Point Estimate and C.I. Simulation Evaluation

## Load Libraries

library(tidyverse)

── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──  
✔ dplyr 1.1.4 ✔ readr 2.1.5  
✔ forcats 1.0.0 ✔ stringr 1.5.1  
✔ ggplot2 3.5.1 ✔ tibble 3.2.1  
✔ lubridate 1.9.4 ✔ tidyr 1.3.1  
✔ purrr 1.0.4   
── Conflicts ────────────────────────────────────────── tidyverse\_conflicts() ──  
✖ dplyr::filter() masks stats::filter()  
✖ dplyr::lag() masks stats::lag()  
ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors

library(tidymodels)

── Attaching packages ────────────────────────────────────── tidymodels 1.2.0 ──  
✔ broom 1.0.7 ✔ rsample 1.2.1  
✔ dials 1.4.0 ✔ tune 1.2.1  
✔ infer 1.0.7 ✔ workflows 1.2.0  
✔ modeldata 1.4.0 ✔ workflowsets 1.1.0  
✔ parsnip 1.3.0 ✔ yardstick 1.3.2  
✔ recipes 1.1.1   
── Conflicts ───────────────────────────────────────── tidymodels\_conflicts() ──  
✖ scales::discard() masks purrr::discard()  
✖ dplyr::filter() masks stats::filter()  
✖ recipes::fixed() masks stringr::fixed()  
✖ dplyr::lag() masks stats::lag()  
✖ yardstick::spec() masks readr::spec()  
✖ recipes::step() masks stats::step()  
• Use tidymodels\_prefer() to resolve common conflicts.

## Simulate Data and Fit Regression Models

We are going to simulate a bunch of different data sets with the same underlying data generating process.

# Set population parameter values. These govern the data generating process  
beta0 <- 10  
beta1 <- 5  
  
# set the number of simulations  
N <- 100

Define the model we want to use.

# Specify the model type and engine.  
model <- linear\_reg() |>   
 set\_engine("lm")

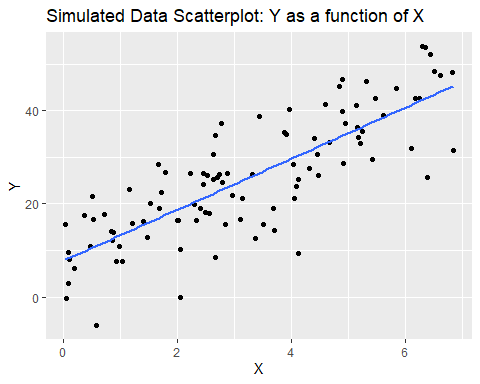
Perform N simulations and extract the parameter estimates using the model applied to each simulated data set.

# set seed for reproducibility  
set.seed(31296)  
  
# vectors for parameter estimates  
beta0\_estimates <- c()  
beta1\_estimates <- c()  
  
# for each simulation,  
for (i in 1:N){  
   
 # Simulate data.  
 sim\_data <- tibble(  
 x = runif(100, min = 0, max = 7),  
 y = beta0 + beta1 \* x + rnorm(100, mean = 0, sd = 7))  
   
 # fit the model, regressing y on x  
 model\_results <- model |>   
 fit(y ~ x, data = sim\_data)  
   
 # extract tidy model results and pull out the parameter estimates  
 estimated\_params <- model\_results |>  
 tidy() |>  
 pull(estimate)  
   
 # store parameter estimates in their corresponding vectors  
 beta0\_estimates <- c(beta0\_estimates, estimated\_params[1])  
 beta1\_estimates <- c(beta1\_estimates, estimated\_params[2])  
}

Look at the last simulated data set.

sim\_data |>  
 ggplot(aes(x = x, y = y)) +  
 geom\_point() +  
 labs(x = "X",  
 y = "Y",  
 title = "Simulated Data Scatterplot: Y as a function of X") +  
 geom\_smooth(method = "lm",   
 se = FALSE)

`geom\_smooth()` using formula = 'y ~ x'



Look at point estimates from all simulated data sets.

(estimated\_params <- tibble(  
 sim\_number = 1:length(beta0\_estimates),  
 beta0 = beta0\_estimates,  
 beta1 = beta1\_estimates,  
))

# A tibble: 100 × 3  
 sim\_number beta0 beta1  
 <int> <dbl> <dbl>  
 1 1 9.24 5.11  
 2 2 8.18 4.94  
 3 3 11.9 4.23  
 4 4 9.09 5.24  
 5 5 11.5 4.90  
 6 6 10.2 5.31  
 7 7 9.19 4.86  
 8 8 9.42 5.19  
 9 9 8.87 5.03  
10 10 10.8 5.29  
# ℹ 90 more rows

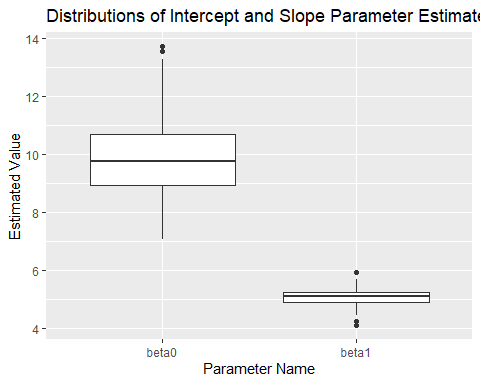
Tidy the parameter estimates.

(estimated\_params <- estimated\_params |>  
 pivot\_longer(-sim\_number, names\_to = "parameter", values\_to = "estimate"))

# A tibble: 200 × 3  
 sim\_number parameter estimate  
 <int> <chr> <dbl>  
 1 1 beta0 9.24  
 2 1 beta1 5.11  
 3 2 beta0 8.18  
 4 2 beta1 4.94  
 5 3 beta0 11.9   
 6 3 beta1 4.23  
 7 4 beta0 9.09  
 8 4 beta1 5.24  
 9 5 beta0 11.5   
10 5 beta1 4.90  
# ℹ 190 more rows

Plot the distributions of point estimates.

estimated\_params |>  
 ggplot(aes(x = parameter, y = estimate)) +  
 geom\_boxplot() +  
 labs(x = "Parameter Name",  
 y = "Estimated Value",  
 title = "Distributions of Intercept and Slope Parameter Estimates")



What values did we choose for these parameters?

beta0

[1] 10

beta1

[1] 5

What should we take away from this? Point estimates can vary widely using different data sets from the same data generating process, and thus we should not rely on a point estimate alone to inform our managerial decisions.

## Confidence Intervals

Let’s repeat the simulation process but evaluate our model estimates using confidence intervals.

What is the formal interpretation of a confidence interval?

“If you were to repeat the data sampling, modeling, and confidence interval computation many times, 95% of the calculated intervals would contain the true population parameter.”

This process is unfeasible with real-world data. But, we CAN do it with simulated data. Let’s test this theory!

Perform N simulations and extract the confidence interval estimates from the model applied to each simulated data set.

# set seed for reproducibility  
set.seed(31296)  
  
# tibble for confidence interval estimates  
conf\_int\_estimates <- tibble()  
  
for (i in 1:N){  
   
 # Simulate data.  
 sim\_data <- tibble(  
 x = runif(100, min = 0, max = 7),  
 y = beta0 + beta1 \* x + rnorm(100, mean = 0, sd = 7))  
   
 # fit model to simulated data  
 model\_results <- model |>   
 fit(y ~ x, data = sim\_data)  
   
 # extract tidy model results including confidence intervals,  
 # create variables to track simulation number, parameter name,  
 # and select desired variables  
 estimated\_params <- model\_results |>  
 tidy(conf.int = TRUE) |>  
 mutate(sim\_number = i,  
 parameter = c("beta0", "beta1")) |>  
 select(sim\_number, parameter, estimate, conf.low, conf.high)  
   
 # add estimates from current model to dataframe  
 conf\_int\_estimates <- conf\_int\_estimates |>  
 bind\_rows(estimated\_params)  
}

View confidence interval estimates.

conf\_int\_estimates

# A tibble: 200 × 5  
 sim\_number parameter estimate conf.low conf.high  
 <int> <chr> <dbl> <dbl> <dbl>  
 1 1 beta0 9.24 6.25 12.2   
 2 1 beta1 5.11 4.37 5.85  
 3 2 beta0 8.18 5.01 11.3   
 4 2 beta1 4.94 4.19 5.69  
 5 3 beta0 11.9 9.27 14.5   
 6 3 beta1 4.23 3.60 4.87  
 7 4 beta0 9.09 6.13 12.1   
 8 4 beta1 5.24 4.50 5.98  
 9 5 beta0 11.5 8.67 14.3   
10 5 beta1 4.90 4.17 5.62  
# ℹ 190 more rows

Sort the estimates of each parameter by the point estimate (in ascending order) - note that this breaks the association between beta0 and beta1 estimates from the same models. This is done for visualization purposes only.

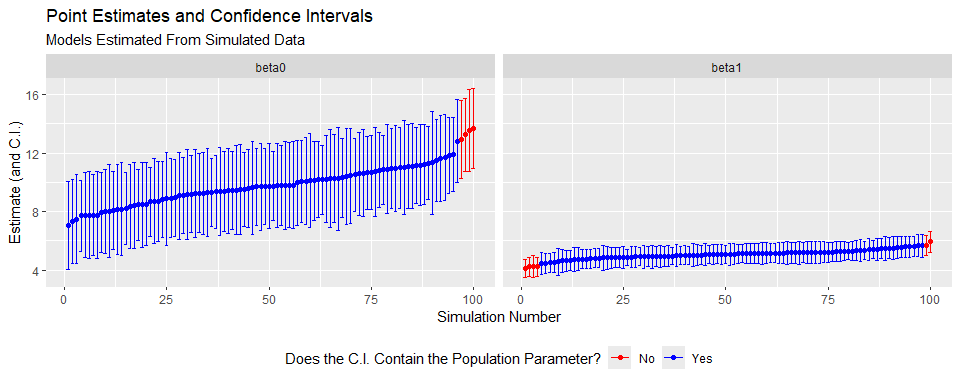
conf\_int\_estimates\_sorted <- conf\_int\_estimates |>  
 # arrange is ascending order of point estimate  
 arrange(parameter, estimate) |>  
 # for each parameter,  
 group\_by(parameter) |>  
 # re-define simulation number in order of point estimate magnitude  
 # create new variable contains\_param that is "Yes" if the confidence interval contains the population estimate and "No" otherwise  
 mutate(sim\_number = 1:n(),  
 contains\_param = if\_else(parameter == "beta0", if\_else(conf.low <= beta0 & conf.high >= beta0, "Yes", "No"), if\_else(conf.low <= beta1 & conf.high >= beta1, "Yes", "No"))) |>  
 ungroup()

conf\_int\_estimates\_sorted

# A tibble: 200 × 6  
 sim\_number parameter estimate conf.low conf.high contains\_param  
 <int> <chr> <dbl> <dbl> <dbl> <chr>   
 1 1 beta0 7.05 4.05 10.1 Yes   
 2 2 beta0 7.30 4.44 10.2 Yes   
 3 3 beta0 7.49 4.47 10.5 Yes   
 4 4 beta0 7.72 5.29 10.2 Yes   
 5 5 beta0 7.76 4.84 10.7 Yes   
 6 6 beta0 7.76 4.78 10.7 Yes   
 7 7 beta0 7.77 4.98 10.6 Yes   
 8 8 beta0 7.77 4.78 10.8 Yes   
 9 9 beta0 7.97 5.25 10.7 Yes   
10 10 beta0 7.99 5.11 10.9 Yes   
# ℹ 190 more rows

Plot the point estimates (in ascending order) along with “error bars” showing the boundaries of the associated confidence intervals.

conf\_int\_estimates\_sorted %>%  
 ggplot(aes(x = sim\_number, y = estimate, color = contains\_param)) +  
 geom\_point() +  
 geom\_errorbar(aes(ymax = conf.high, ymin = conf.low)) +  
 facet\_wrap(~parameter) +  
 scale\_color\_manual(values=c("red", "blue")) +  
 theme(legend.position="bottom") +  
 labs(x = "Simulation Number",  
 y = "Estimate (and C.I.)",  
 color="Does the C.I. Contain the Population Parameter?",  
 title="Point Estimates and Confidence Intervals",  
 subtitle = "Models Estimated From Simulated Data")



What percentage of confidence intervals contain the population parameter estimate?

conf\_int\_estimates\_sorted |>  
 group\_by(parameter) |>  
 summarize(percent\_contains = mean(contains\_param == "Yes"))

# A tibble: 2 × 2  
 parameter percent\_contains  
 <chr> <dbl>  
1 beta0 0.96  
2 beta1 0.94

While confidence intervals are not guaranteed to contain the population parameter, they give a more conservative estimate of coefficient values.

So what are the advantages of confidence intervals?

* They give an estimate of the coefficient magnitude AND statistical significance all in one.
* They quantify uncertainty - the wider the interval, the less certain we are about our estimate and the less likely we will have statistical significance.

How do I interpret a single confidence interval?

“The method used to construct this interval will capture the population parameter in 95% of cases.”

“With 95% confidence, I infer that the true value of beta is between (lower bound, upper bound).”