

# WU #7 - Simulating CIs

Math 154 - Jo Hardin

Thursday, September 23, 2021

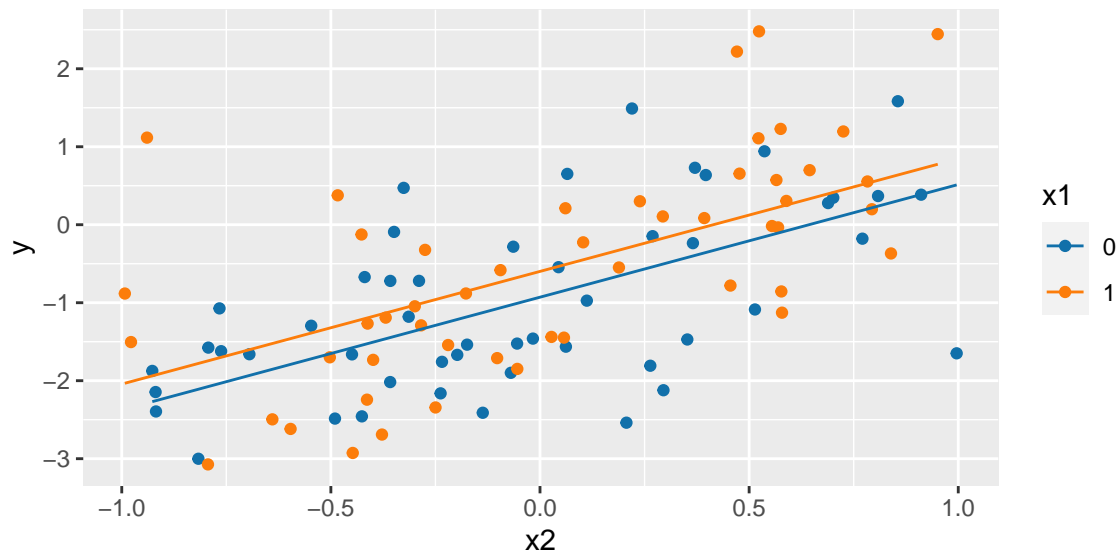
Name: \_\_\_\_\_

Let's say that you want to check the *mathematical formula* for creating CIs for the slope coefficient in a linear model. In particular, you want to know if the procedure you use actually does capture the population parameter 95% of the time. You are simulating from the following population model:

$$Y = -1 + 0.5X_1 + 1.5X_2 + \epsilon, \quad \epsilon \sim N(0,1)$$

Consider:

```
n_obs <- 100
x1 <- rep(c(0,1), each=n_obs/2)
x2 <- runif(n_obs, min=-1, max=1)
beta0 <- -1; beta1 <- 0.5; beta2 <- 1.5
y <- beta0 + beta1*x1 + beta2*x2 + rnorm(n_obs, 0,1)
```



```
## # A tibble: 3 x 7
##   term      estimate std.error statistic  p.value conf.low conf.high
##   <chr>      <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
## 1 (Intercept) -0.929    0.142    -6.52 3.10e- 9  -1.21   -0.647
## 2 x1          0.330    0.202     1.64 1.05e- 1  -0.0698  0.731
## 3 x2          1.44    0.192     7.53 2.66e-11   1.06    1.83
```

Note that the **estimated** model (estimated from the single observed dataset) is:

$$\hat{Y} = -0.929 + 0.33 \cdot X_1 + 1.445 \cdot X_2$$

Write down the steps for the simulation that would assess whether the CI procedure is valid for the parameter on  $X_2$ . (No R code here, use words to explain / enumerate the process.)

**Solution:**

1. Note the data / population above and also the model you are working with. Your results will be based on those data and that model. Sometimes it will be straightforward to generalize, sometimes it won't.
2. Take a sample from the population you've set, and create a linear model.
3. Use the linear model created in 2. to find a CI for the  $\beta_2$  slope parameter.
4. You know the truth! So check to see if  $\beta_2$  is in the CI created in step 3.
5. Repeat steps 2 - 4 many times. Find the rate at which your CI captures the true parameter. If your interval in 3 is constructed as a 95% interval, you should be capturing  $\beta_2$  about 95% of the time. (And furthermore, we'd expect  $\beta_2$  to be lower than the interval 2.5% of the time and higher than the interval 2.5% of the time.)

**R code:** The gist of the idea for a single simulated CI. Notice that we *do* capture the true parameter  $\beta_2$  in the confidence interval.

```
CI <- lm(y~x1+x2) %>% tidy(conf.int=TRUE) %>% data.frame()
```

```
CI
```

```
##           term estimate std.error statistic p.value conf.low
## 1 (Intercept)  -0.93      0.14      -6.5 3.1e-09   -1.21
## 2           x1   0.33      0.20       1.6 1.0e-01   -0.07
## 3           x2   1.44      0.19       7.5 2.7e-11    1.06
##   conf.high
## 1    -0.65
## 2     0.73
## 3     1.83
```

```
between(beta2, CI[3,6], CI[3,7])
```

```
## [1] TRUE
```

A tidy way to run the code (the fewer for-loops, the faster your code will be). Note that the first thing to do is to create all 1000 datasets and put them into a tidy dataframe where one of the columns is a **column of dataframes**!

```
set.seed(74)
reps <- 1000
lin_data <-
  data.frame(row_id = seq(1, n_obs, 1)) %>%
  slice(rep(row_id, each = reps)) %>%
  mutate(
    sim_id = rep(1:reps, n_obs),
    x1 = rep(c(0,1), each = n()/2),
    x2 = runif(n(), min = -1, max = 1),
    y = beta0 + beta1*x1 + beta2*x2 + rnorm(n(), mean = 0, sd = 1)
  ) %>%
  arrange(sim_id, row_id) %>%
  group_by(sim_id) %>%
  nest()

lin_data
```

```
## # A tibble: 1,000 x 2
## # Groups:   sim_id [1,000]
##   sim_id data
##   <int> <list>
## 1     1 <tibble [100 x 4]>
## 2     2 <tibble [100 x 4]>
## 3     3 <tibble [100 x 4]>
## 4     4 <tibble [100 x 4]>
## 5     5 <tibble [100 x 4]>
## 6     6 <tibble [100 x 4]>
## 7     7 <tibble [100 x 4]>
## 8     8 <tibble [100 x 4]>
## 9     9 <tibble [100 x 4]>
## 10    10 <tibble [100 x 4]>
## # ... with 990 more rows
```

```

# the linear model function
beta_coef <- function(df){
  lm(y ~ x1 + x2, data = df) %>%
    tidy(conf.int = TRUE) %>%
    filter(term == "x2") %>%
    select(estimate, conf.low, conf.high)
}

lin_data %>%
  mutate(b2_vals = map(data, beta_coef)) %>%
  select(b2_vals) %>%
  unnest() %>%
  summarize(capture = between(beta2, conf.low, conf.high)) %>%
  summarize(capture_rate = sum(capture)/reps)

```

```

## # A tibble: 1 x 1
##   capture_rate
##         <dbl>
## 1         0.945

```

Note that our capture rate was 94.5%. Pretty good!

If you prefer to think about for loops, a for-loop that repeats the process 1000 times.

```

beta2.in <- c()
beta0 <- -1; beta1 <- 0.5; beta2 <- 1.5
n_obs<- 100; reps=1000

set.seed(4747)
for(i in 1:reps){
  x1 <- rep(c(0,1), each=n_obs/2)
  x2 <- runif(n_obs, min=-1, max=1)
  y <- beta0 + beta1*x1 + beta2*x2 + rnorm(n_obs, mean=0, sd = 1)

  CI <- lm(y~x1+x2) %>% tidy(conf.int=TRUE) %>% data.frame()

  beta2.in <- c(beta2.in, between(beta2, CI[3,6], CI[3,7]))
}
sum(beta2.in)/reps

```

```
## [1] 0.94
```