Potential outcomes model, randomized experiments, and power analysis

Tutorial 3

Stanislav Avdeev

Goal for today's tutorial

1. Enumerate tools used to discuss causal questions
2. Set terminology for causal inference
3. Understand different treatment effects
4. Perform power analysis simulations

The fundamental problem of CI

- The main goal we have in doing **causal inference** (CI) is in making as good a guess as possible as to what that Y would have been if X had been different
- That "would have been" is called a **counterfactual** - counter to the fact of what actually happened
    - in doing so, we want to think about two people/firms/countries that are basically exactly the same except that one has $X = 0$ and one has $X = 1$

The fundamental problem of CI

- Suppose there are two variables
  - $Y \in \{0, 1\}$: whether a person is immune to Covid-19
  - $D \in \{0, 1\}$: whether a person gets a vaccine
- Our question: does D cause Y?
- **The fundamental problem of causal inference** (Holland 1986) is that for a given individual, we can only observe one world - either they get the vaccine, or they do not
- What is knowable?
  - first, we need some notation - begin with the **potential outcomes model** (Neyman-Rubin causal model)

Potential outcomes model

- The logic we just went through is the basis of the potential outcomes model, which is one way of thinking about causality
  - we can't observe the counterfactual, and must make an estimate of what the **outcome** would **potentially** have been under the counterfactual
  - figuring out that makes a good counterfactual estimate is a key part of causal inference
- What is the key assumption to make causal inference?
  - **SUTVA** - Stable Unit Treatment Variable Assignment, which states that person $i's$ outcome is only affected by their own treatment
  - How can we ensure SUTVA is satisfied? Randomized experiments are one of the available tools

Randomized experiments

- A common way to do causal inference in many fields is an experiment
  - if you can **randomly assign** D, then you know that the people with $D = 0$ are, on average, exactly the same as the people with $D = 1$
  - then we can easily estimate this model

$$Y_i = \alpha + \delta D_i + U_i$$

- However, when we're working with people/firms/countries, running experiments is often infeasible, impossible, or unethical
- So we have to think hard about a **model** of what the world looks like
  - so we can use this model to figure out what the counterfactual outcome would be (we will discuss that in the 5^{th} and 6^{th} tutorials)

Randomized experiments: simulation

- Let's simulate a dataset with a randomized treatment
- Let's say that getting a treatment D causes Y to increase by 1
- And let's run a randomized experiment of who actually gets D

```
set.seed(7)
df <- tibble(D = sample(c(0, 1), 1000, replace=T),
             Y0 = rnorm(1000),
             Y1 = Y0 + 1,
             Y_observed = ifelse(D == 1, Y1, Y0))
```

```
## # A tibble: 6 × 4
##       D     Y0     Y1 Y_observed
##   <dbl>  <dbl>  <dbl>      <dbl>
## 1     1 -0.206  0.794      0.794
## 2     0 -0.588  0.412     -0.588
## 3     0 -0.685  0.315     -0.685
## 4     1  1.00   2.00       2.00
## 5     0 -0.773  0.227     -0.773
## 6     1 -1.99  -0.994     -0.994
```

Randomized experiments: simulation

```r
# The true effect is 1
df %>% group_by(D) %>% summarize(Y = mean(Y_observed))
```

```
## # A tibble: 2 × 2
##       D       Y
##   <dbl>   <dbl>
## 1     0 -0.0326
## 2     1  0.976
```

```r
random ← lm(Y_observed ~ D, df) # we can also use a simple regression
```

	Model 1
(Intercept)	-0.033
	(0.045)
D	1.008***
	(0.063)
Num.Obs.	1000
+ p < 0.1, * p < 0.05, ** p < 0.01, *** p < 0.001	

Randomized experiments: simulation

- Now this time we can't randomize D

```
set.seed(7)
df <- tibble(Z = runif(1000),
             D = ifelse(Z > 0.7, 1, 0),
             Y0 = rnorm(1000) + Z,
             Y1 = Y0 + 1,
             Y_observed = ifelse(D == 1, Y1, Y0))
```

```
## # A tibble: 6 × 5
##         Z     D     Y0     Y1 Y_observed
##     <dbl> <dbl>  <dbl>  <dbl>      <dbl>
## 1 0.989       1  0.783  1.78       1.78
## 2 0.398       0 -0.190  0.810     -0.190
## 3 0.116       0 -0.570  0.430     -0.570
## 4 0.0697      0  1.07   2.07       1.07
## 5 0.244       0 -0.529  0.471     -0.529
## 6 0.792       1 -1.20  -0.202     -0.202
```

Randomized experiments: simulation

```
# The true effect is 1
df %>% group_by(D) %>% summarize(Y = mean(Y_observed))
```

```
## # A tibble: 2 × 2
##       D     Y
##   <dbl> <dbl>
## 1     0 0.309
## 2     1 1.85
```

```
not_random ← lm(Y_observed ~ D, df)
```

	Model 1
(Intercept)	0.309***
	(0.038)
D	1.540***
	(0.068)
Num.Obs.	1000
+ p < 0.1, * p < 0.05, ** p < 0.01, *** p < 0.001	

Randomized experiments: simulation

- But if we properly **model** the process and compare apples to apples

```
# The true effect is 1
not_random_modeled ← lm(Y_observed ~ D, df %>%
                           filter(abs(Z - 0.7) < 0.01)) # looks like RDD
```

	Model 1
(Intercept)	0.829+
	(0.429)
D	1.084+
	(0.575)
Num.Obs.	18

+ p < 0.1, * p < 0.05, ** p < 0.01, *** p < 0.001

Identification

- In the first randomized case `lm(Y ~ D, df)` identifies the causal effect of X on Y
  - in other words, when we see the estimate, we can claim that it's isolating just the causal effect
- In the second non-randomized case `lm(Y ~ D, df)` does not identify the causal effect
- In the apples-to-apples comparison we could identify the causal effect (practically by using RDD)
- Causal inference is all about figuring out what calculation we need to do to identify the effects
  - but what effects are we identifying?

Treatment effects

- For any given treatment, there are likely to be **many treatment effects**
- Average Treatment Effect

$$ATE = \mathbb{E}(Y_1^* - Y_0^*) = \mathbb{E}(Y_1^*) - \mathbb{E}(Y_0^*)$$

  - ATE is the effect for the **full** population
- Average Treatment Effect on the Treated

$$ATET = \mathbb{E}(Y_1^* - Y_0^* | D = 1) = \mathbb{E}(Y_1^* | D = 1) - \mathbb{E}(Y_0^* | D = 1)$$

  - ATET is the effect for individuals who actually **received** the treatment
- Heterogeneous Treatment Effect

$$ATE(X) = \mathbb{E}(Y_1^* - Y_0^* | X) = \mathbb{E}(Y_1^* | X) - \mathbb{E}(Y_0^* | X)$$

  - ATE(X) is the effect that is different for individuals with **different** characteristics

Treatment effects

- What we get depends on the research design itself as well as the estimator we use to perform that design
- Which average you'd want depends on what you'd want to do with it
  - want to know how effective a treatment would be if applied to **everyone/at random**? ATE
  - want to know how effective a treatment **was** when it was applied? ATET
  - want to know how effective a treatment **was** when it was applied for males? ATE(X)
  - want to know how effective a treatment would be if applied **just a little more broadly?** Local Average Treatment Effect - LATE (next lecture)
- Different treatment effects aren't wrong but we need to pay attention to which one we're getting, or else we may apply the result incorrectly
  - a result could end up representing a different group than you're really interested in

Treatment effects: simulation

- Let's simulate some data and see what different methods give us
- We'll start with some basic data where the effect is already identified

```
set.seed(7)
df <- tibble(group = sample(c('A', 'B'), 1000, replace = TRUE),
             W = rnorm(1000, mean = 0, sd = sqrt(8)),
             b = case_when(group == 'A' ~ rnorm(1000, mean = 5, sd = 2),
                           group == 'B' ~ rnorm(1000, mean = 7, sd = 2)),
             X = rnorm(1000),
             Y = b*X + rnorm(1000))
```

```
## # A tibble: 6 × 5
##   group      W     b      X      Y
##   <chr>  <dbl> <dbl>  <dbl>  <dbl>
## 1 B     -0.583  7.52 -1.22  -8.51
## 2 A     -1.66   4.59  2.46  11.3
## 3 A     -1.94   4.59 -0.186 -0.581
## 4 B      2.84  10.4   1.64  18.1
## 5 A     -2.19   5.92 -0.958 -6.69
## 6 B     -5.64   4.68 -1.02  -6.39
```

Treatment effects: simulation

- The true effect for group A is 5, for B is 7

```r
m1 ← lm(Y ~ X, data = df)
m2 ← lm(Y ~ X*group, data = df)
m3 ← lm(Y ~ X, data = df[df$group == 'A',])
m4 ← lm(Y ~ X, data = df[df$group == 'B',])
```

	Model 1	**Model 2**	**Model 3**	**Model 4**
X	6.035***	5.080***	5.080***	7.023***
	(0.077)	(0.099)	(0.098)	(0.101)
groupB		-0.003		
		(0.140)		
X × groupB		1.944***		
		(0.141)		
Num.Obs.	1000	1000	484	516

+ p < 0.1, * p < 0.05, ** p < 0.01, *** p < 0.001

Performing an experiment

1. Figure out what needs to be randomized (the treatment, usually)
2. Figure out what you want your outcome to be
3. **Figure out where you'll do the experiment and how much data you need**
4. Figure out how to randomize the treatment
5. Perform the experiment and collect data
6. Check balance
7. Analyze the data
8. Check for threats to the experiment

Power analysis

- In experiments, we have some control over our sample size
    - so **before** collecting any data, we need to do a power analysis - what sample size do we need to get our standard errors down to a useful level?
- Power analysis also applies to observational data/non-experimental data
    - we just don't do it as often because we can't control the sample size anyway, and it's easier to get huge samples
- You need to have a huge sample to reasonably study small effects
    - so don't pursue effects that are likely to be really tiny, or at least tinier than your sample can handle
    - if we ran the underpowered study anyway and **do** get a significant result, it would be more likely to be a false positive than a true positive. That's **low power**

Power analysis

- Using D as shorthand for the treatment and Y as shorthand for the outcome, assuming we're doing a power analysis for a study of the relationship between D and Y
- Power analysis balances five things
    1. size of the effect (coefficient in a regression, a correlation, etc.)
    2. amount of variation in the treatment (the variance of D, say)
    3. amount of other variation in Y (the R^2, or the variation from the residual after explaining Y with D, etc.)
    4. power (the standard error of the estimate, statistical power, i.e. the true-positive rate)
    5. sample size

Power analysis: implementation

- In order to do power analysis, you need to be able to fill in the values for four of those five pieces, so that power analysis can tell you the fifth one
  - use previous research results about these values
  - requirement of the NGO, municipality, etc.
  - cost-benefit analysis
- Use standard practice for statistical power
  - a goal is to achieve $80\% - 90\%$ statistical power
- You can use a standard formula to calculate your outcomes

$$\text{MDE} = (t_{1-\alpha/2} - t_{1-q})\sqrt{\frac{1}{p(1-p)}}\sqrt{\frac{\sigma^2}{n}}$$

- Empirically you can do power analysis using
  - `power.t.test()` in **stats**
  - Multiple functions in **powerMediation**
  - Simulations

Power analysis: simulation

- The idea of a power analysis is:
  - to have data with certain properties (variance of X, size of the effect, etc.)
  - to use certain analytic methods (regression, etc.)
  - to make some claims about the sampling variation of the estimator (statistical power, size of standard errors, etc.)
- That's what simulations allow you to do

Power analysis: simulation

- Let's perform power analysis with simulated datasets

```
set.seed(777)
# create vectors to store your estimates
coef_results ← c()
sig_results ← c()

for (i in 1:500) {
  # have to re-create the data every time or it will just be the same data
  df ← tibble(D = runif(1000, 0, 1),
              Y = 0.9*D + rnorm(1000, mean = 0, sd = 3))

  # Run the analysis
  model ← feols(Y ~ D, df, se = 'hetero')

  # Get the results
  coef_results[i] ← coef(model)[2]
  sig_results[i] ← tidy(model)$p.value[2] ⩽ .05
}
```

Power analysis: simulation

- Our estimate of statistical power is the proportion of the results that are significant:

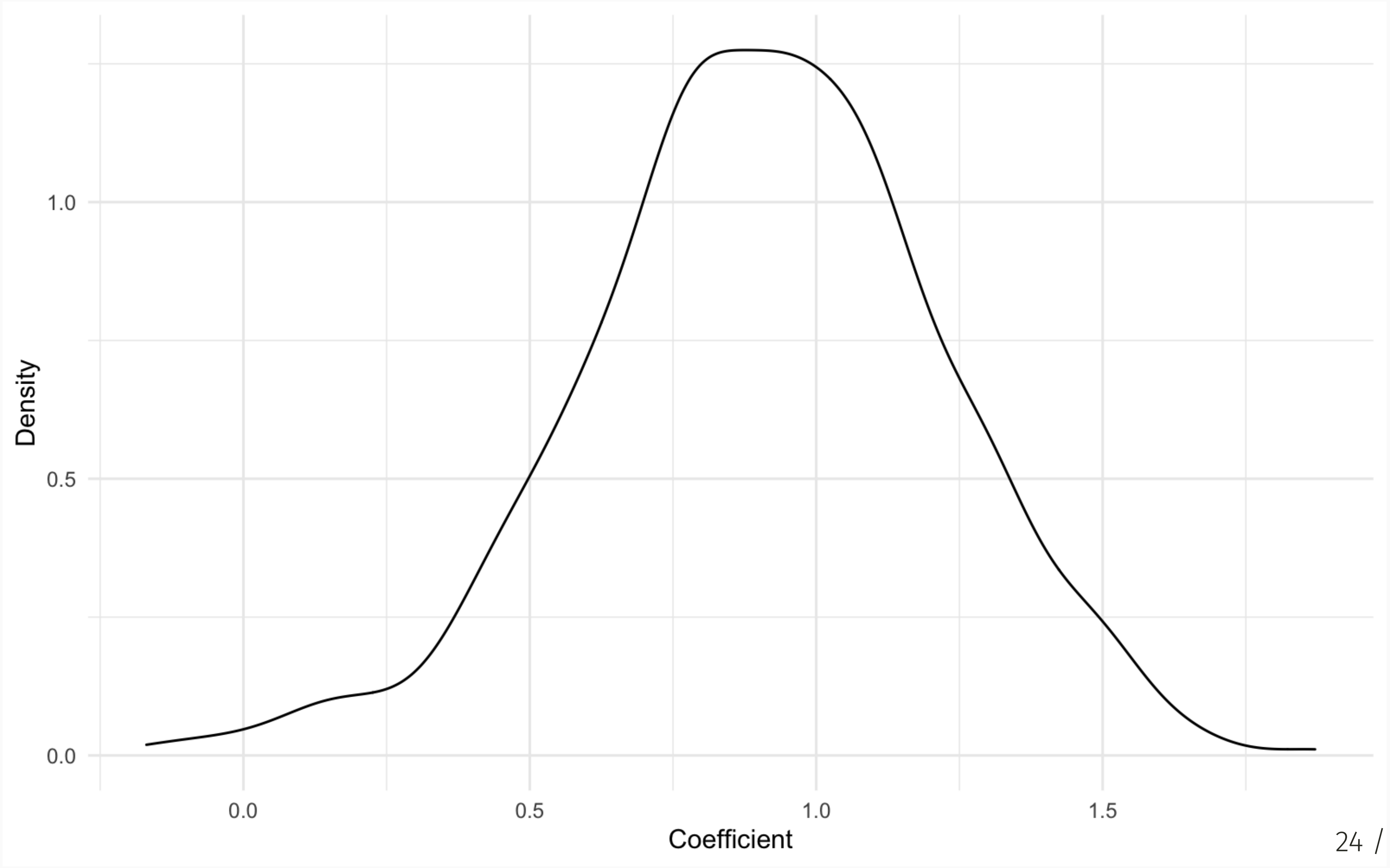
```
mean(sig_results)
```

```
## [1] 0.812
```

- So we have statistical power of 81.20%
- We might also want to look at the distribution of the coefficient itself
- The standard deviation of the coefficient across all the simulated runs gives you a good idea of what the standard error of the coefficient will be (`sd(coef_results)`, which gives us $\hat{\sigma}_\beta = 0.31$).

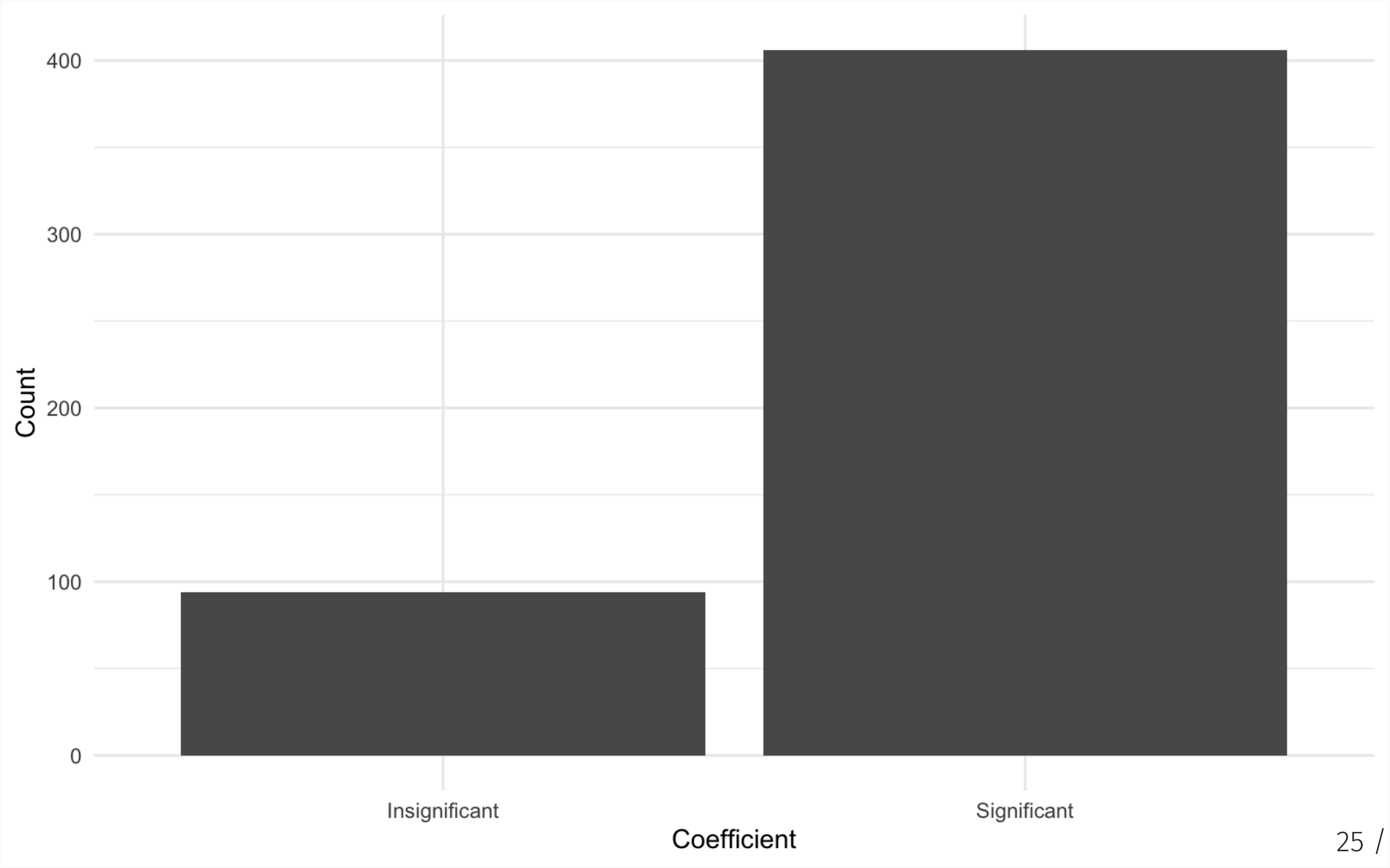
Power analysis: simulation

- Check the distribution of the effects. The true effect is 0.8



Power analysis: simulation

- Check the distribution of the significance levels



Power analysis: simulation

- The two main goals of power analysis is to calculate
  - **minimum detectable effect**
  - **smallest sample size** for a given power level
- How can we do that here?
  - by trying different values of effect size and sample size and seeing what we get
  - to do this, we're first going to take everything we've done so far and put it inside a `function()` that we can call

Power analysis: simulation

Let's create a function t calculate power analysis with a given effect and sample sizes

```r
my_power_function <- function(effect, sample_size) {
  sig_results <- c()

  for (i in 1:500) {
    # Have to re-create the data EVERY TIME or it will just be the same data
    df <- tibble(D = runif(sample_size, 0, 1),
                 Y = effect*D + rnorm(sample_size, mean = 0, sd = 3))

    # Run the analysis
    model <- feols(Y ~ D, data = df, se = 'hetero')

    # Get the results
    sig_results[i] <- tidy(model)$p.value[2] <= .05
  }

  sig_results %>%
    mean() %>%
    return()
}
```

Power analysis: simulation

- Now we can just call the function, setting `effect` and `sample_size` to whatever we want, and get the power back

```
my_power_function(.9, 1000)
```

```
## [1] 0.784
```

- Seems good
- If you are running repeated simulations (e.g., for a power calculation), make sure you never use `set.seed()` inside of a `function()` that creates random numbers, or the function will always give you the exact same numbers

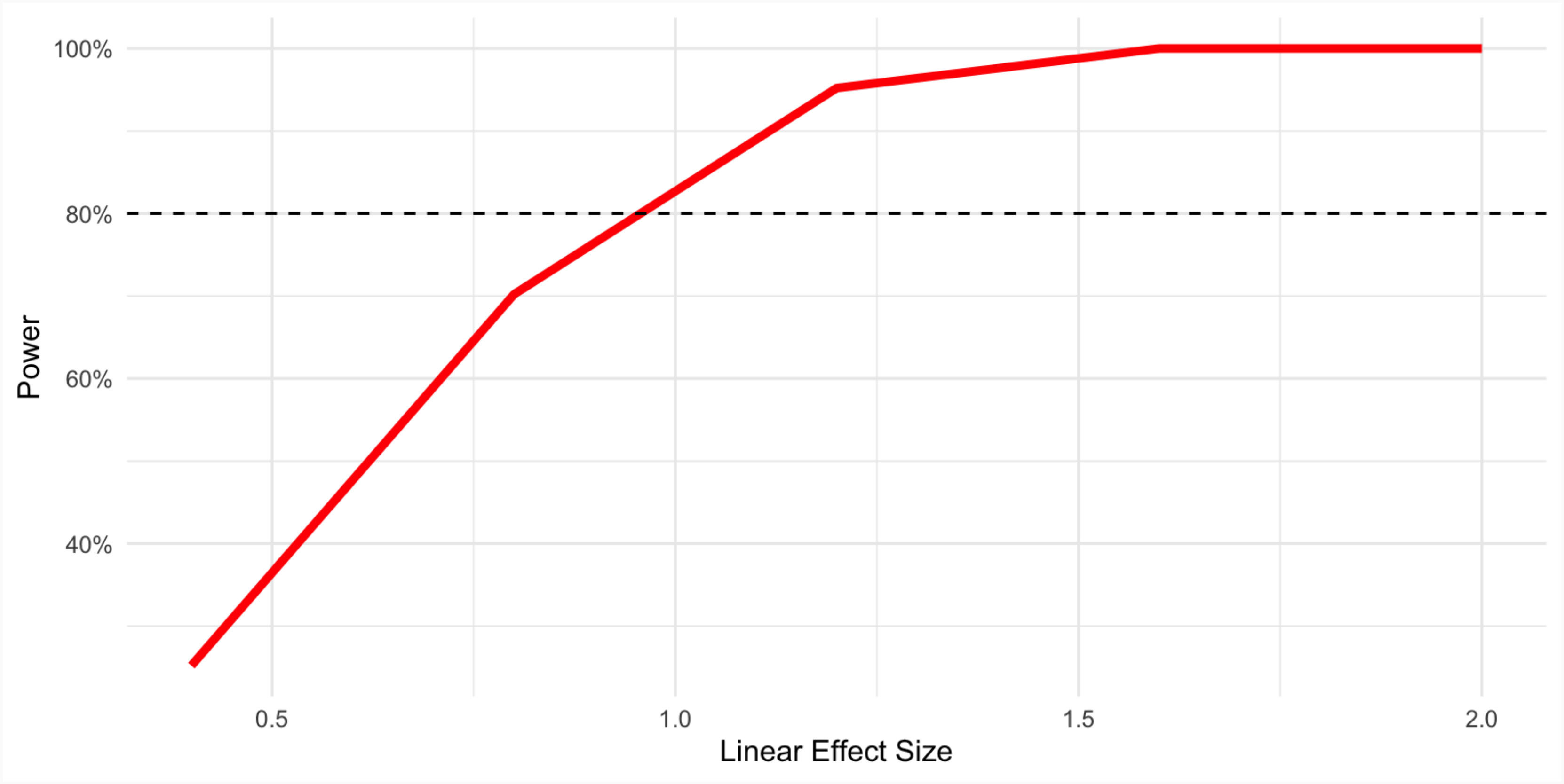
Power analysis: simulation

- Now let's say we really are stuck with a sample size of 1000 and we want to know the **minimum detectable effect** size we can get a power of 0.8 with
- To do this, we can just run our function with `sample_size = 1000` and a bunch of different `effect` values until we get back a power above 0.8

```r
power_levels ← c()
effects_to_try ← c(0.4, 0.8, 1.2, 1.6, 2)
for (i in 1:5) {
  power_levels[i] ← my_power_function(effects_to_try[i], 1000)
}
power_results ← tibble(effect = effects_to_try,
                        power = power_levels)
power_results # Where do we cross 80%?
```

```
## # A tibble: 5 × 2
##   effect power
##    <dbl> <dbl>
## 1    0.4 0.252
## 2    0.8 0.702
## 3    1.2 0.952
## 4    1.6 1
## 5    2   1
```

Power analysis: simulation



- So it looks like we need an effect somewhere between 0.8 and 1.2 to have an 80% chance of finding a significant result
- If we don't think the effect is actually likely to be that large, then we need to figure out something else to do - find a bigger sample, use a more precise estimation method

References

Books

- Huntington-Klein, N. The Effect: An Introduction to Research Design and Causality, Chapter 10: Treatment Effects
- Cunningham, S. Causal Inference: The Mixtape, Chapter 4: Potential Outcomes Causal Model

Slides

- Huntington-Klein, N. Econometrics Course Slides, Week 8: Experiments
- Huntington-Klein, N. Causality Inference Course Slides, Lecture 3: Causality and Lecture 18: Treatment Effects
- Goldsmith-Pinkham P. Applied Empirical Methods Course, Week 1: Potential Outcomes and Directed Acylic Graphs