 minutes

Matching the handout numbers. `set.seed(123)` only runs once in my script, so every extra `sample()` call shifts all my later results. If I rerun a sampling line after a typo, I restart R and run all the chunks above. Or I just read the numbers on my screen instead of the ones in the script.

How I read this script. I say everything outside the code blocks as written, and I type the code blocks. When I see ___, I say the number on my screen.

Part 1 — Intro and learning goals

Hi everyone, welcome back. In this video, I'm going to walk you through Handout 6: Confidence Intervals for a Mean.

Here's the big question for today. In real life, we almost never get to see the whole population. We take one sample, we compute one sample mean, and then we want to say something about the population mean that we can't see.

A single number, like "the average is 7.4," is almost certainly not exactly right. So instead of giving one number, we give a range — an interval — and we attach a level of confidence to it. That's a confidence interval.

By the end of this video, you should be able to do two things. First, construct a confidence interval for a population mean in R. Second, use simulation to see what "95% confidence" actually means in the long run. That second idea is honestly the most important thing in this handout, so stick with me to the end.

This handout is practice only — there's nothing to submit. But I strongly recommend you open RStudio and type along with me. You'll learn this much faster with your hands on the keyboard.

Part 2 — Setup and helper functions

I'll start at the top, with the setup chunk. I'm loading two packages: tidyverse, which gives us tools for working with data and making plots, and openintro, which contains the dataset we'll use. Then `data(ncbirths)` loads a dataset of births in North Carolina.

Let me run this chunk. You'll see a bunch of messages and warnings pop up. Those are normal. They're just telling us which packages got loaded and which versions. Nothing is broken.

Next, I set the seed: `set.seed(123)`. This one matters. I'm about to draw random samples, and normally every random sample gives different numbers. Setting the seed makes the randomness reproducible — if you run the same code in the same order, you'll get exactly the same samples I get.

One caution. The seed fixes the whole sequence of random draws. If you run a sampling line twice, everything after it shifts. So if your numbers don't match mine, don't panic. Restart R and run the chunks from the top, in order.

Now I'll scroll down to the helper functions. The handout says you just need to run this chunk, and that's true. But I want to take thirty seconds to show you what's inside, because it'll make the rest of the video make sense.

The first function, `get_t_star`, gives us the critical value from the t-distribution. It uses `qt`, which is R's quantile function for the t-distribution. It asks for 1 minus alpha over 2. Why? If alpha is 0.05, we want 95% of the area in the middle. That leaves 2.5% in each tail. So we want the point that has 97.5% of the area to its left. That point is the t-star I'll multiply by.

The second function, `simulate_cis`, repeats the whole confidence-interval recipe many times — by default, 100 times. Each time, it draws a new sample, builds an interval, and records whether that interval caught the true population mean. Don't worry about the details yet. You'll notice later that it's exactly the same steps I'm about to write by hand.

Okay, let me run this chunk so both functions are ready to use.

Part 3 — Defining the population and sample

Okay, now comes the part I'm going to write myself. First, I'll set up the population.

Normally we can't see the population — that's the whole point. But for learning purposes, I'm going to pretend the ncbirths data is our entire population. That way, we know the true answer, and we can check whether our confidence intervals actually work.

I'll use the `weight` variable. That's each baby's birth weight, in pounds. Let me type this in.

```
#population
population_data <- ncbirths$weight
pop_mean <- mean(population_data, na.rm = TRUE)
pop_mean
```

`population_data` stores every birth weight. `pop_mean` is the population mean — in symbols, that's mu. And `na.rm = TRUE` just tells R to skip any missing values.

Let me run it. The true mean is 7.101 pounds. Remember that number. In real life, this is exactly the number we'd never get to see.

Now I'll take a sample of 30 babies.

```
#sample
n <- 30 #sample size
sample_data <- sample(population_data, n)
xbar <- mean(sample_data) #sample mean
xbar
s <- sd(sample_data) #sample standard deviation
s
```

`n` is my sample size. `sample()` randomly picks n values from the population. Then I compute the sample mean, x-bar, and the sample standard deviation, s.

Let me run that. My sample mean is about 7.47, and my sample standard deviation is about 1.02.

Notice: x-bar is 7.47, but the true mean is 7.10. My sample mean is off by more than a third of a pound. That's not a mistake. That's sampling variability. A different 30 babies would give a different x-bar. And that's exactly why one number isn't enough, and why we need an interval.

If your numbers are a little different from mine, that's fine — it just means your random sample is different. Everything I do next works exactly the same way.

Part 4 — One confidence interval

Before I write any code, let me show you the formula for a confidence interval for a mean.

```
\bar{x} \pm t^{*}_{\alpha/2} \cdot \frac{s}{\sqrt{n}}
```

Let me read it piece by piece. x-bar is the sample mean. That's the center of the interval — our best single guess. Then we go out some distance in both directions. That distance is the margin of error, and it's made of two parts.

The first part is t-star, the critical value. It controls how confident we want to be. More confidence means a bigger t-star, which means a wider interval.

The second part is the standard error: s over the square root of n. It measures how much x-bar typically bounces around from sample to sample. More spread in the data means a bigger s. A bigger sample means a bigger square root of n. So bigger samples give smaller standard errors.

Why the t-distribution and not the normal? Because we don't know the population standard deviation. We're estimating it with s, and that adds a bit of extra uncertainty. The t-distribution accounts for that, and its shape depends on the degrees of freedom, which is n minus 1.

Now I'll turn this formula into code.

```
#Confidence Interval parameters
alpha <- 0.05
df <- n - 1
t_star <- get_t_star(alpha, df)
se <- s / sqrt(n)
moe <- t_star * se
```

I set alpha to 0.05, which means I want a 95% confidence interval — 1 minus alpha. Degrees of freedom is n minus 1, so 29. `t_star` comes from the helper function. `se` is s over root n. And the margin of error, `moe`, is t-star times the standard error.

If you're curious, you can print `t_star` yourself. It's about 2.05. Compare that with 1.96 from the normal distribution. It's slightly bigger, because of that extra uncertainty from estimating s.

Now I'll compute the interval itself.

```
#calculate the confidence interval
c(xbar - moe, xbar + moe)
```

Let me run both chunks. My 95% confidence interval runs from about 7.09 to 7.85 pounds.

Now, how do we interpret this? The correct statement is: "We are 95% confident that the true mean birth weight is between 7.09 and 7.85 pounds."

And here's what "confident" means. It does not mean there's a 95% chance that mu is in this particular interval. Mu is a fixed number. It's either in there or it's not. What's random is the interval, because it came from a random sample. "95% confident" is a statement about the method: if we repeated this whole process over and over, about 95% of the intervals we built would capture the true mean. I'll show you that with a simulation in a few minutes.

Does this CI capture the true mean?

The handout asks: does this interval capture the true mean? Since I'm pretending to know the population, I can actually check.

```
#population mean
pop_mean
#lower bound of CI
lower <- xbar - moe
#upper bound of CI
upper <- xbar + moe
#check if population mean is within the CI bounds
lower < pop_mean & upper > pop_mean
```

I save the lower and upper bounds. Then I ask R two questions at once: is the lower bound below the true mean, and is the upper bound above it? The ampersand means "and" — both have to be true.

Let me run it. TRUE. My interval captured the true mean.

But look how close that was. The lower bound is 7.089, and the true mean is 7.101. I caught it by about one hundredth of a pound. If my sample had been just a little heavier, I would have missed. Keep that in mind — some intervals will miss, and that's built into the method.

If you're following along and your interval looks different, it might capture the mean comfortably, or it might even miss. Either is possible — and in a few minutes, I'll show you how often each one happens.

Part 5 — Exploring CI width

Next question: what happens when I take a bigger sample? I'll go from 30 to 100.

```
#update sample
n <- 100
sample_data <- sample(ncbirths$weight, n)
xbar <- mean(sample_data)
s <- sd(sample_data)
```

Same steps as before: new n, new sample, new x-bar, new s. By the way, `ncbirths$weight` here is the same thing as `population_data`.

Now the same CI recipe.

```
#update CI calculation
alpha <- 0.05
df <- n - 1
t_star <- get_t_star(alpha, df)
se <- s / sqrt(n)
moe <- t_star * se
c(xbar - moe, xbar + moe)
```

Notice I have to recompute the degrees of freedom and t-star, because n changed. Let me run both chunks.

Now the interval runs from about 6.63 to 7.32. Does it still contain 7.101? Yes, it does.

How does it compare to the one with n equals 30? Let's look at the width. The first interval was 7.85 minus 7.09, so roughly 0.76 pounds wide. This one is 7.32 minus 6.63, roughly 0.69 pounds wide. So it got narrower. That's the main lesson: bigger samples give more precise intervals.

But here's something worth noticing. The square root of n went from about 5.5 to 10 — it nearly doubled. So you might expect the interval to be almost half as wide. It's not. It only shrank a little. Why? Let me print s to check.

```
s
```

That's the answer: s changed too. My first sample happened to have a small standard deviation, about 1.0. This new sample's standard deviation is about 1.7. So the bigger sample pulled the interval in, but the noisier sample pushed it back out.

That's sampling variability again. Every piece of the interval — x-bar and s — depends on which sample we happened to draw. On average, though, as n goes up, the interval gets narrower.

Part 6 — Many intervals: simulate and plot

So far I've built two intervals, and both caught the true mean. But "two out of two" doesn't tell us much about what "95%" means. To really see it, we need lots of intervals. That's what `simulate_cis` does.

```
results_df <- simulate_cis(n = 30, alpha = 0.05)
```

I'm asking for intervals from samples of size 30, at 95% confidence. By default, it repeats 100 times. Each repetition — call it trial i — draws a brand-new sample, so every interval comes out a little different.

Let me run it, and then take a peek at what I got.

```
head(results_df)
```

Each row is one trial. `lower` and `upper` are the interval bounds. `contains` is 1 if the interval caught the true mean, and 0 if it missed. `trial` is just the trial number.

Here's one extra line that isn't in the handout, but I find it really useful.

```
mean(results_df$contains)
```

Since `contains` is all 1s and 0s, its mean is the proportion of intervals that captured the true mean. Let me run it. I get ___. That's close to 0.95 — not exactly, because 100 trials is still a limited number.

Plot the simulated intervals

Numbers are nice, but a picture is better. I'm going to plot all 100 intervals.

```
results_df_plot <- results_df %>%
  mutate(contains_f = factor(contains))
```

First, I turn `contains` into a factor — a category. That way ggplot treats 0 and 1 as two groups to color, not as numbers on a scale.

Now the plot itself.

```
ggplot(data = results_df_plot,
       mapping = aes(y = trial)) +
  geom_segment(aes(x = lower, xend = upper, yend = trial, color =
contains_f)) +
  geom_vline(xintercept = pop_mean, linetype = "dashed") +
  scale_color_manual(
    values = c("0" = "red", "1" = "green"),
    breaks = c("0", "1"),
    labels = c("No", "Yes"),
    name = "Contains true mean?"
  )
```

Each interval becomes a horizontal line segment. On the y-axis I put the trial number, so the 100 intervals stack on top of each other. `geom_segment` draws each line from the lower bound to the upper bound, colored by whether it contains the true mean. `geom_vline` draws a dashed vertical line at the true mean. And `scale_color_manual` says: misses in red, hits in green, with a legend that reads "No" and "Yes."

Let me run it.

Here it is. Every horizontal line is one confidence interval from one sample. The dashed line is the true mean, 7.1. Green lines cross the dashed line — they caught it. Red lines miss it completely.

Look at how the intervals jump around. Their centers shift left and right, and their lengths differ too, because each sample has its own x-bar and its own s. But the vast majority cross the dashed line. Only a handful are red. Let me count them — in my plot, that's ___ out of 100.

This is what 95% confidence means. The true mean never moves — it's the dashed line. The intervals are what move. If we build intervals this way over and over, about 95% of them will catch the truth.

And for any single interval, like the one I built earlier, we don't know whether it's one of the green ones or one of the red ones. We only know that the method works about 95% of the time.

Part 7 — Summary and practice

To wrap up, let me scroll down to the summary table at the end of the handout. t-star comes from `get_t_star`, using alpha and the degrees of freedom. x-bar is the mean of the sample, and s is the standard deviation of the sample. Mu is the mean of the population. The margin of error is t-star times the standard error. And the confidence interval is x-bar plus or minus the margin of error. Finally, `simulate_cis` lets us repeat the whole thing many times, to see how often the method is wrong.

Now it's your turn. Right below the summary table, there are three practice problems.

One: change n to 10, resample, and build a 95% interval. Before you run it, make a prediction. Will the margin of error go up or down? Think about two things: the standard error, and t-star with only 9 degrees of freedom.

Two: set alpha to 0.10. That gives you a 90% confidence interval. Predict what happens to the width. And think about the trade-off: is a narrower interval free, or are you giving something up?

Three: try a different variable — `mage`, the mother's age — with a sample of 50. Hint: you'll need to change which column you sample from, and the true mean you compare against will be different too.

For each one, predict the answer before you run the code. That's how you'll know you actually understand it, rather than just copying.

That's it for Handout 6. To recap the big idea: a confidence interval is x-bar plus or minus a margin of error. The margin shrinks with bigger samples and grows with higher confidence. And "95% confident" describes how often the method works across many, many samples. Thanks for watching, and I'll see you in the next one!