2.6 — Statistical Inference

ECON 480 • Econometrics • Fall 2021

Ryan Safner

Assistant Professor of Economics

safner@hood.edu

ryansafner/metricsF21

metricsF21.classes.ryansafner.com

Outline

Why Uncertainty Matters

Confidence Intervals

Confidence Intervals Using the infer Package

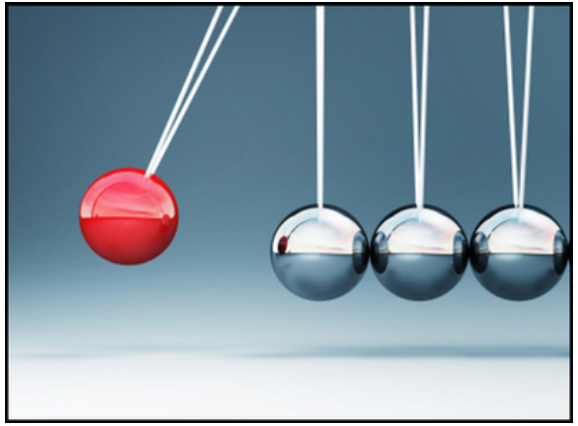
Why Uncertainty Matters

Recall: The Two Big Problems with Data

- We use econometrics to **identify** causal relationships and 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

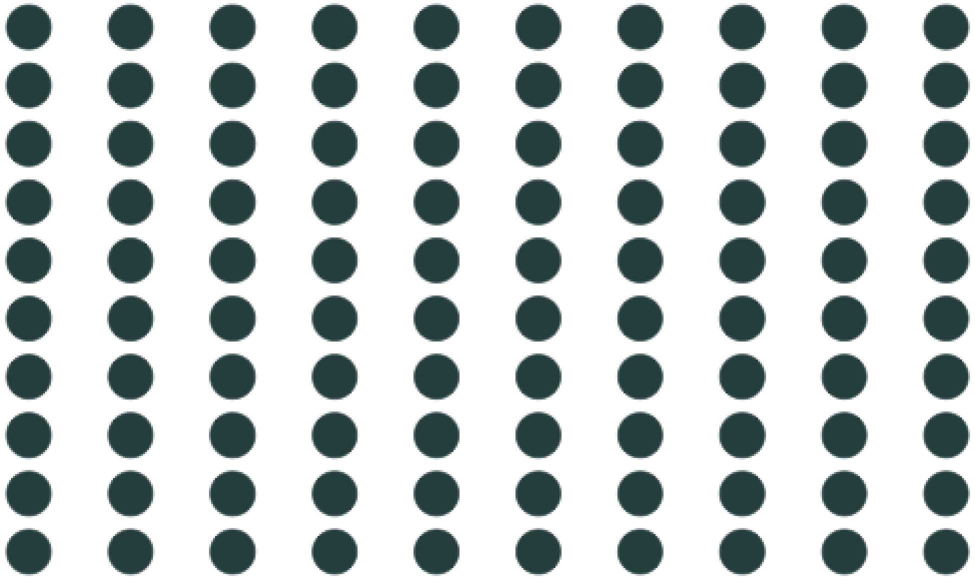
- OLS estimators ($\hat{\beta}_0$ and $\hat{\beta}_1$) are computed from a finite (specific) sample of data
- Our OLS model contains **2 sources of randomness**:
- ***Modeled* randomness**: u includes all factors affecting Y *other* than X
  - different samples will have different values of those other 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

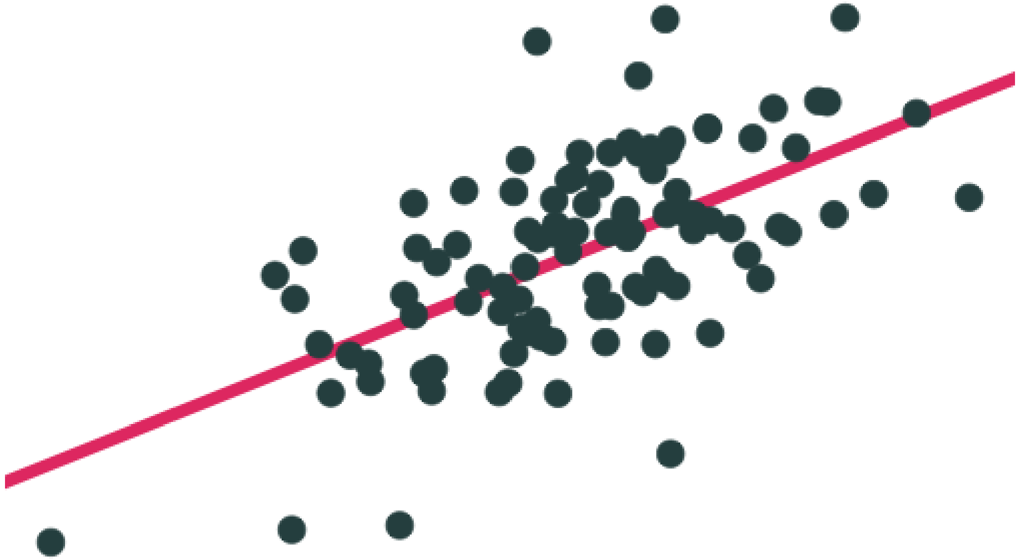
Sample $\xrightarrow{\text{statistical inference}}$ **Population** $\xrightarrow{\text{causal indentification}}$ **Unobserved Parameters**

- 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

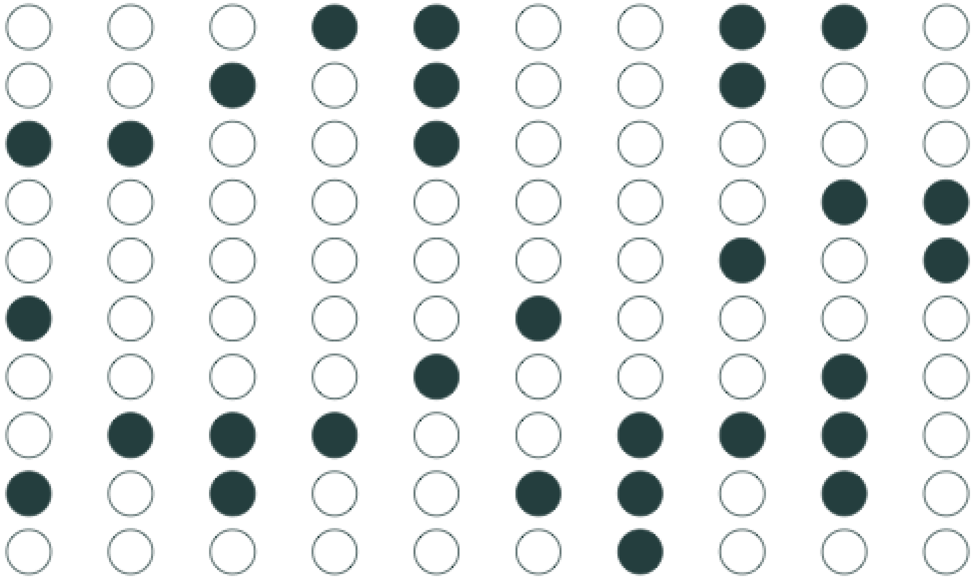


Population relationship

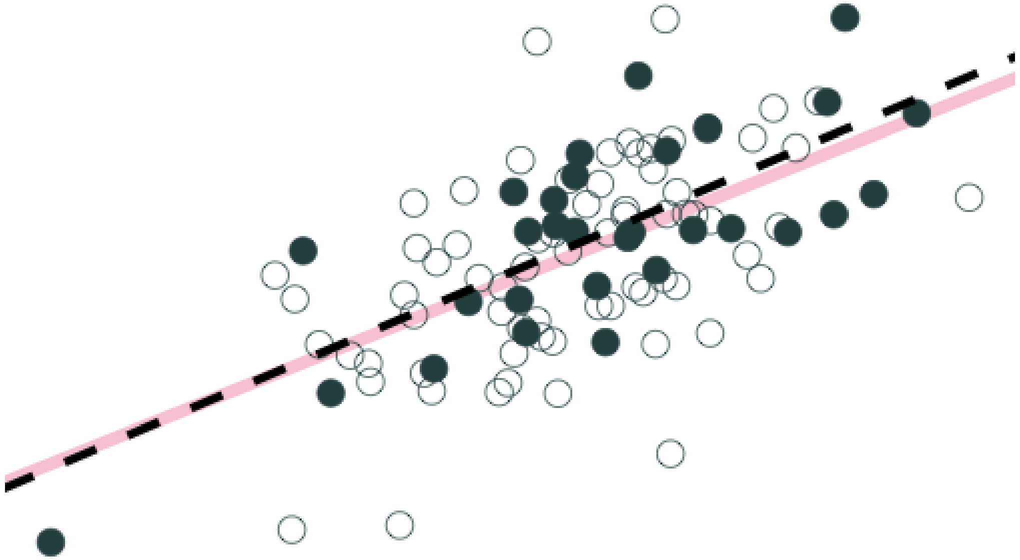
$$Y_i = 3.24 + 0.44X_i + u_i$$

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

Why Sample vs. Population Matters



Sample 1: 30 random individuals



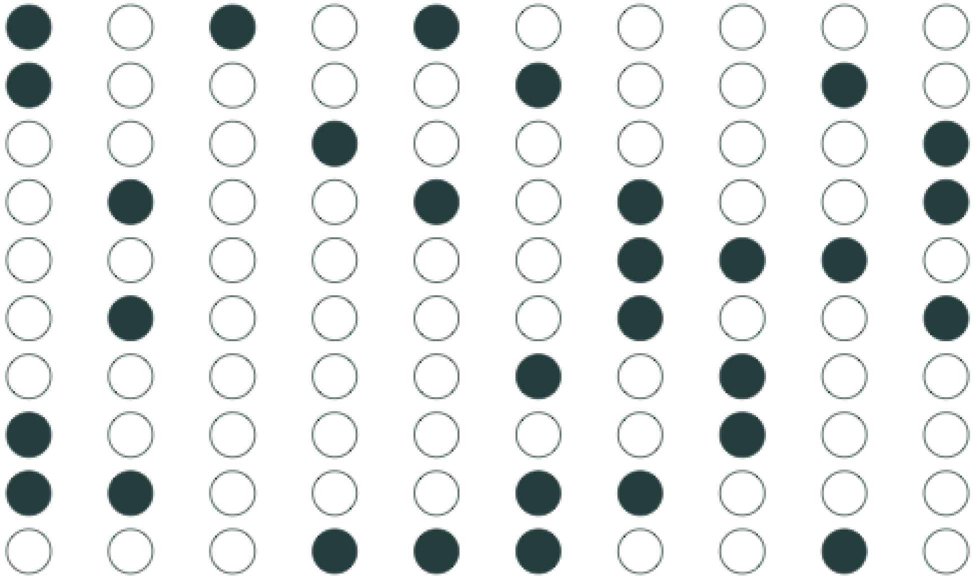
Population relationship

$$Y_i = 3.24 + 0.44X_i + u_i$$

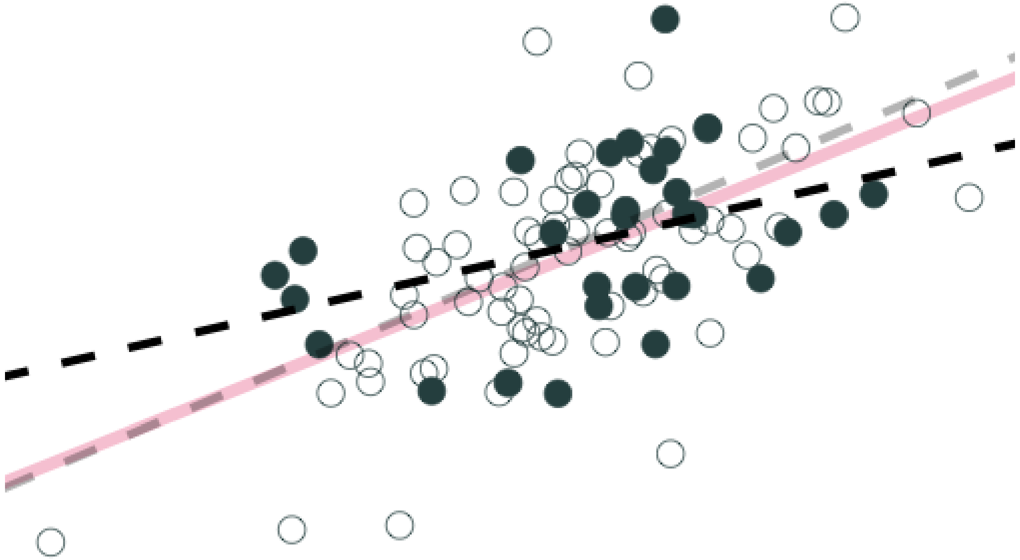
Sample relationship

$$\hat{Y}_i = 3.19 + 0.47X_i$$

Why Sample vs. Population Matters



Sample 2: 30 random individuals



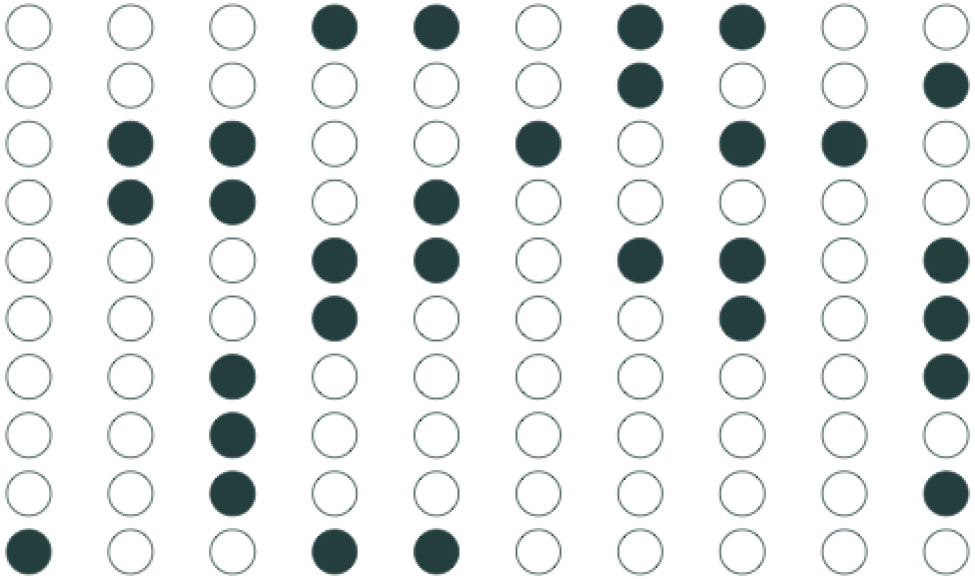
Population relationship

$$Y_i = 3.24 + 0.44X_i + u_i$$

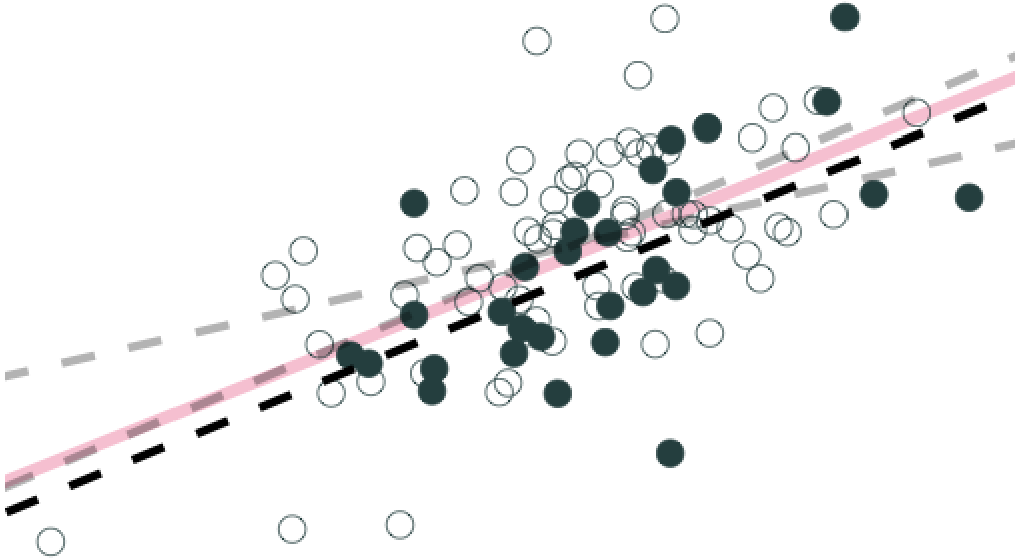
Sample relationship

$$\hat{Y}_i = 4.26 + 0.25X_i$$

Why Sample vs. Population Matters



Sample 3: 30 random individuals



Population relationship

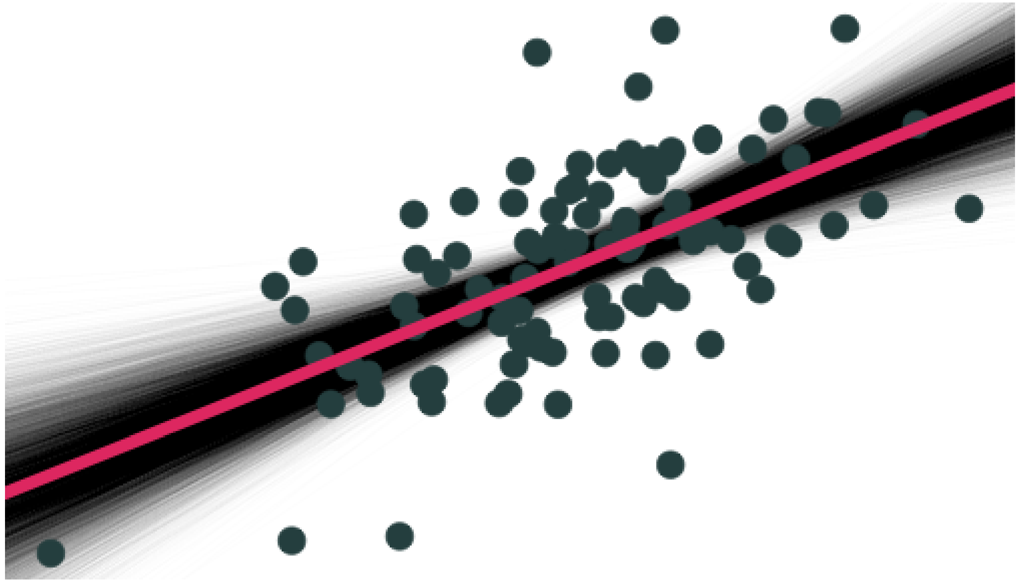
$$Y_i = 3.24 + 0.44X_i + u_i$$

Sample relationship

$$\hat{Y}_i = 2.91 + 0.46X_i$$

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



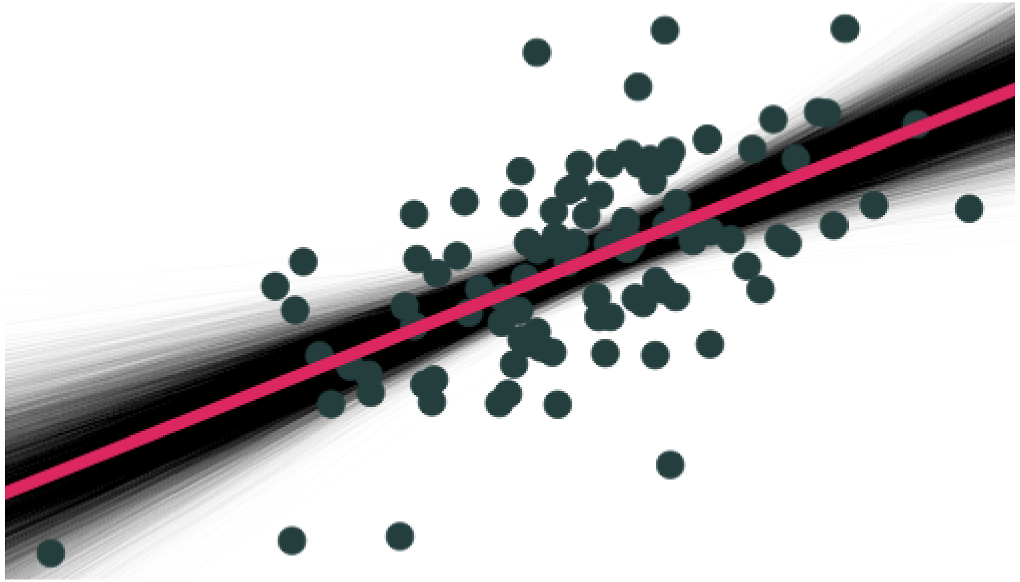
Why Sample vs. Population Matters

- **On average** estimated regression lines from our hypothetical samples provide an unbiased estimate of the true population regression line

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

- However, any *individual line* (any *one* sample) can miss the mark

- This leads to **uncertainty** about our estimated regression line
  - Remember, we only have *one* sample in reality!
  - This is why we care about the **standard error** of our line: $se(\hat{\beta}_1)$!



Confidence Intervals

Statistical Inference

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

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!

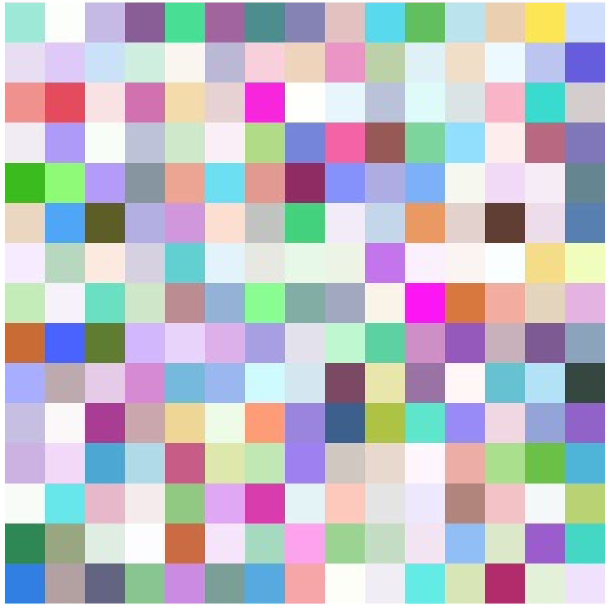
Estimation and Statistical Inference

- 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

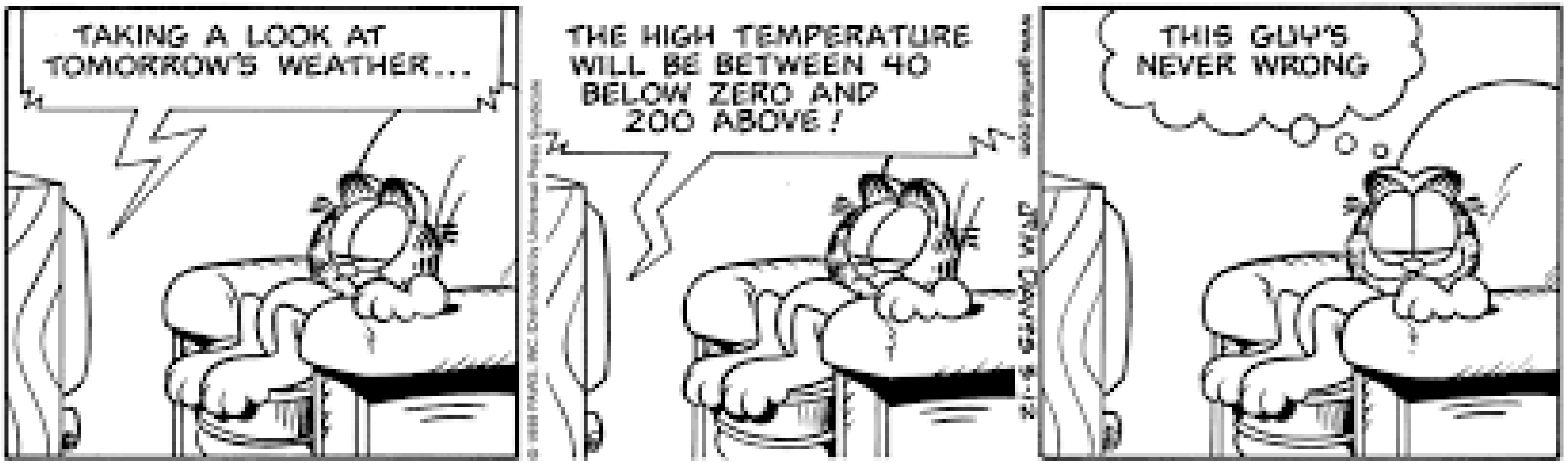
Point estimate

- Use our $\hat{\beta_1}$ and $se(\hat{\beta_1})$ to determine if we have statistically significant evidence to reject a hypothesized β_1

Confidence interval

- Use $\hat{\beta_1}$ and $se(\hat{\beta_1})$ to create an *range* of values that gives us a good chance of capturing the true β_1

Accuracy vs. Precision



- More typical in econometrics to do hypothesis testing (next class)

Generating Confidence Intervals

- We can generate our confidence interval by generating a **"bootstrap"** sampling distribution

- This takes our sample data, and resamples it by selecting random observations with replacement

- 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

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

```
# install first!
install.packages("infer")

# load
library(infer)
```

Confidence Intervals with the infer Package I

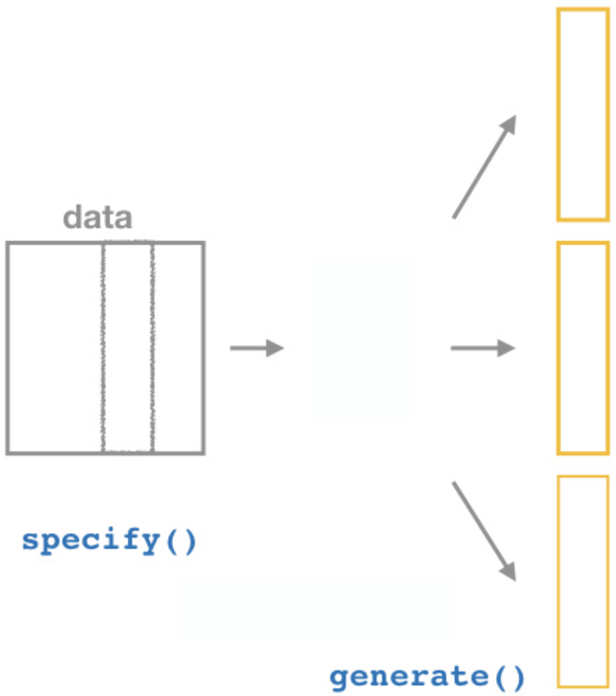
- `infer` allows you to run through these steps manually to understand the process:

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

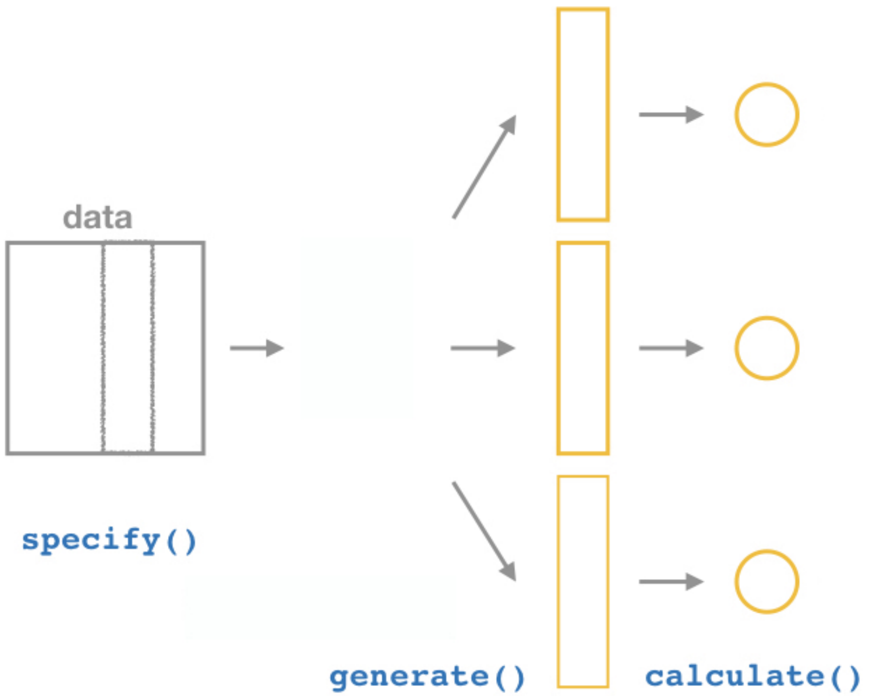
Confidence Intervals with the infer Package II

Confidence Intervals with the infer Package II

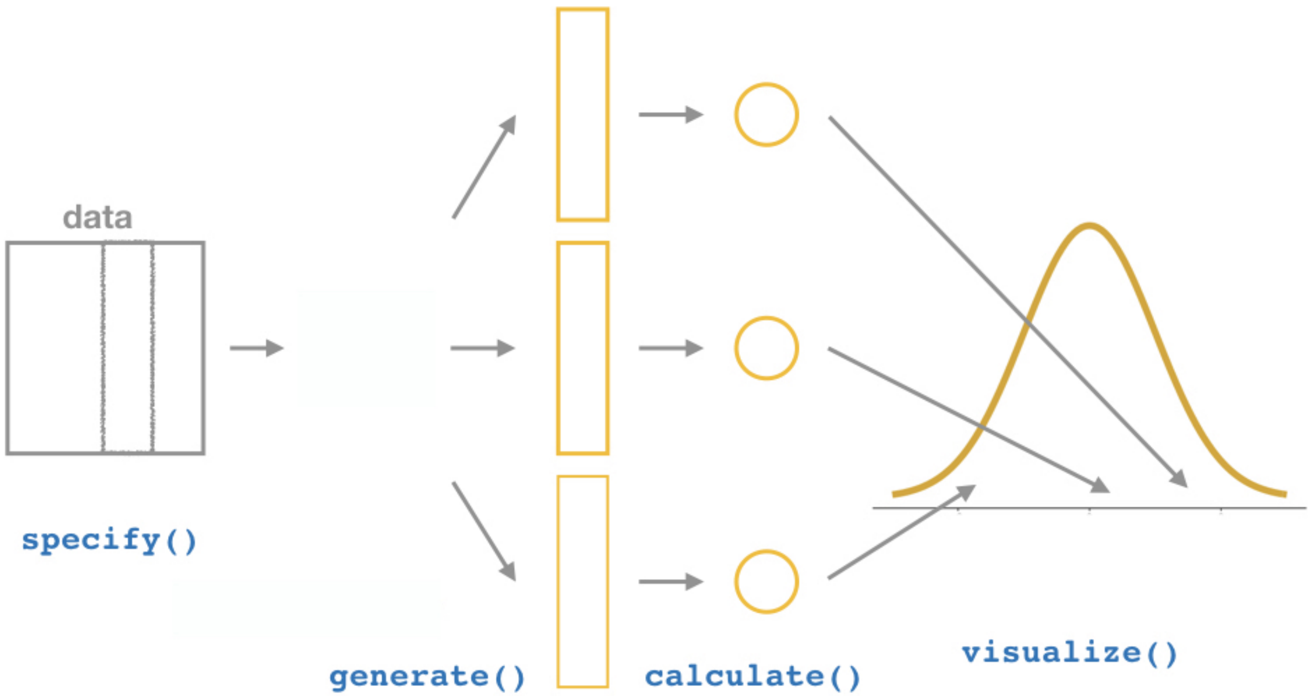
Confidence Intervals with the infer Package II



Confidence Intervals with the infer Package II



Confidence Intervals with the infer Package II



Bootstrapping

Our Sample

term	estimate	std.error	statistic
<chr>	<dbl>	<dbl>	<dbl>
(Intercept)	698.932952	9.4674914	73.824514
str	-2.279808	0.4798256	-4.751327

2 rows | 1-4 of 5 columns

Another "Sample"

term	estimate	std.error	statistic
<chr>	<dbl>	<dbl>	<dbl>
(Intercept)	708.270835	9.5041448	74.522311
str	-2.797334	0.4802065	-5.825274

2 rows | 1-4 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

`specify()`

The *infer* Pipeline: Specify

Specify

`data %>% specify(y ~ x)`

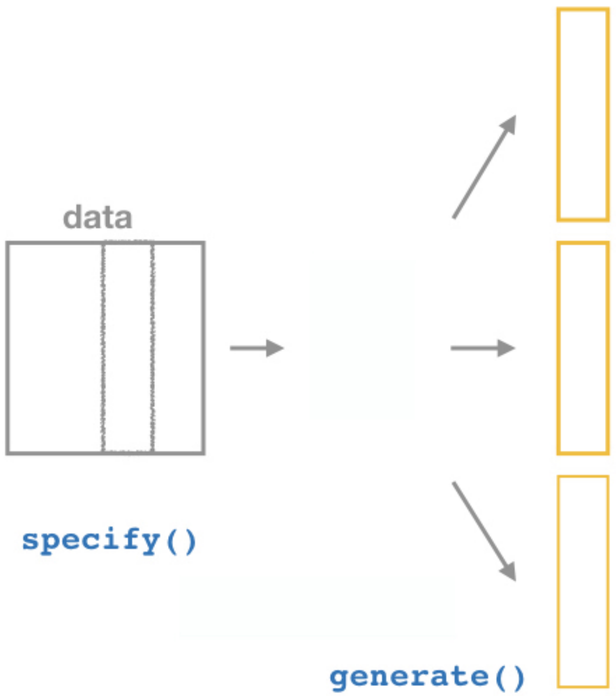
- Take our data and pipe it into the `specify()` function, which is essentially a `lm()` function for regression (for our purposes)

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

testscr <dbl>	str <dbl>
690.80	17.88991
661.20	21.52466
643.60	18.69723
647.70	17.35714
640.85	18.67133

5 rows

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"`

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

The *infer* Pipeline: Generate

Specify

Generate

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

replicate	testscr	str
<int>	<dbl>	<dbl>
1	642.20	19.22221
1	664.15	19.93548
1	671.60	20.34927
1	640.90	19.59016
1	677.25	19.34853
1	672.20	20.20000
1	621.40	22.61905
1	657.00	20.86808
1	664.95	25.80000
1	635.20	17.75499

1-10 of 10,000 rows 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

Specify

Generate

Calculate

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

```
CASchool %>%
  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

Specify

Generate

Calculate

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

replicate <int>	stat <dbl>
1	-3.0370939
2	-2.2228021
3	-2.6601745
4	-3.5696240
5	-2.0007488
6	-2.0979764
7	-1.9015875
8	-2.5362338
9	-2.3061820
10	-1.9369460

1-10 of 1,000 rows Previous **1** 2 3 4 5 6 ... 100 Next

The *infer* Pipeline: Calculate

Specify

Generate

Calculate

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

```r
boot <- CASchool %>% #<< # save this
  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

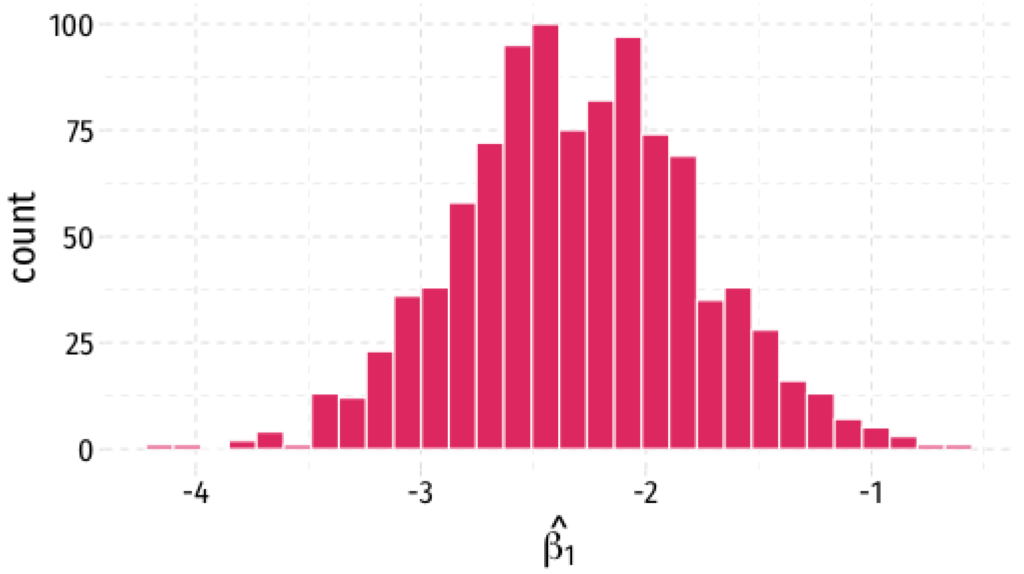
Confidence Interval

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

lower	upper
<dbl>	<dbl>
-3.340545	-1.238815

1 row

```
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 Condens
           base_size=20)
sampling_dist
```



Confidence Interval

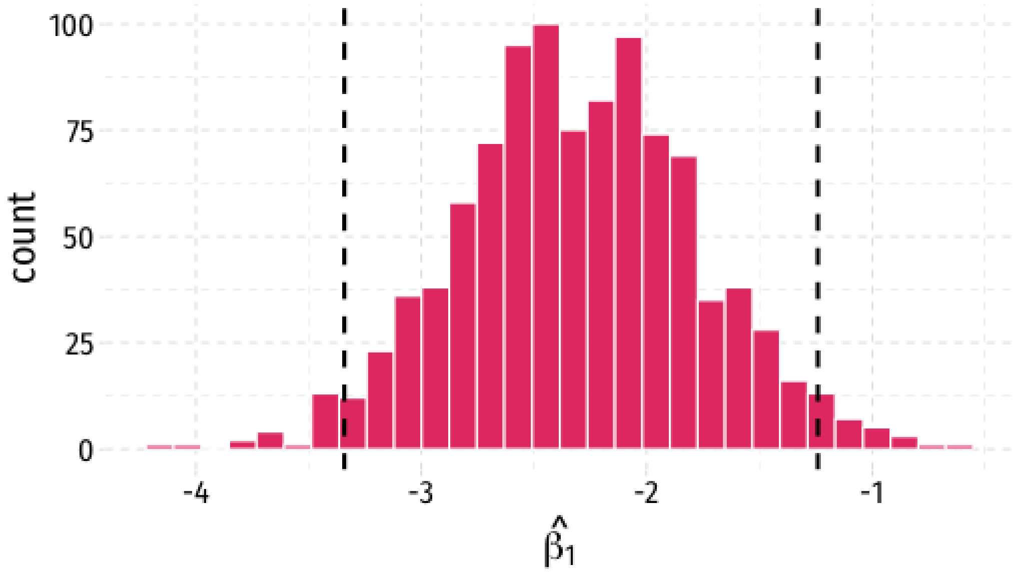
- A confidence interval is the middle 95% of the sampling distribution

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

	lower	upper
	<dbl>	<dbl>
	-3.340545	-1.238815

1 row

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



The *infer* Pipeline: Confidence Interval

Specify

Generate

Calculate

Get Confidence Interval

`%>%` `get_confidence_interval()`

```r
CASchool %>% #<< # 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>
-3.273376	-1.286624

1 row

Broom Can Estimate a Confidence Interval

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

term	estimate	std.error	statistic	p.value	conf.low	conf.high
<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
(Intercept)	698.932952	9.4674914	73.824514	6.569925e-242	680.32313	717.542779
str	-2.279808	0.4798256	-4.751327	2.783307e-06	-3.22298	-1.336637

2 rows

```r
# save and extract confidence interval
our_CI <- tidy_reg %>%
  filter(term == "str") %>%
  select(conf.low, conf.high)

our_CI
```

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

1 row

The *infer* Pipeline: Confidence Interval

Specify

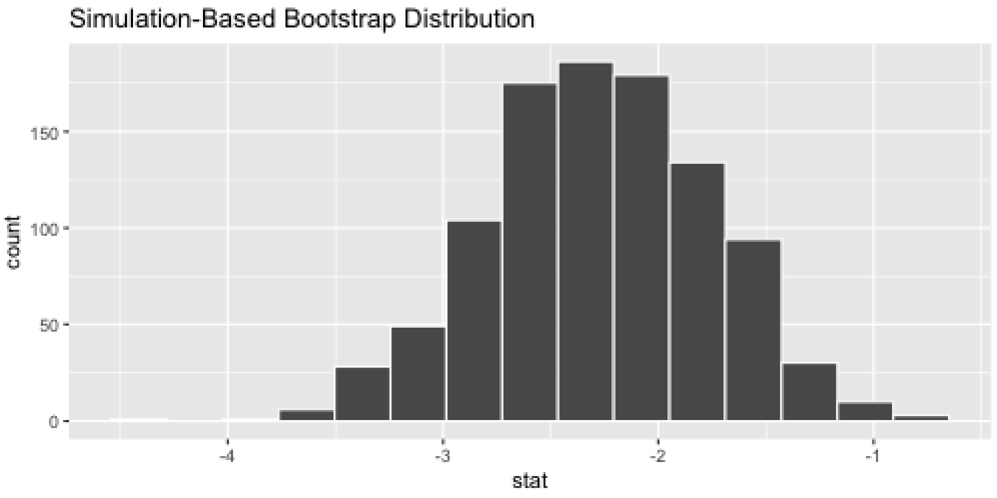
Generate

Calculate

Visualize

`%>% visualize()`

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



- `visualize()` is just a wrapper for `ggplot()`

The *infer* Pipeline: Confidence Interval

Specify

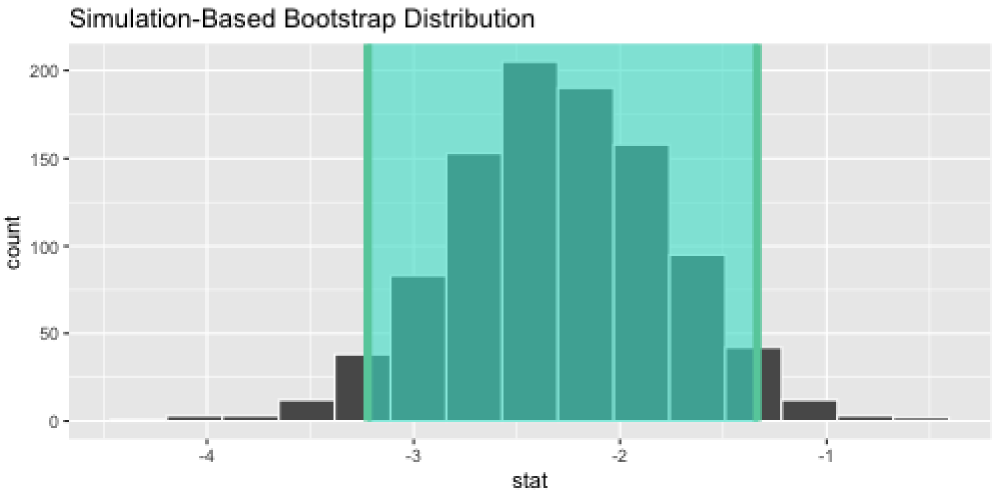
Generate

Calculate

Visualize

`%>% visualize()`

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



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

Confidence Intervals

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

$$\left(\left[\text{estimate - MOE}\right], \left[\text{estimate + MOE}\right]\right)$$

- 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)

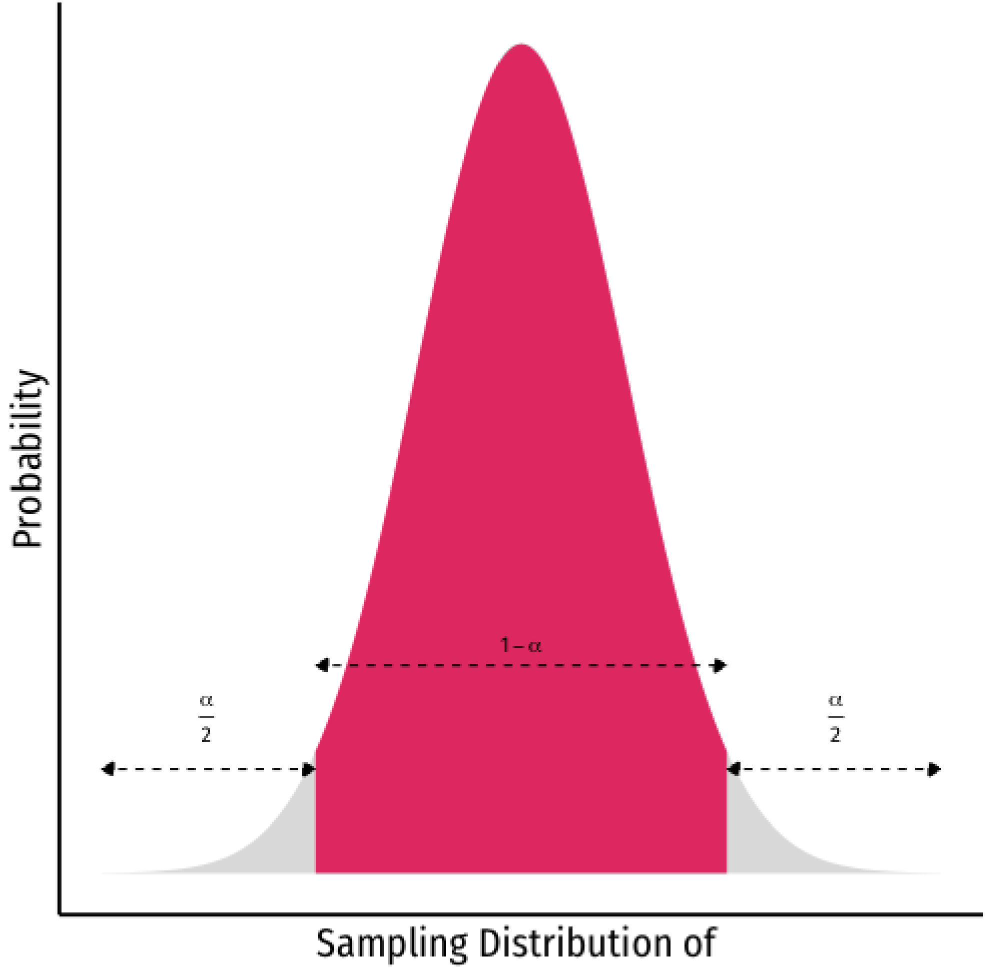
Confidence Intervals

- $(1-\alpha)$ 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

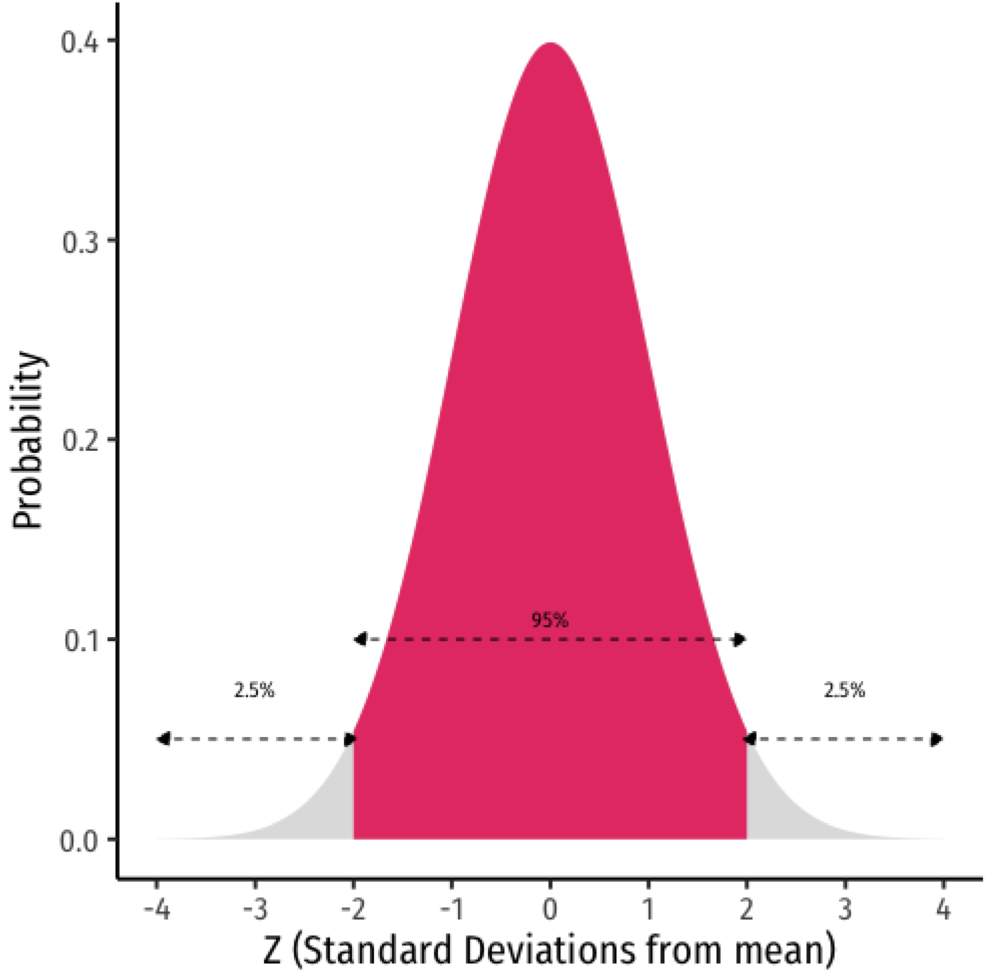


Confidence Levels and the Empirical Rule

- Recall the **68-95-99.7% empirical rule** for (standard) normal distributions![†]

- 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



[†] 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 class notes for more.

Interpreting 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 (Via Regression, rather than `infer`)

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

```
confint(school_reg)
```

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

- `broom` can include confidence intervals

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

term	estimate	std.error
<chr>	<dbl>	<dbl>
(Intercept)	698.932952	9.4674914
str	-2.279808	0.4798256

2 rows | 1-3 of 7 columns