# Bootstrap

Dancun Juma

2025-03-12

## Task 1: Done

## Task 2: Load the necessary packages

```r
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```r
library(tidymodels)
```

```
## -- Attaching packages -------------------------------------- tidymodels 1.2.0 --
## v broom        1.0.6     v rsample      1.2.1
## v dials        1.3.0     v tune         1.2.1
## v infer        1.0.7     v workflows    1.1.4
## v modeldata    1.4.0     v workflowsets 1.1.0
## v parsnip      1.2.1     v yardstick    1.3.1
## v recipes      1.1.0
## -- Conflicts ----------------------------------------- tidymodels_conflicts() --
## x scales::discard() masks purrr::discard()
## x dplyr::filter()   masks stats::filter()
## x recipes::fixed()  masks stringr::fixed()
## x dplyr::lag()      masks stats::lag()
## x yardstick::spec() masks readr::spec()
## x recipes::step()   masks stats::step()
## * Learn how to get started at https://www.tidymodels.org/start/
```

## Task 3: Creating the Data

```r
# Set a random seed value so we can obtain the same "random" results
set.seed(2025)

# Create a data frame/tibble named sim_dat
```

```r
sim_dat <- tibble(
  x1 = runif(20, -5, 5), # Random uniform values between -5 and 5
  x2 = runif(20, 0, 100), # Random uniform values between 0 and 100
  x3 = rbinom(20, 1, 0.5) # Random binary values (0 or 1) with equal probability
)

b0 <- 2
b1 <- 0.25
b2 <- -0.5
b3 <- 1
sigma <- 1.5
errors <- rnorm(20, 0, sigma)

sim_dat <- sim_dat %>%
  mutate(
    y = b0 + b1*x1 + b2*x2 + b3*x3 + errors,
    x3 = case_when(
      x3 == 0 ~ "No",
      TRUE ~ "Yes"
    )
  )
```

## Task 4: Traditional MLR Model

```r
mlr_fit <- linear_reg() %>%
  set_mode("regression") %>%
  set_engine("lm") %>%
  fit(y ~ x1 + x2 + x3, data = sim_dat)

tidy(mlr_fit, conf.int = TRUE)
```

```
## # A tibble: 4 x 7
##   term        estimate std.error statistic  p.value conf.low conf.high
##   <chr>          <dbl>     <dbl>     <dbl>    <dbl>    <dbl>     <dbl>
## 1 (Intercept)    1.56     0.743       2.10 5.15e- 2  -0.0116      3.14
## 2 x1             0.434    0.148       2.94 9.66e- 3   0.121       0.747
## 3 x2            -0.491    0.0123    -39.8  2.00e-17  -0.517      -0.464
## 4 x3Yes          1.01     0.792       1.28 2.20e- 1  -0.668       2.69
```

## Task 5: Bootstrapping

```r
# Set a random seed value so we can obtain the same "random" results
set.seed(631)

# Generate the 2000 bootstrap samples
boot_samps <- sim_dat %>%
  bootstraps(times = 2000)

boot_samps
```

```
## # Bootstrap sampling
## # A tibble: 2,000 x 2
```

```
##    splits          id
##    <list>          <chr>
##  1 <split [20/8]>  Bootstrap0001
##  2 <split [20/6]>  Bootstrap0002
##  3 <split [20/6]>  Bootstrap0003
##  4 <split [20/6]>  Bootstrap0004
##  5 <split [20/10]> Bootstrap0005
##  6 <split [20/10]> Bootstrap0006
##  7 <split [20/7]>  Bootstrap0007
##  8 <split [20/6]>  Bootstrap0008
##  9 <split [20/8]>  Bootstrap0009
## 10 <split [20/6]>  Bootstrap0010
## # i 1,990 more rows
```

```
# Create a function that fits a fixed MLR model to one split dataset
fit_mlr_boots <- function(split) {
  lm(y ~ x1 + x2 + x3, data = analysis(split))
}

# Fit the model to each split and store the information
boot_models <- boot_samps %>%
  mutate(
    model = map(splits, fit_mlr_boots),
    coef_info = map(model, tidy)
  )

boots_coefs <- boot_models %>%
  unnest(coef_info)

boots_coefs
```

```
## # A tibble: 8,000 x 8
##    splits         id            model term   estimate std.error statistic  p.value
##    <list>         <chr>         <lis> <chr>     <dbl>     <dbl>     <dbl>    <dbl>
##  1 <split [20/8]> Bootstrap00~ <lm>  (Int~     2.02      0.860       2.35 3.19e- 2
##  2 <split [20/8]> Bootstrap00~ <lm>  x1        0.399     0.146       2.74 1.46e- 2
##  3 <split [20/8]> Bootstrap00~ <lm>  x2       -0.502     0.0138    -36.3  8.64e-17
##  4 <split [20/8]> Bootstrap00~ <lm>  x3Yes     1.52      0.751       2.03 5.94e- 2
##  5 <split [20/6]> Bootstrap00~ <lm>  (Int~     1.34      0.782       1.72 1.05e- 1
##  6 <split [20/6]> Bootstrap00~ <lm>  x1        0.349     0.138       2.53 2.23e- 2
##  7 <split [20/6]> Bootstrap00~ <lm>  x2       -0.488     0.0131    -37.3  5.61e-17
##  8 <split [20/6]> Bootstrap00~ <lm>  x3Yes     1.47      0.813       1.81 8.88e- 2
##  9 <split [20/6]> Bootstrap00~ <lm>  (Int~     2.10      0.714       2.95 9.46e- 3
## 10 <split [20/6]> Bootstrap00~ <lm>  x1        0.412     0.153       2.69 1.60e- 2
## # i 7,990 more rows
```

## Bootstrap Confidence Intervals

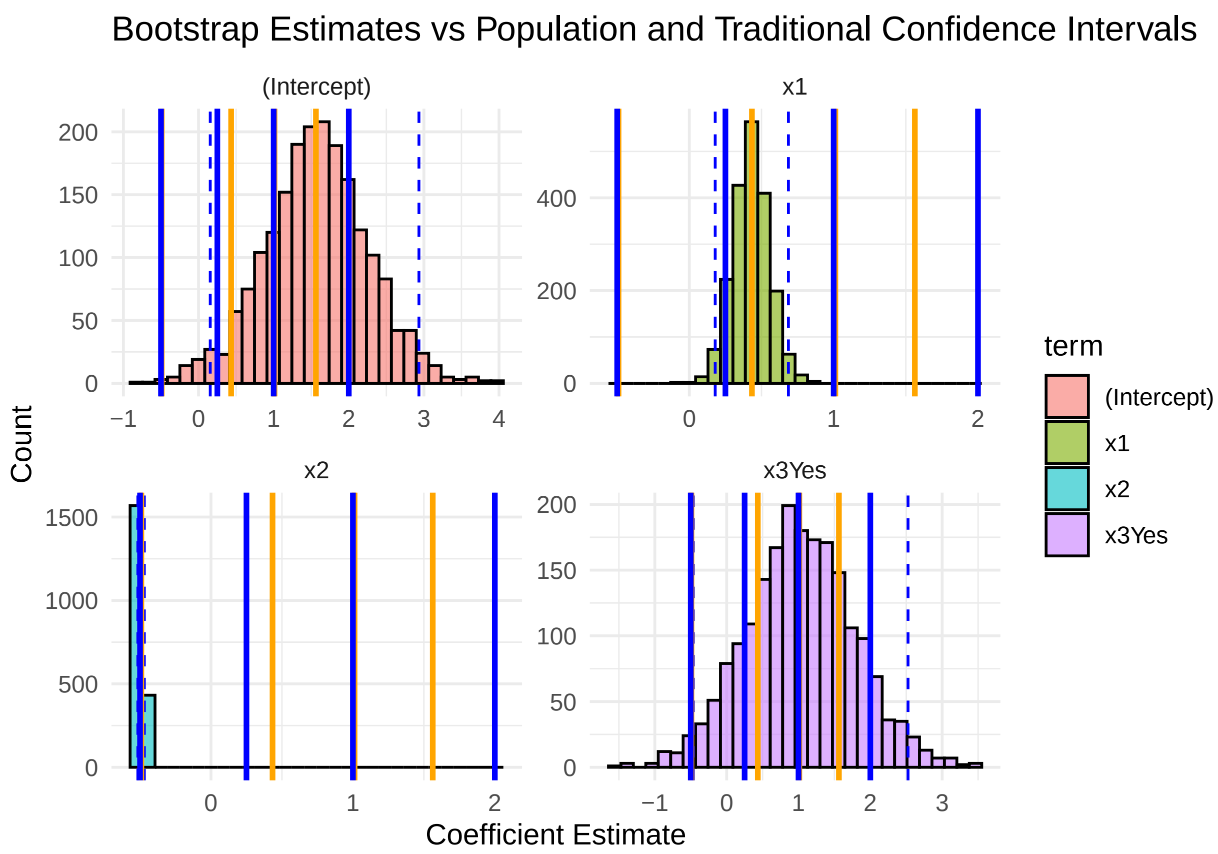
```
boot_int <- int_pctl(boot_models, statistics = coef_info, alpha = 0.05)
boot_int
```

```
## # A tibble: 4 x 6
##   term         .lower .estimate .upper .alpha .method
##   <chr>         <dbl>     <dbl>  <dbl>  <dbl> <chr>
## 1 (Intercept)  0.156      1.57   2.93    0.05 percentile
```

```
## 2 x1           0.179     0.427  0.687   0.05 percentile
## 3 x2          -0.515    -0.491 -0.470   0.05 percentile
## 4 x3Yes       -0.464     1.03   2.52    0.05 percentile
```

### Visualization

```r
ggplot(boots_coefs, aes(x = estimate, fill = term)) +
  geom_histogram(bins = 30, alpha = 0.6, color = "black") +
  facet_wrap(~ term, scales = "free") +
  geom_vline(data = boot_int, aes(xintercept = .lower), col = "blue", linetype = "dashed") +
  geom_vline(data = boot_int, aes(xintercept = .upper), col = "blue", linetype = "dashed") +
  geom_vline(xintercept = c(1.5626060, 0.4336503, -0.4905565, 1.0117182), col = "orange", linetype = "so
  geom_vline(xintercept = c(2, 0.25, -0.5, 1), col = "blue", linetype = "solid", size = 1) +
  theme_minimal() +
  labs(title = "Bootstrap Estimates vs Population and Traditional Confidence Intervals",
       x = "Coefficient Estimate",
       y = "Count")
```




## Answer to Question 5

The bootstrap estimates align closely with the population-level model coefficients. The traditional confidence intervals (orange lines) provide a reference, while the bootstrap intervals (blue dashed lines) capture the variability from resampling.

### Accuracy Assessment:

- The bootstrap confidence intervals contain the true population values for most estimates, indicating reasonable accuracy.
- The traditional method's confidence intervals are slightly narrower than the bootstrap intervals, reflecting the differences in estimation methods.
- The variability of bootstrap estimates is evident, but their central tendency aligns well with the expected population coefficients.

This suggests that bootstrapping provides a robust alternative to traditional inference methods, especially when assumptions about normality or small sample sizes need to be considered.

## Challenge Enhancements

- Added vertical orange lines for traditional confidence intervals.
- Added vertical solid blue lines for population slope values.
- Applied `theme_minimal()` for a cleaner plot appearance.
- Added histogram colors and adjusted transparency for better visibility.
- Included clear axis labels and a title to improve interpretability.