2.6 — Inference for Regression

ECON 480 • Econometrics • Fall 2022

Dr. Ryan Safner
Associate Professor of Economics

safner@hood.edu
ryansafner/metricsF22
metricsF22.classes.ryansafner.com

Contents

Why Uncertainty Matters

Recall: Two Big Problems with Data

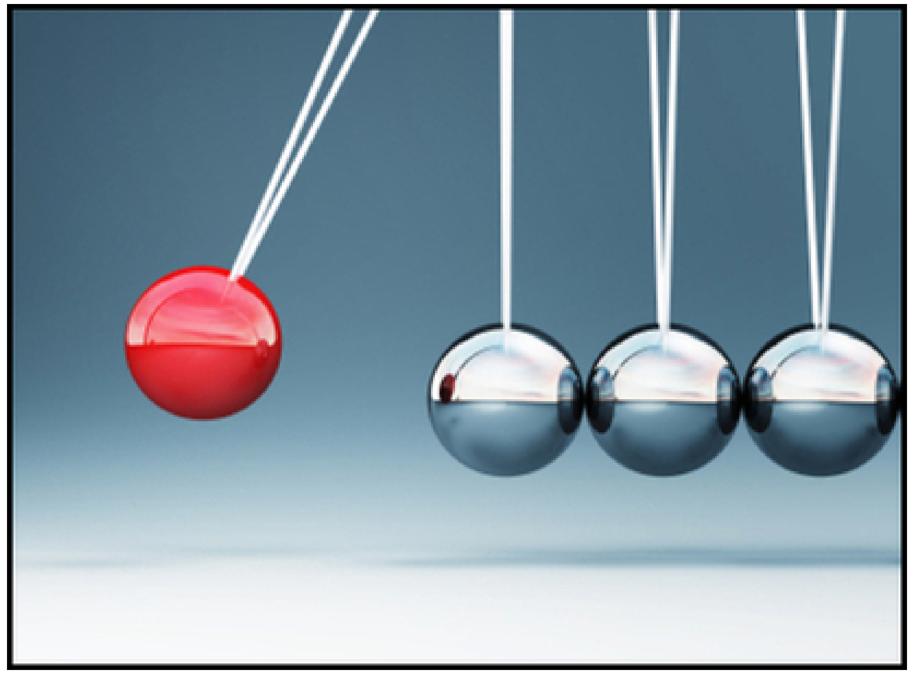
- We use econometrics to **identify** causal relationships & make **inferences** about them:

1. Problem for **identification**: **endogeneity**

- X is **exogenous** if $cor(x, u) = 0$
- X is **endogenous** if $cor(x, u) \neq 0$

2. Problem for **inference**: **randomness**

    - Data is random due to **natural sampling variation**
    - Taking one sample of a population will yield slightly different information than another sample of the same population



Distributions of the OLS Estimators

$$Y_i = \beta_0 + \beta_1 X_i + u_i$$

- OLS estimators ($\hat{\beta_0}$ and $\hat{\beta_1}$) are computed from a finite (specific) sample of data
- Our OLS model contains **2 sources of randomness**:

- ***Modeled* randomness**: population u_i includes all factors affecting Y *other* than X
  - different samples will have different values of those other factors (u_i)

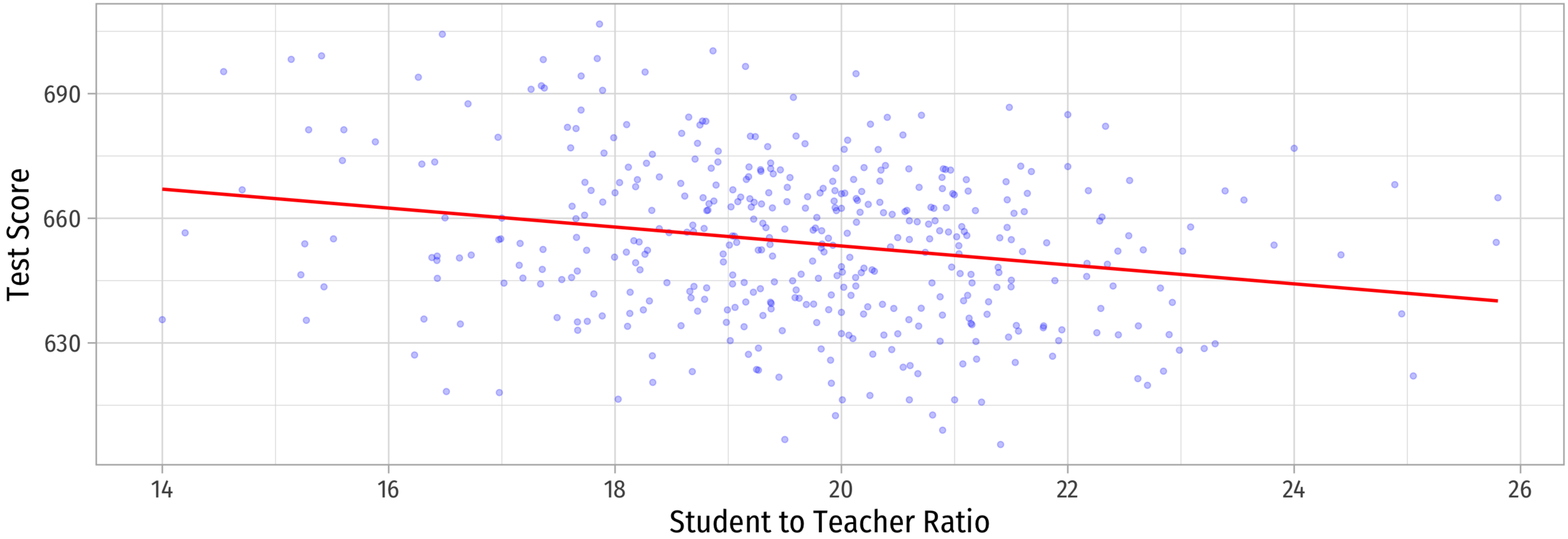
- ***Sampling* randomness**: different samples will generate different OLS estimators
  - Thus, $\hat{\beta_0}, \hat{\beta_1}$ are *also* **random variables**, with their own **sampling distribution**

The Two Problems: Where We're Heading...Ultimately

$$\textbf{Sample} \xrightarrow{\text{statistical inference}} \textbf{Population} \xrightarrow{\text{causal indentification}} \textbf{Unobserved Parameters}$$

- We want to **identify** causal relationships between **population** variables
  - Logically first thing to consider
  - **Endogeneity problem**
- We'll use **sample** *statistics* to **infer** something about population *parameters*
  - In practice, we'll only ever have a finite *sample distribution* of data
  - We *don't* know the *population distribution* of data
  - **Randomness problem**

Why Sample vs. Population Matters

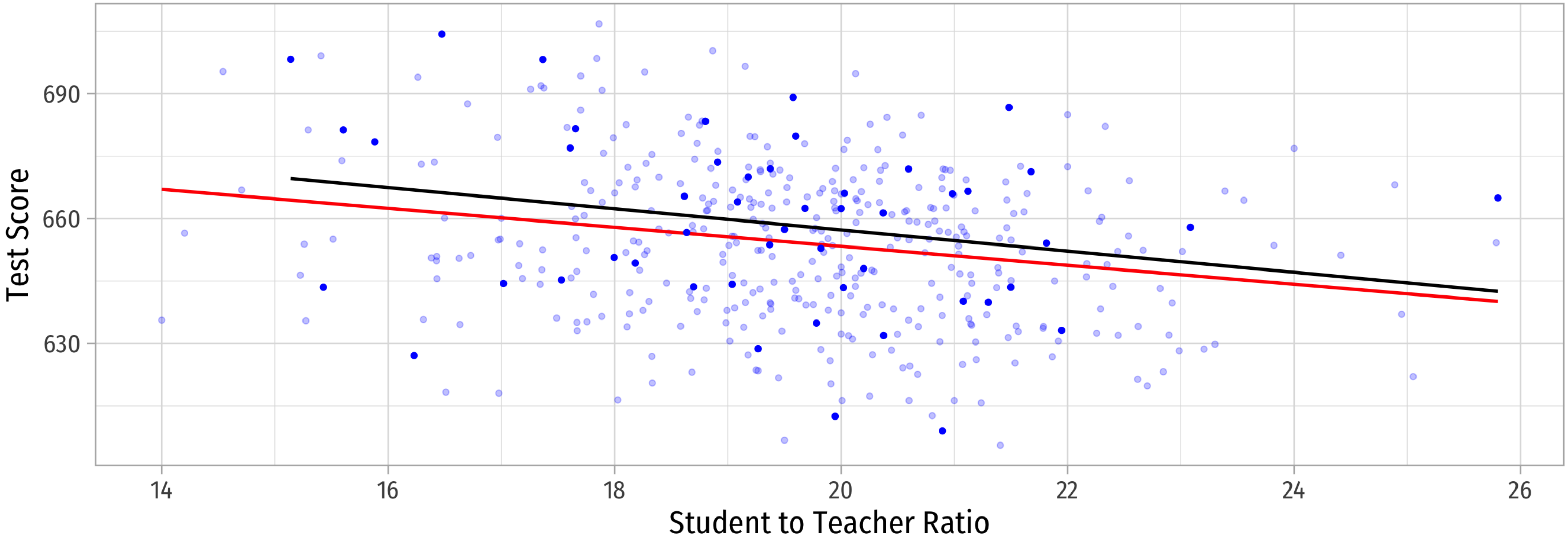


Population relationship

$$Y_i = 698.93 + -2.28X_i + u_i$$

$$Y_i = \beta_0 + \beta_1 X_i + u_i$$

Why Sample vs. Population Matters



Sample 1: 50 random observations

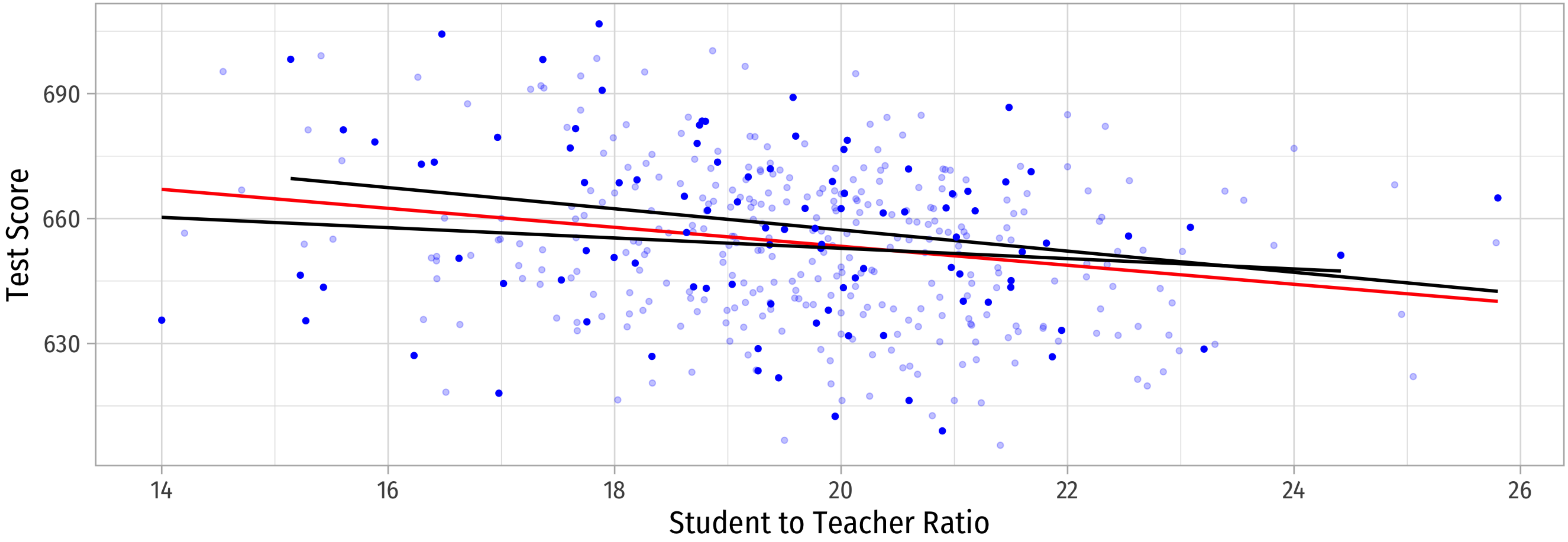
Population relationship

$$Y_i = 698.93 + -2.28X_i + u_i$$

Sample relationship

$$\hat{Y}_i = 708.12 + -2.54X_i$$

Why Sample vs. Population Matters



Sample 2: 50 random individuals

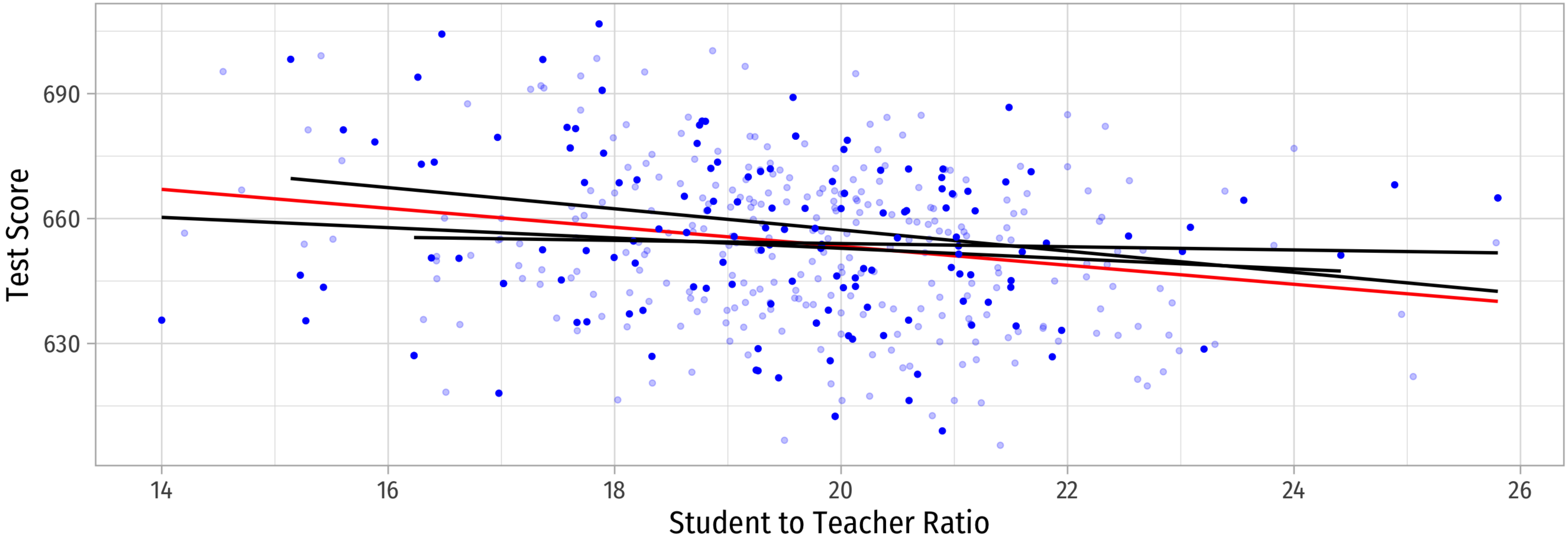
Population relationship

$Y_i = 698.93 + -2.28X_i + u_i$

Sample relationship

$\hat{Y}_i = 708.12 + -2.54X_i$

Why Sample vs. Population Matters



Sample 3: 50 random individuals

Population relationship

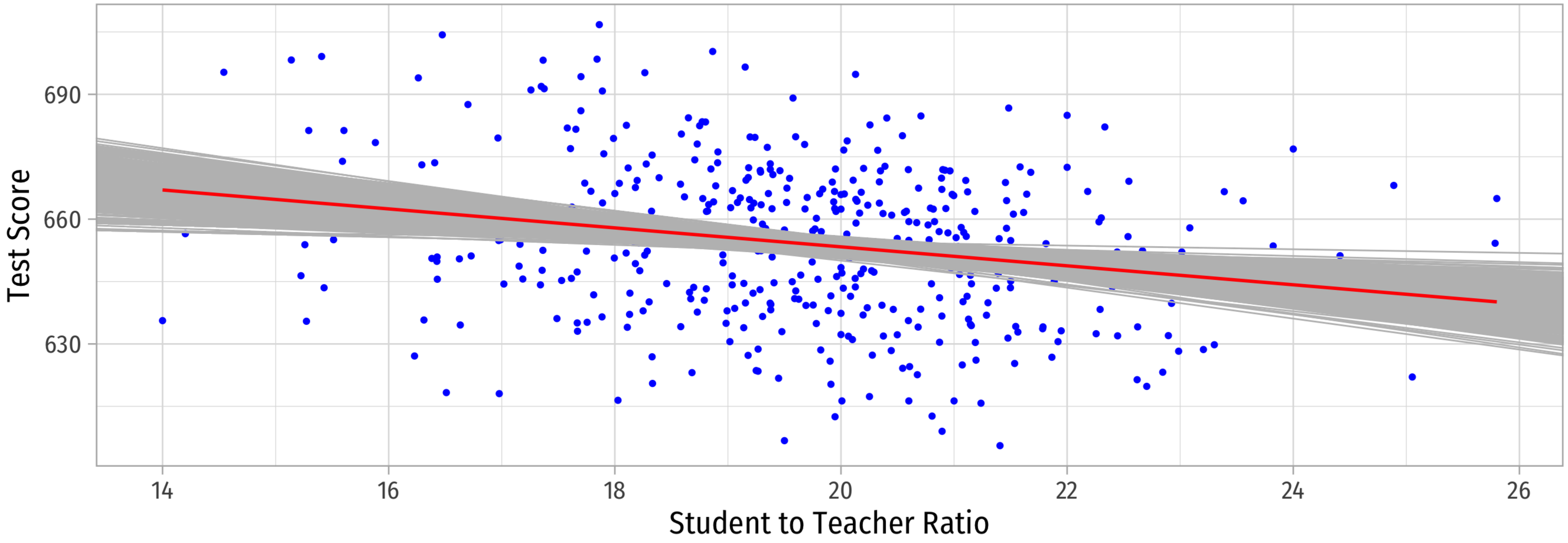
$Y_i = 698.93 + -2.28X_i + u_i$

Sample relationship

$\hat{Y}_i = 708.12 + -2.54X_i$

Why Sample vs. Population Matters

- Let's repeat this process **10,000 times**!
- This exercise is called a **(Monte Carlo) simulation**
  - I'll show you how to do this next class with the `infer` package

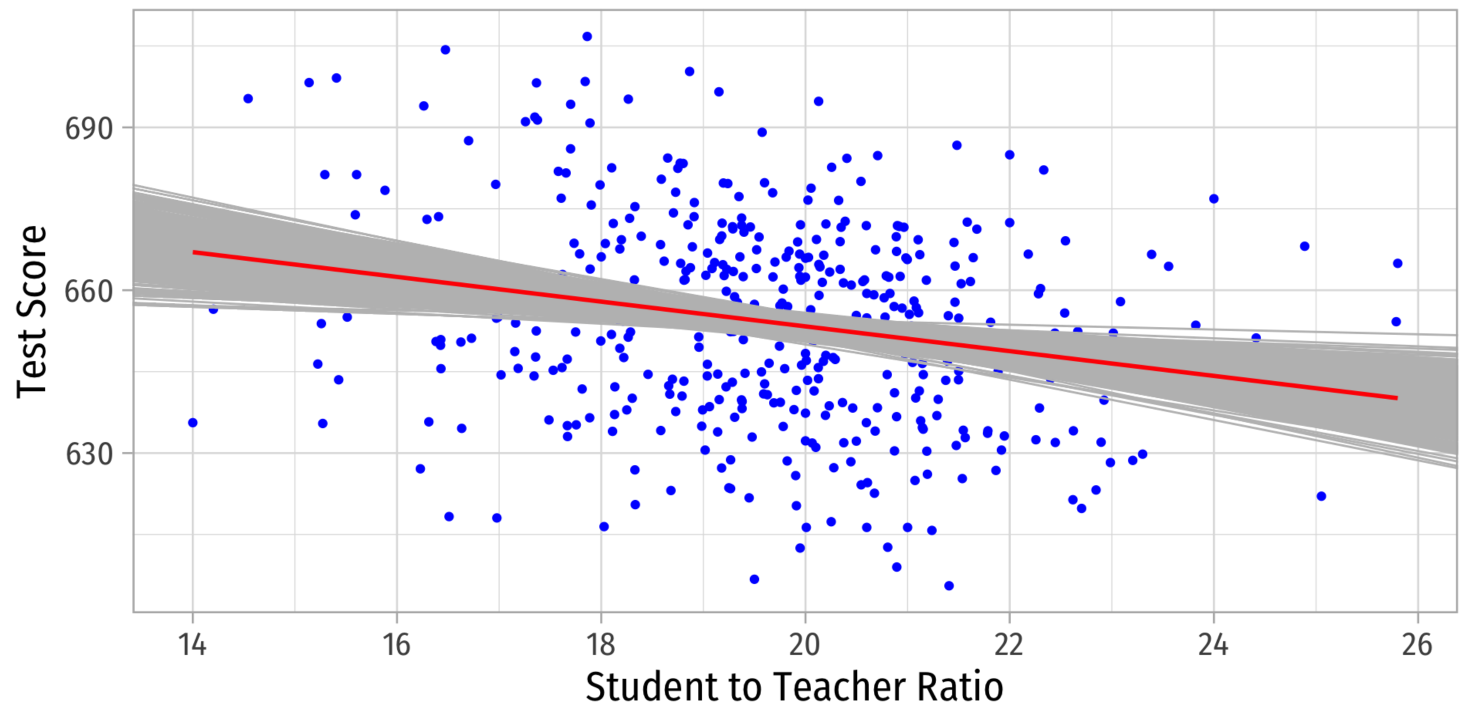


Why Sample vs. Population Matters

- ***On average***, estimated regression lines from (hypothetical) samples provide an unbiased estimate of true population regression line

$$\mathbb{E}[\hat{\beta}_1] = \beta_1$$

- But, any *individual* estimate can miss the mark
- This leads to **uncertainty** about our estimated regression line
  - We only have *1* sample in reality!
  - This is why we care about the **standard error** of our line: $se(\hat{\beta}_1)$!



Confidence Intervals

Statistical Inference

$$\textbf{Sample} \xrightarrow{\text{statistical inference}} \textbf{Population} \xrightarrow{\text{causal indentification}} \textbf{Unobserved Parameters}$$

- We want to start **inferring** what the true population regression model is, using our estimated regression model from our sample

$$\hat{Y}_i = \hat{\beta}_0 + \hat{\beta}_1 X \xrightarrow{\text{🤞 hopefully 🤞}} Y_i = \beta_0 + \beta_1 X + u_i$$

- We can't yet make ***causal* inferences** about whether/how X *causes* Y
  - coming after the midterm!

Estimation and Statistical Inference

- Our problem with **uncertainty** is we don't know whether our sample estimate is *close* or *far* from the unknown population parameter
- But we can use our errors to learn how well our model statistics likely estimate the true parameters
- Use $\hat{\beta}_1$ and its standard error, $se(\hat{\beta}_1)$ for statistical inference about true β_1
- We have two options...

Estimation and Statistical Inference

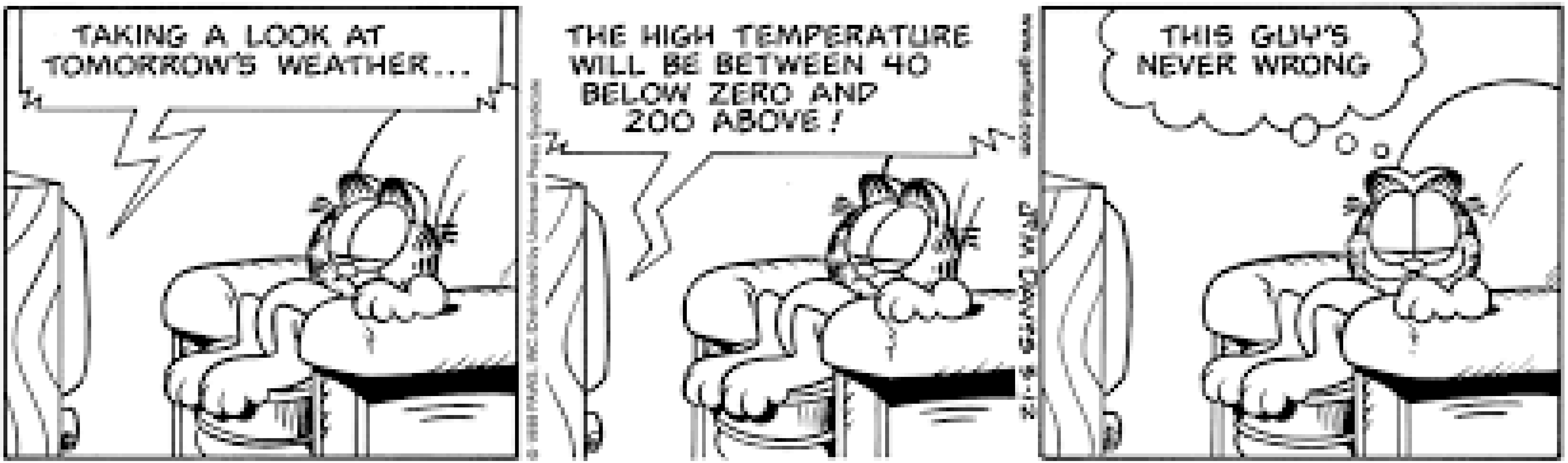
Point estimate

- Use our $\hat{\beta}_1$ & $se(\hat{\beta}_1)$ to determine if statistically significant evidence to reject a hypothesized β_1

Confidence Interval

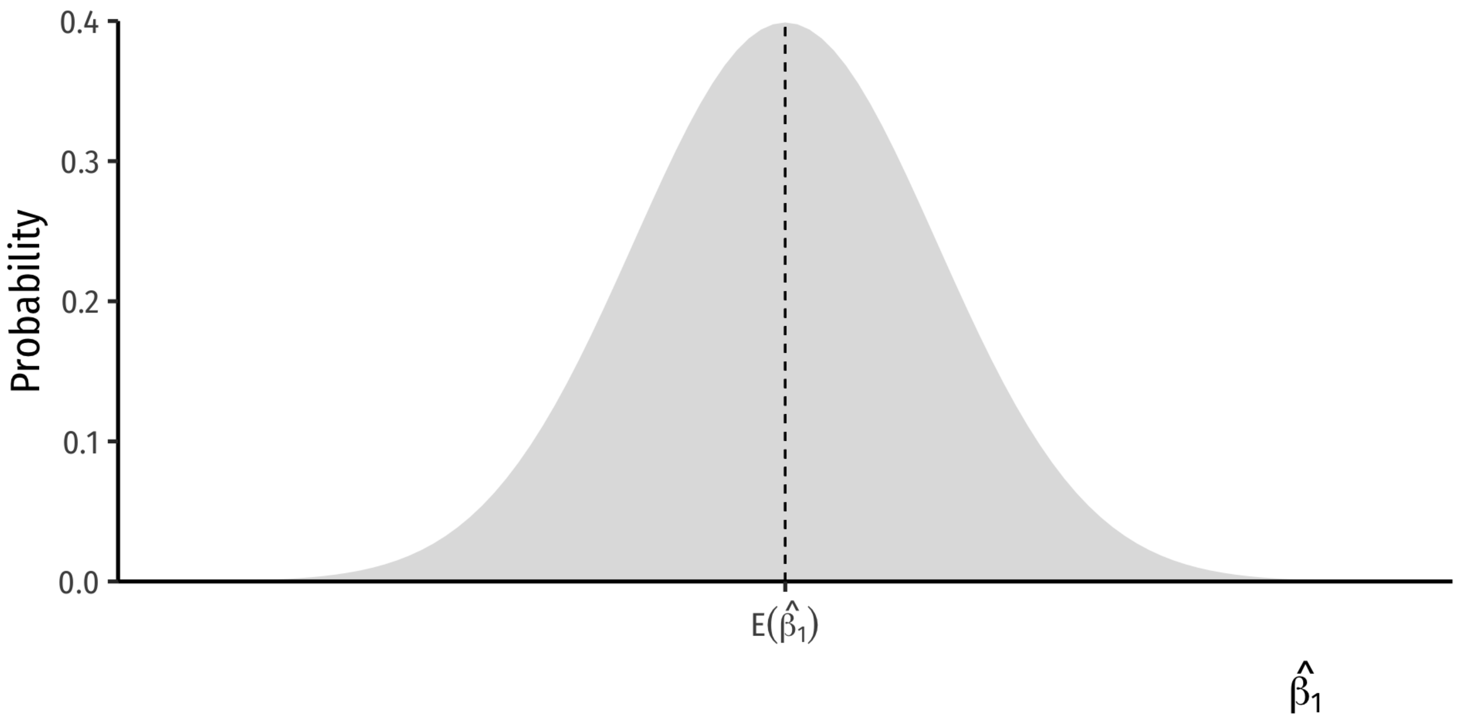
- Use our $\hat{\beta}_1$ & $se(\hat{\beta}_1)$ to create a *range* of values that gives us a good chance of capturing the true β_1

Accuracy vs. Precision



Generating Confidence Intervals

- We can generate our confidence interval by generating a **"bootstrap"** sampling distribution:
  - Take our sample data and resample it many times by selecting random observations and then replacing them
- This allows us to approximate the sampling distribution of $\hat{\beta}_1$ by simulation!



Confidence Intervals Using the `infer` Package

Confidence Intervals Using the `infer` Package I

- The `infer` package allows you to do statistical inference in a *tidy* way, following the philosophy of the `tidyverse`

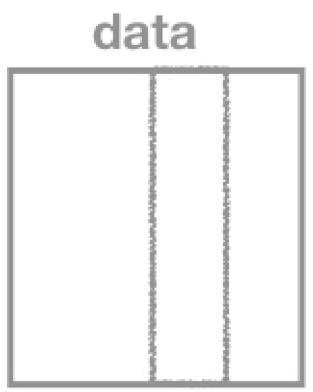
```r
# install.packages("infer")

# load
library(infer)
```

Confidence Intervals Using the `infer` Package II

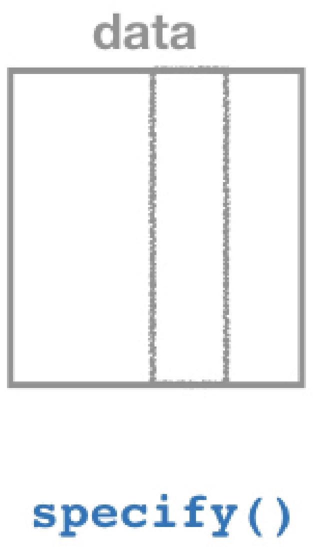
- `infer` allows you to run through these steps manually to understand the process:

1. `specify()` a model
2. `generate()` a bootstrap distribution
3. `calculate()` the confidence interval
4. `visualize()` with a histogram (optional)

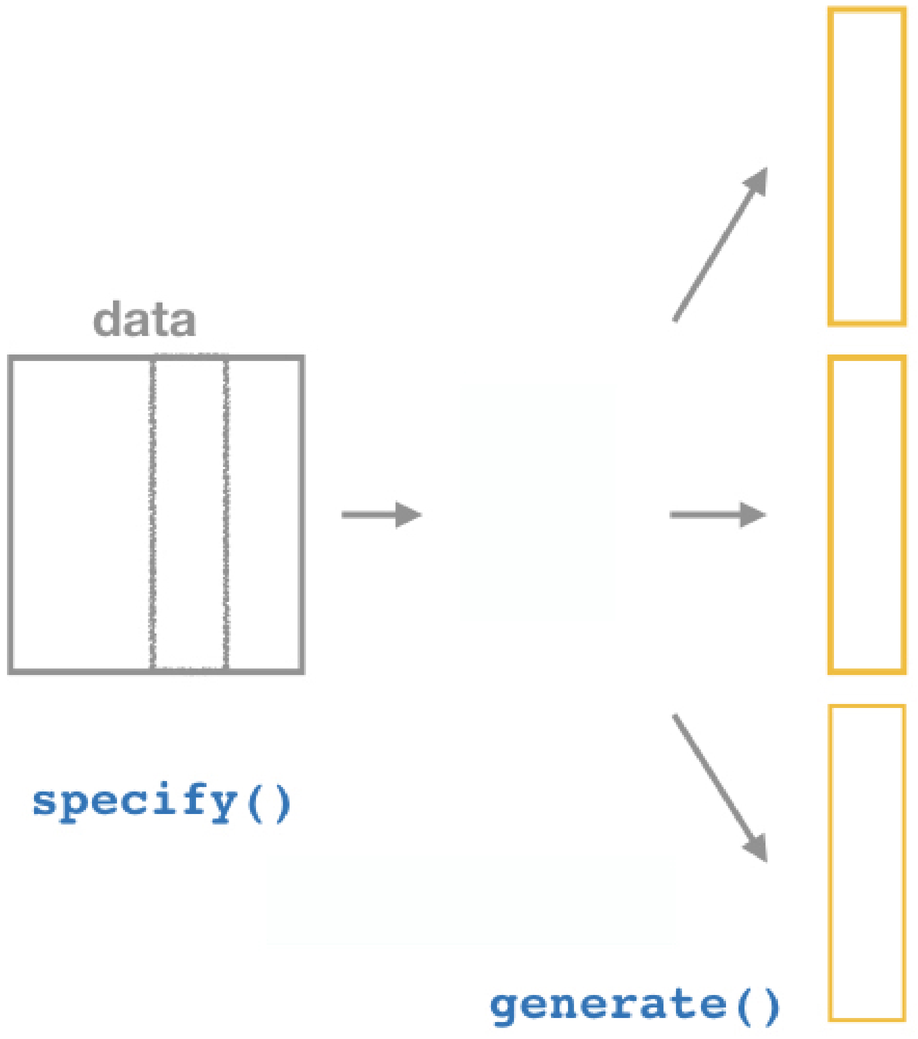
Confidence Intervals Using the `infer` Package III



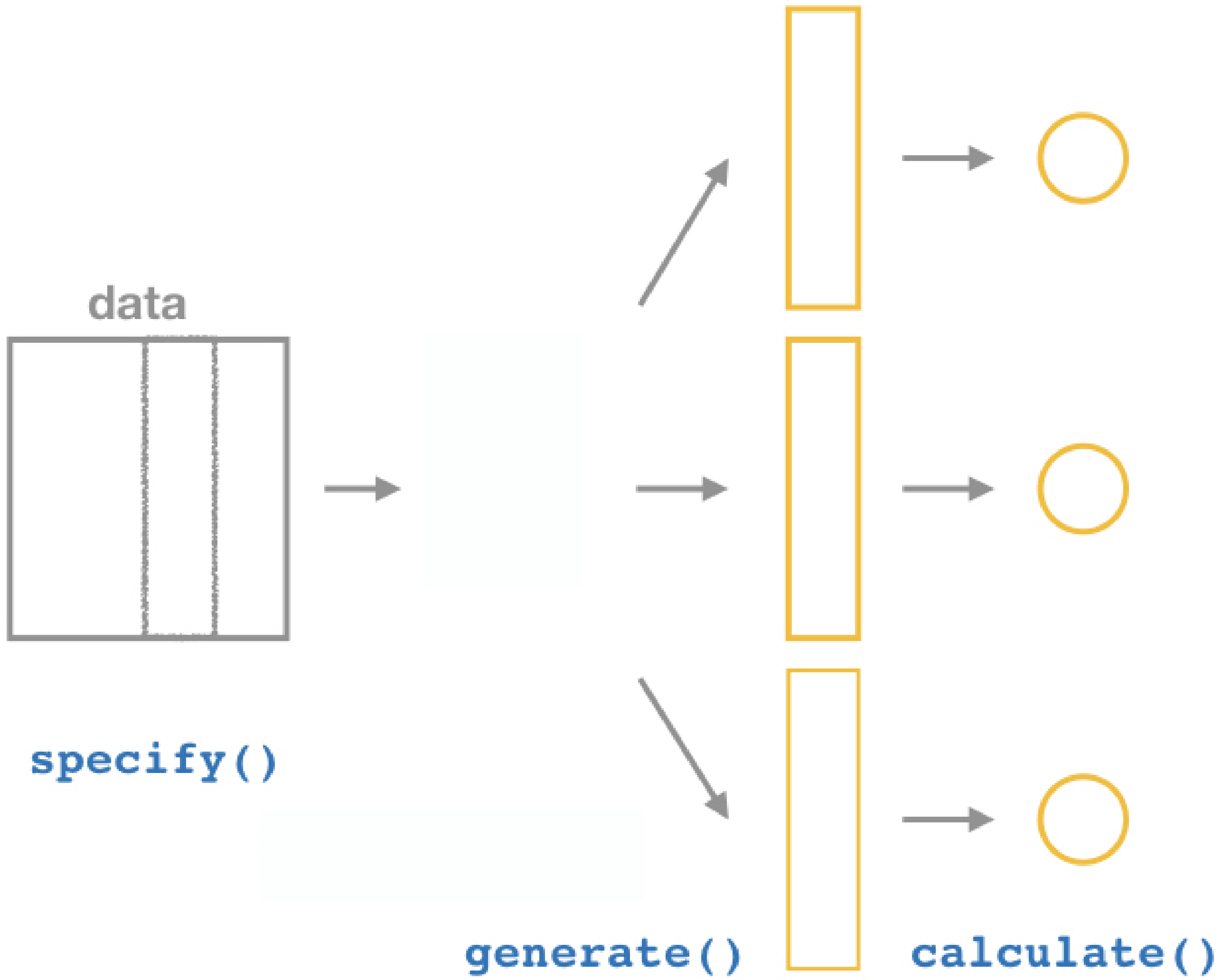
Confidence Intervals Using the `infer` Package III



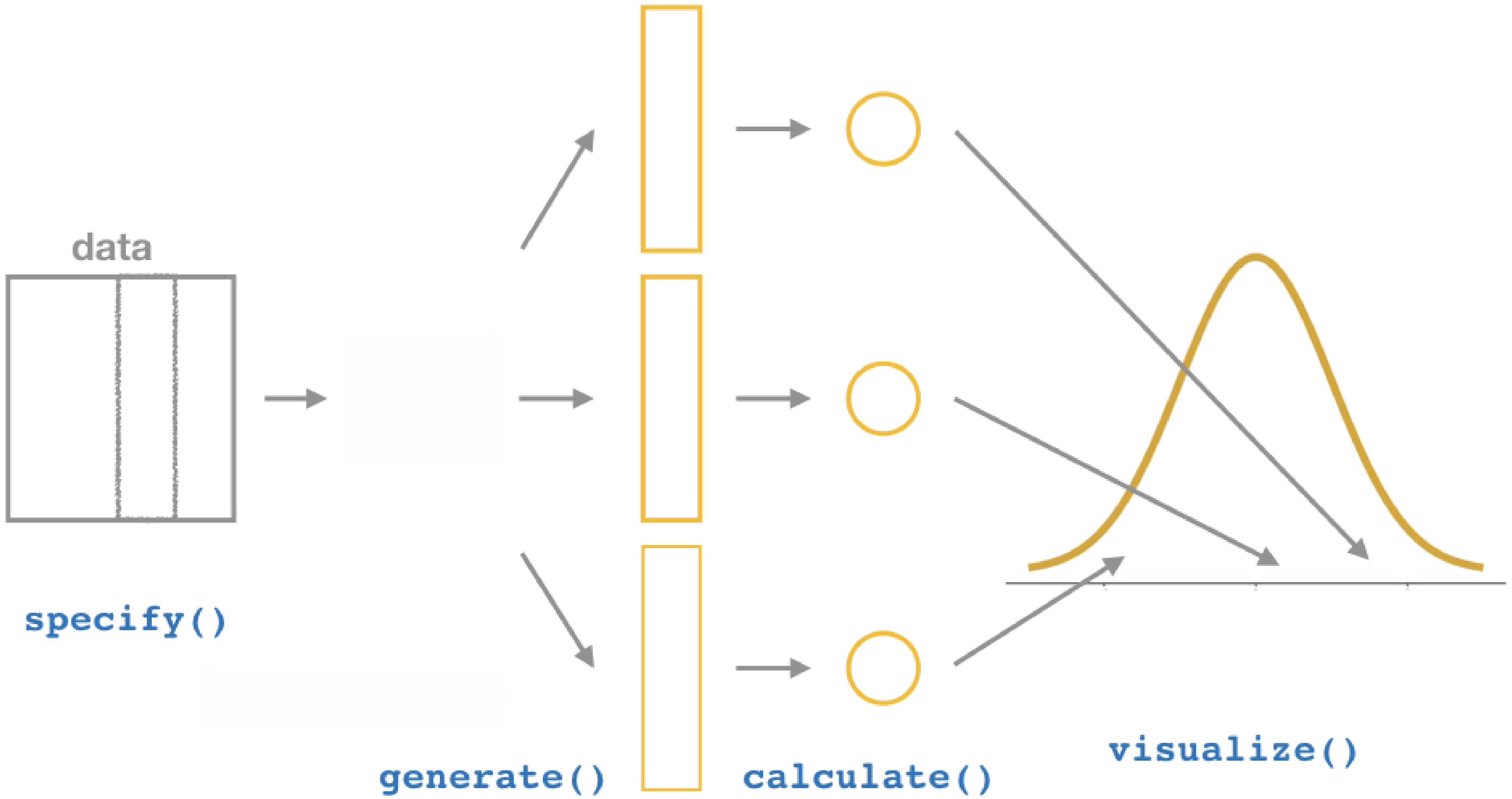
Confidence Intervals Using the `infer` Package III



Confidence Intervals Using the `infer` Package III



Confidence Intervals Using the `infer` Package III



Bootstrapping

Our Sample

term <chr>
(Intercept)
str

2 rows | 1-1 of 5 columns

Another “Sample”

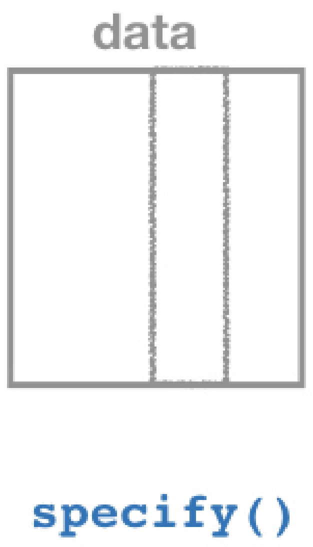
term <chr>
(Intercept)
str

2 rows | 1-1 of 5 columns

👆 Bootstrapped from Our Sample

- Now we want to do this 1,000 times to simulate the (unknown) sampling distribution of $\hat{\beta}_1$

The `infer` Pipeline: `specify()`



The `infer` Pipeline: `specify()`

Specify

`data %>%`

`specify(y ~ x)`

- Take our data and pipe it into the `specify()` function, which is essentially a `lm()` function for regression (for our purposes)

```
ca_school %>%
  specify(testscr ~ str)
```

testscr	str
<dbl>	<dbl>
690.80	17.88991
661.20	21.52466
643.60	18.69723
647.70	17.35714
640.85	18.67133
605.55	21.40625
606.75	19.50000

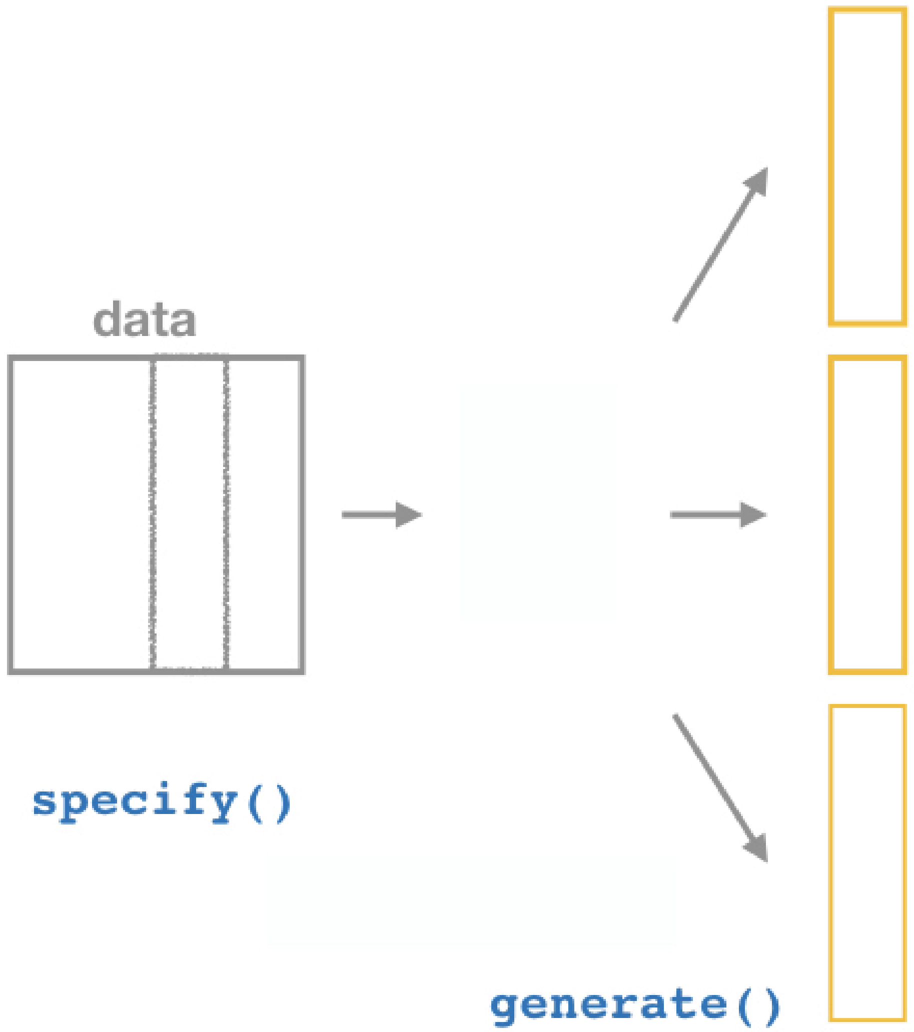
testscr str
<dbl> <dbl>

609.00	20.89412
612.50	19.94737
612.65	20.80556

1-10 of 420 rows

Previous **1** 2 3 4 5 6 ... 42 Next

The `infer` Pipeline: `generate()`



The `infer` Pipeline: `generate()`

Specify

Generate

`%>% generate(reps = n, type = "bootstrap")`

- Now the magic starts, as we run a number of simulated samples
- Set the number of `reps` and set `type` to `"bootstrap"`

```
ca_school %>%
  specify(testscr ~ str) %>%
  generate(reps = 1000, #<<
           type = "bootstrap") #<<
```

The `infer` Pipeline: `generate()`

Specify

Generate

```
%>% generate(reps = n, type = "bootstrap")
```

- Now the magic starts, as we run a number of simulated samples
- Set the number of `reps` and set `type` to `"bootstrap"`

replicate <int>	testscr <dbl>
1	640.85
1	665.65
1	667.45
1	636.50
1	662.90
1	660.05
1	639.85
1	671.60

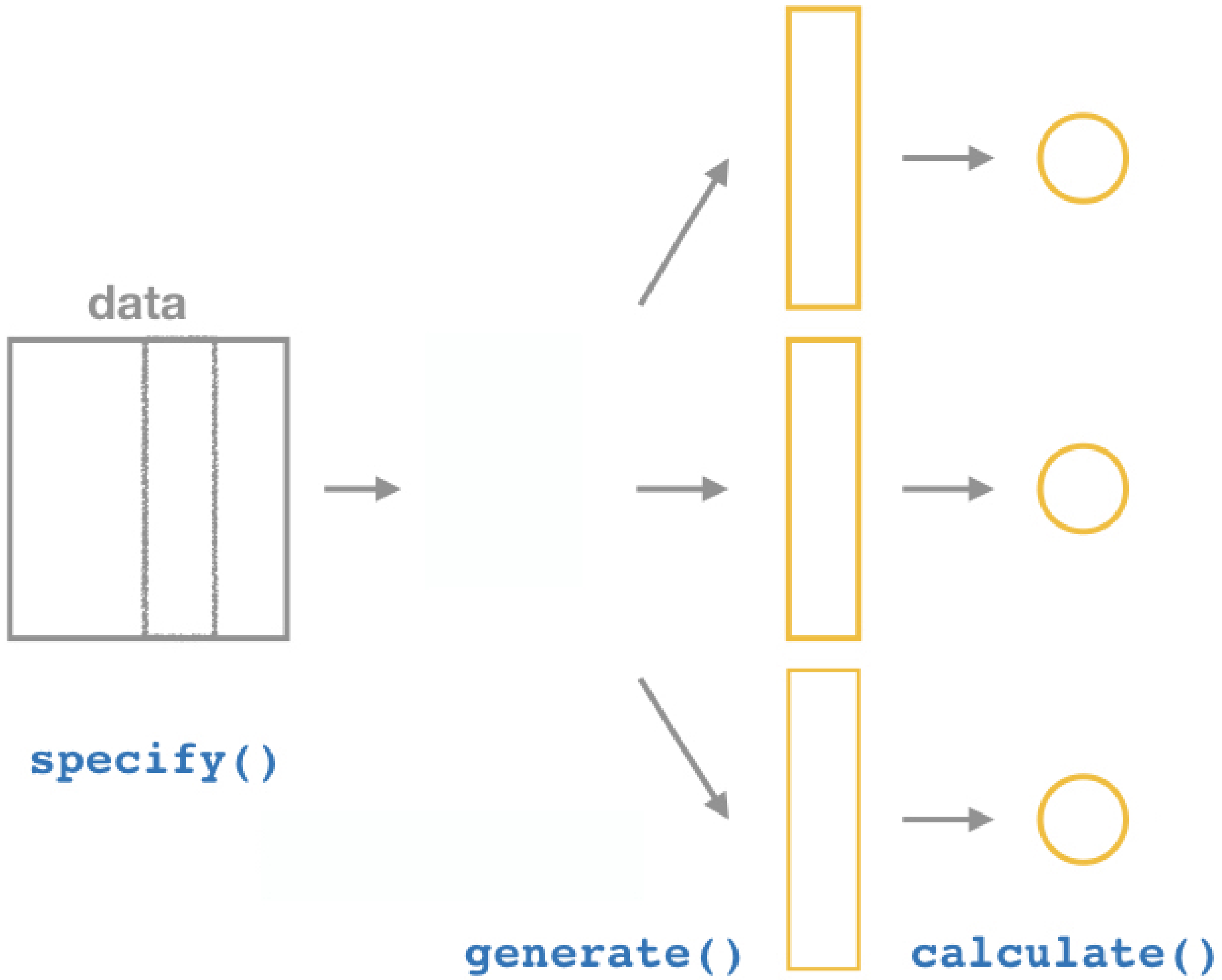
replicate <int> testscr <dbl>

1	655.30
1	669.80

1-10 of 10,000 rows | 1-2 of 3 columns Previous 1 2 3 4 5 6 ... 1000 Next

- `replicate`: the "sample" number (1-1000)
- creates `x` and `y` values (data points)

The `infer` Pipeline: `calculate()`



The `infer` Pipeline: `calculate()`

Specify

Generate

Calculate

`%>% calculate(stat = "slope")`

```r
ca_school %>%
  specify(testscr ~ str) %>%
  generate(reps = 1000,
           type = "bootstrap") %>%
  calculate(stat = "slope") #<<
```

- For each of the 1,000 replicates, calculate `slope` in `lm(testscr ~ str)`
- Calls it the `stat`

The `infer` Pipeline: `calculate()`

The `infer` Pipeline: `calculate()`

Specify

Generate

Calculate

`%>% calculate(stat = "slope")`

```r
boot <- ca_school %>%
  specify(testscr ~ str) %>%
  generate(reps = 1000,
           type = "bootstrap") %>%
  calculate(stat = "slope")
```

- `boot` is (our simulated) sampling distribution of $\hat{\beta}_1$!
- We can now use this to estimate the confidence interval from *our* $\hat{\beta}_1 = -2.28$
- And visualize it

Confidence Interval

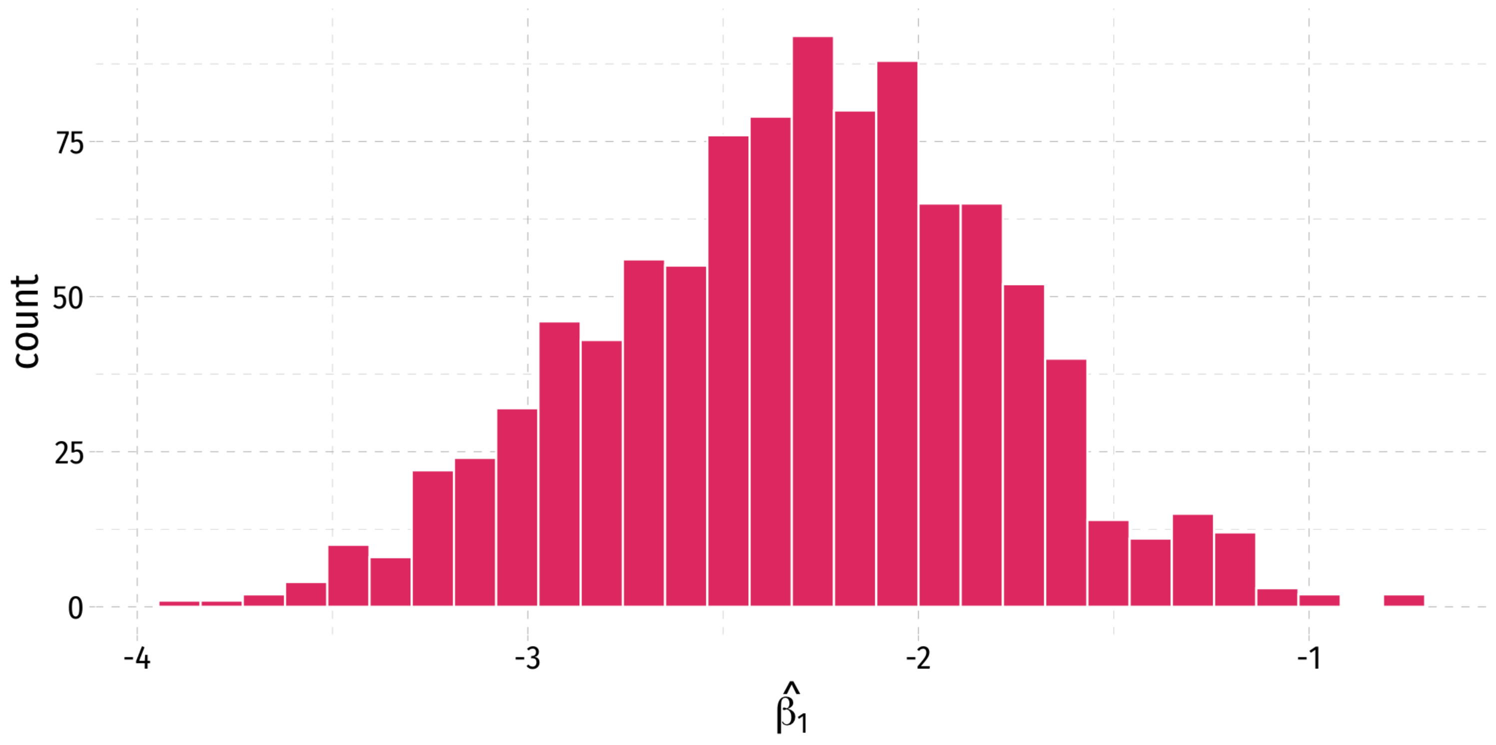
- A 95% confidence interval is the middle 95% of the sampling distribution

```
ci <- boot %>%
  summarize(lower = quantile(stat, 0.025),
            upper = quantile(stat, 0.975))
ci
```

lower <dbl>
-3.314213

1 row | 1-1 of 2 columns

```
sampling_dist <- ggplot(data = boot)+
  aes(x = stat)+
  geom_histogram(color="white", fill = "#e64173")+
  labs(x = expression(hat(beta[1])))+
  theme_pander(base_family = "Fira Sans Condensed",
           base_size=20)

sampling_dist
```



Confidence Interval

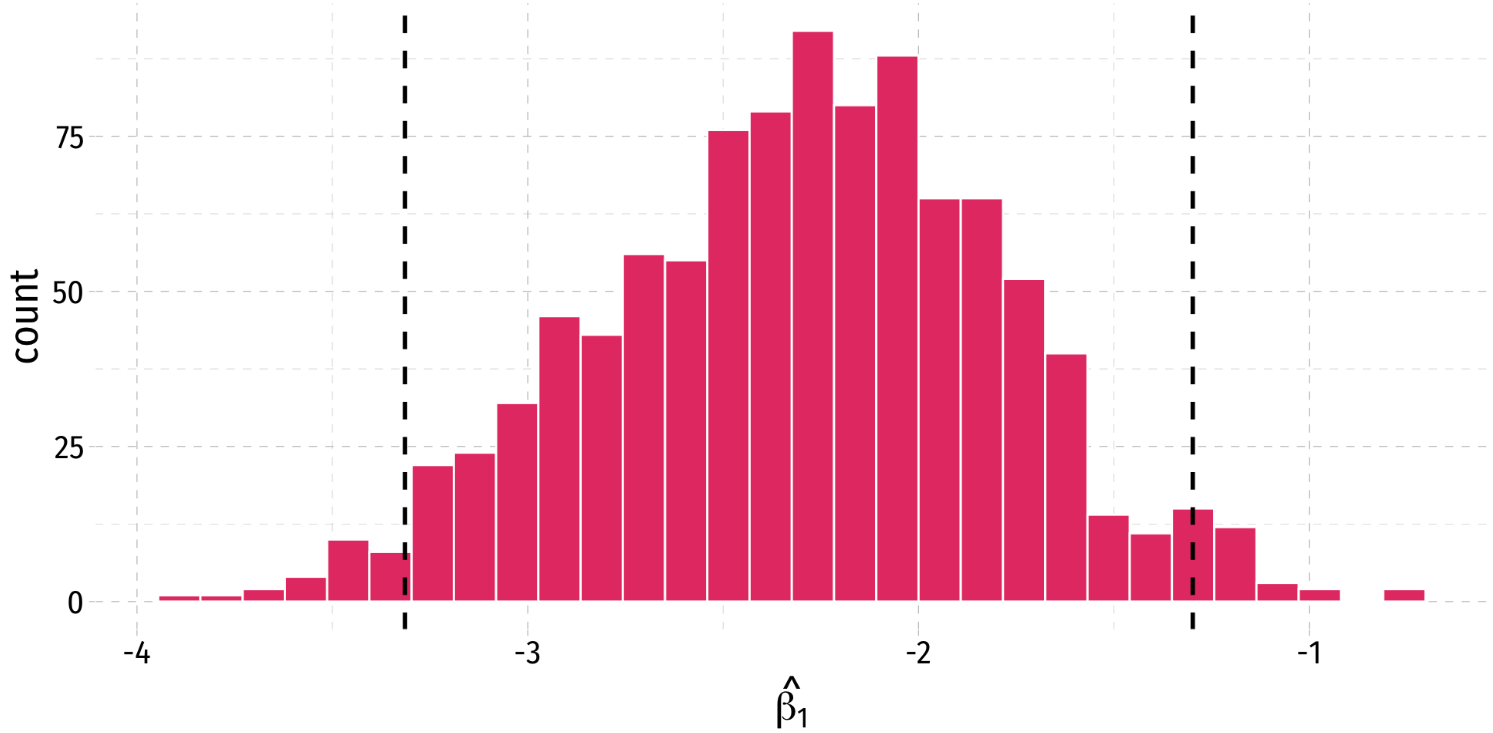
- A 95% confidence interval is the middle 95% of the sampling distribution

```
ci <- boot %>%
  summarize(lower = quantile(stat, 0.025),
            upper = quantile(stat, 0.975))
ci
```

lower <dbl>
-3.314213

1 row | 1-1 of 2 columns

```
sampling_dist+
  geom_vline(data = ci, aes(xintercept = lower), s
  geom_vline(data = ci, aes(xintercept = upper), s
```



The infer Pipeline: get_confidence_interval()

Specify

Generate

Calculate

Get Confidence Interval

%>%
get_confidence_interval()

```
ca_school %>% #<< # save this
  specify(testscr ~ str) %>%
  generate(reps = 1000,
           type = "bootstrap") %>%
  calculate(stat = "slope") %>%
  get_confidence_interval(level = 0.95, #<<
                          type = "se", #<<
                          point_estimate = -2.28) #<<
```

lower_ci <dbl>	upper_ci <dbl>
-3.298084	-1.261916

1 row

Broom Can Estimate a Confidence Interval

```r
school_reg %>%
  tidy(conf.int = T)
```

term <chr>	estimate <dbl>
(Intercept)	698.932952
str	-2.279808

2 rows | 1-2 of 7 columns

```r
our_CI <- school_reg %>%
  tidy(conf.int = T) %>%
  filter(term == "str") %>%
  select(conf.low, conf.high)

our_CI
```

conf.low <dbl>	conf.high <dbl>

conf.low <dbl>	conf.high <dbl>
-3.22298	-1.336637

1 row

The `infer` Pipeline: `visualize()`

Specify

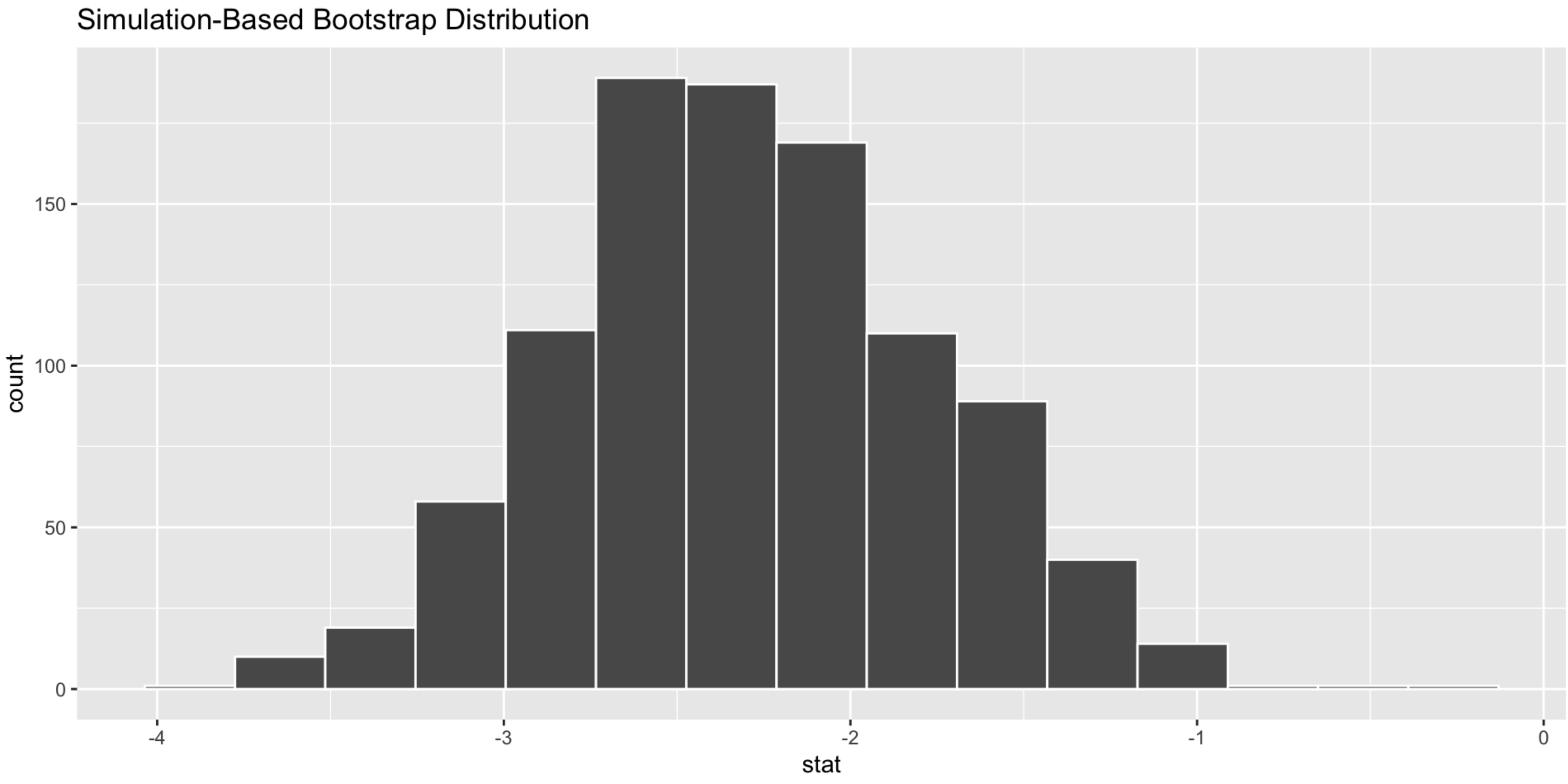
Generate

Calculate

Visualize

`%>% visualize()`

```
ca_school %>%
  specify(testscr ~ str) %>%
  generate(reps = 1000,
           type = "bootstrap") %>%
  calculate(stat = "slope") %>%
  visualize() #<<
```



The `infer` Pipeline: `visualize()`

Specify

Generate

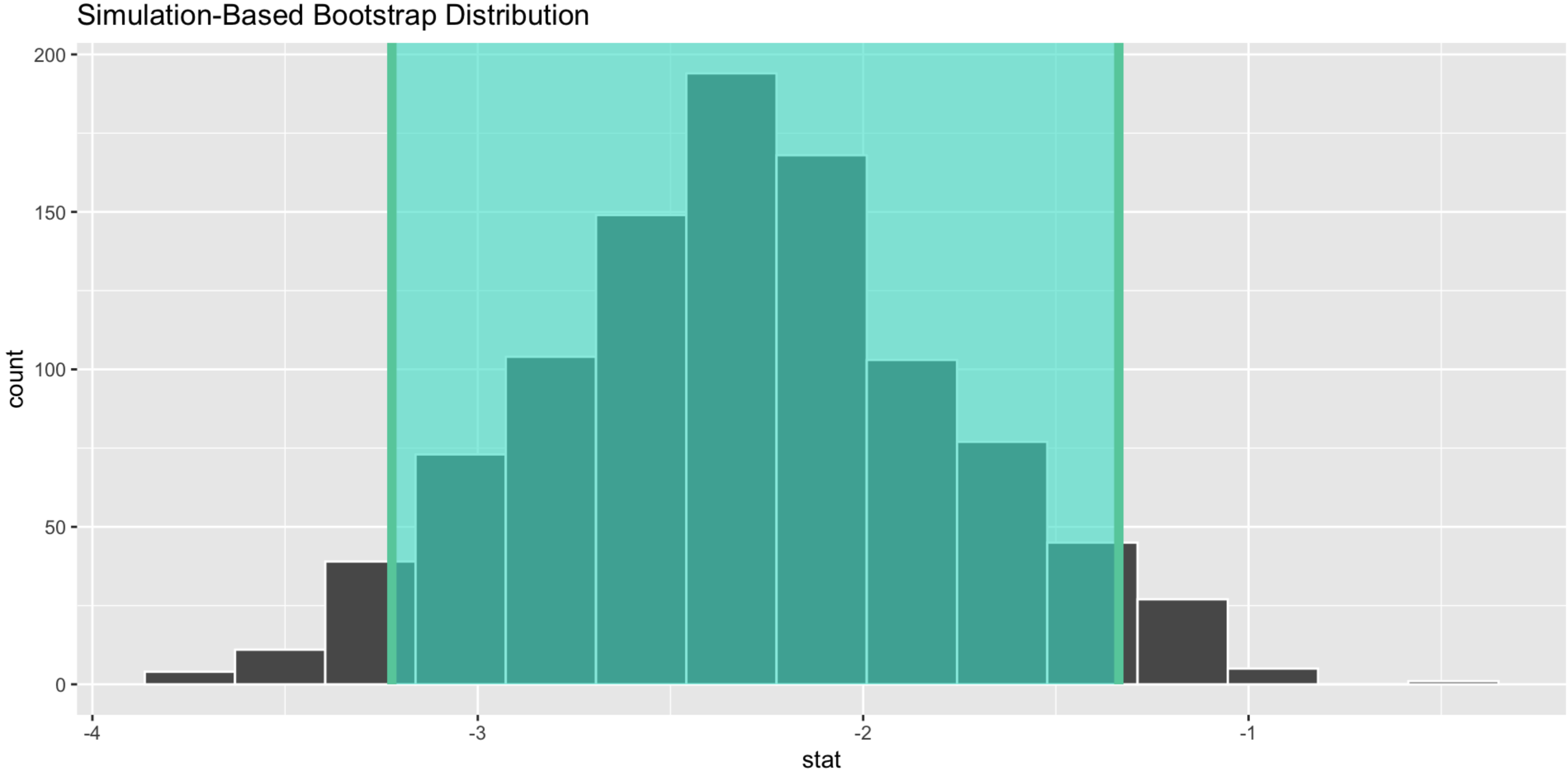
Calculate

Visualize

`%>% visualize()`

- If we have our confidence levels saved (`our_CI`) we can `shade_ci()` in `infer`'s `visualize()` function

```
ca_school %>%
  specify(testscr ~ str) %>%
  generate(reps = 1000,
           type = "bootstrap") %>%
  calculate(stat = "slope") %>%
  visualize()+
  shade_ci(endpoints = our_CI)
```



Confidence Intervals, Theory

Confidence Intervals, Theory

- In general, a **confidence interval (CI)** takes a point estimate and extrapolates it within some **margin of error (MOE)**:

$$\Big(\big[\text{estimate - MOE}\big], \big[\text{estimate + MOE}\big]\Big)$$

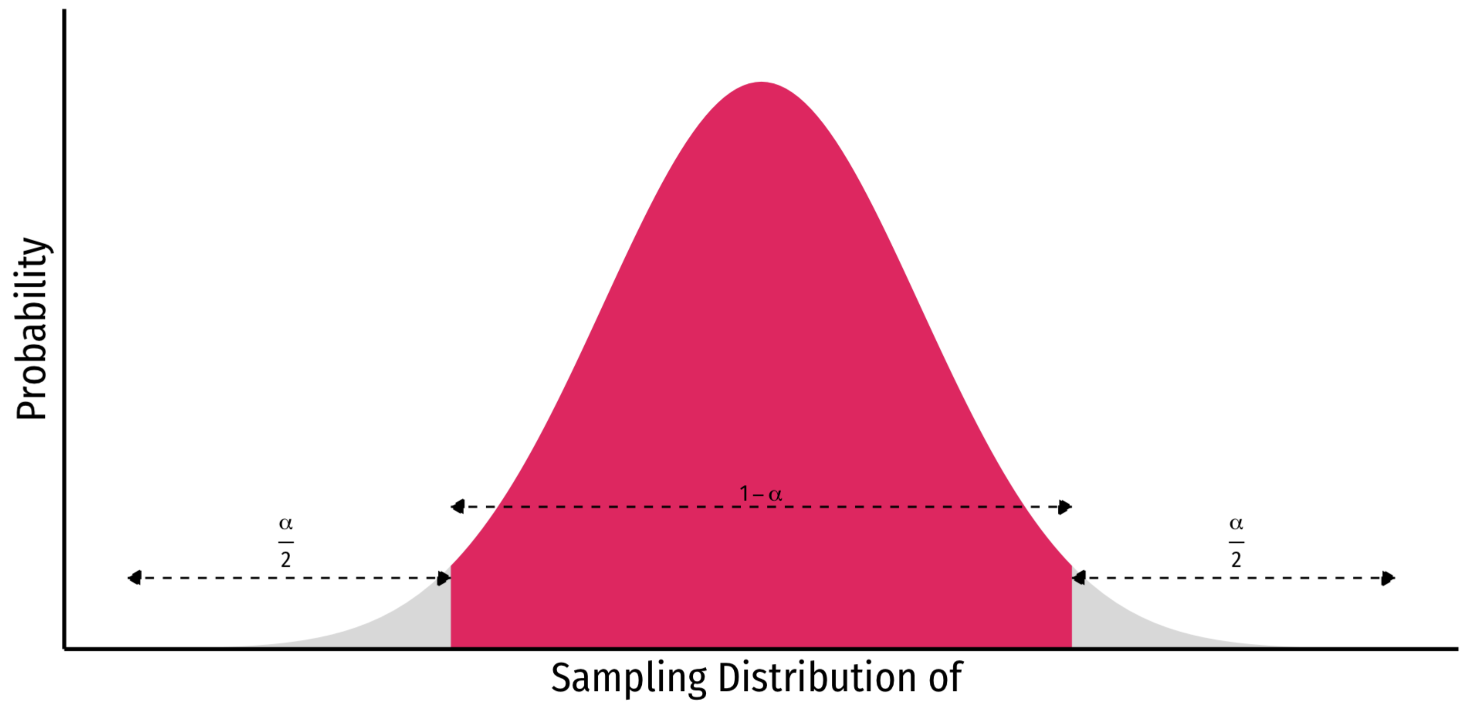
- The main question is, **how confident do we want to be** that our interval contains the true parameter?
  - Larger confidence level, larger margin of error (and thus larger interval)

Confidence Intervals, Theory

- $(1-\alpha)$ is the **confidence level** of our confidence interval
  - α is the **"significance level"** that we use in hypothesis testing
  - $\alpha =$ probability that the true parameter is *not* contained within our interval
- Typical levels: 90%, 95%, 99%
  - 95% is especially common, $\alpha = 0.05$

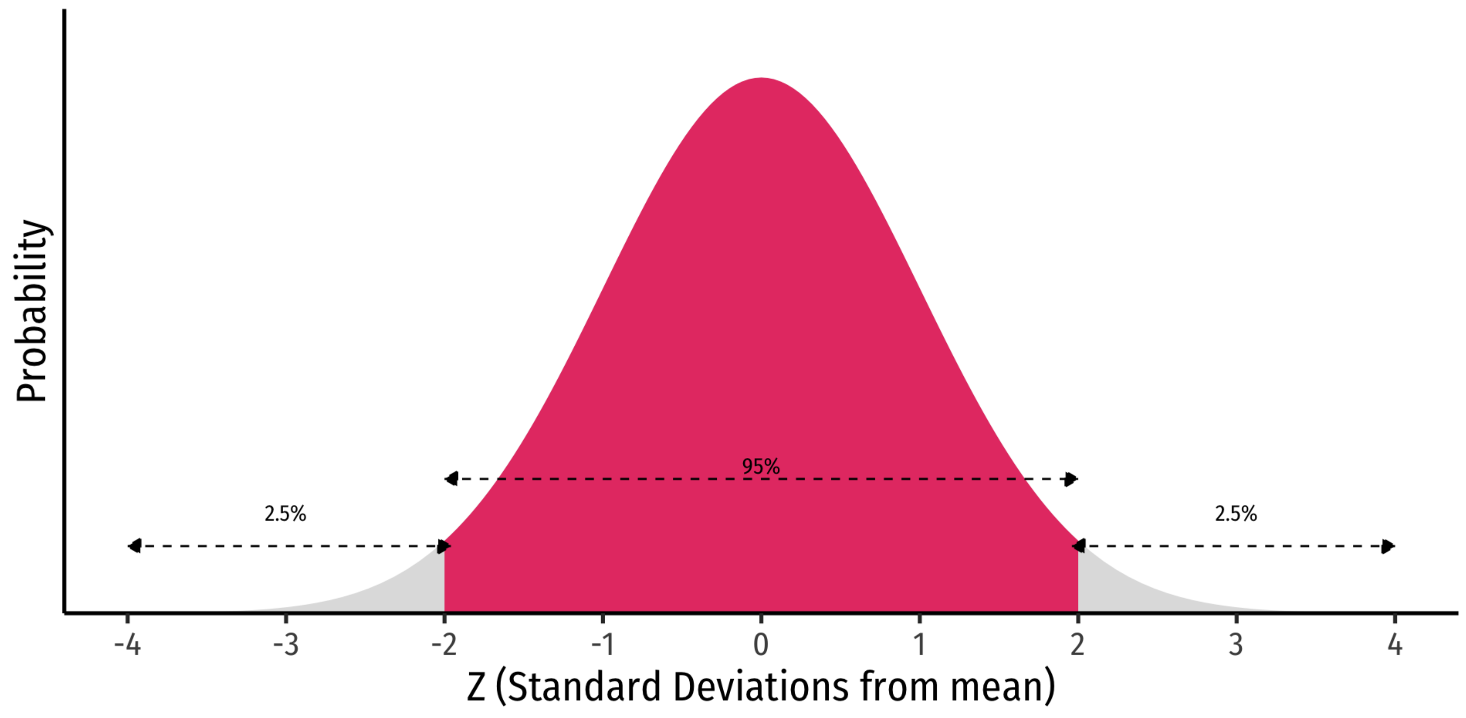
Confidence Levels

- Depending on our confidence level, we are essentially looking for the middle $(1-\alpha)$% of the sampling distribution
- This puts α in the tails; $\frac{\alpha}{2}$ in each tail



Confidence Levels and the Empirical Rule

- Recall the **68-95-99.7% empirical rule** for (standard) normal distributions![1]
- 95% of data falls within 2 standard deviations of the mean
- Thus, in 95% of samples, the true parameter is likely to fall within *about* 2 standard deviations of the sample estimate



1. I'm playing fast and loose here, we can't actually use the normal distribution, we use the Student's t-distribution with n-k-1 degrees of freedom. But there's no need to complicate things you don't need to know about. Look at today's appendix for more.

Interpretting Confidence Intervals

- So our confidence interval for our slope is (-3.22, -1.33), what does this mean again?

❌ 95% of the time, the true effect of class size on test score will be between -3.22 and -1.33

❌ We are 95% confident that a randomly selected school district will have an effect of class size on test score between -3.22 and -1.33

❌ The effect of class size on test score is -2.28 95% of the time.

✅ We are 95% confident that in similarly constructed samples, the true effect is between -3.22 and -1.33

Estimating in R

- `base R` doesn't show confidence intervals in the `lm summary()` output, need the `confint` command

```r
confint(school_reg)
```

```
                2.5 %     97.5 %
(Intercept) 680.32313 717.542779
str          -3.22298  -1.336637
```

Estimating with `broom`

- `broom`'s `tidy()` command can include confidence intervals

```r
school_reg %>%
  tidy(conf.int = TRUE)
```

term <chr>	estimate <dbl>
(Intercept)	698.932952
str	-2.279808

2 rows | 1-2 of 7 columns