BIOS 731 HW 2: Simulation Studies

```
source(here::here("source", "01_simulate_data.R"))
source(here::here("source", "02_apply_methods.R"))
source(here::here("source", "03_extract_estimates.R"))

# Define simulation function to parallelize
run_sim_scenario <- function(index){

  # Define row from parameters in global environment
  p <<- index

  # Load functions
  source(here::here("source", "01_simulate_data.R"))
  source(here::here("source", "02_apply_methods.R"))
  source(here::here("source", "03_extract_estimates.R"))

  # Run simulation process
  source(here::here("simulations", "run_simulations.R"))

  # Save results
  return(list("results" = results_df,
              "timing_log" = timing))
}
```

Problem 0: Reproducibility

The GitHub repository containing this work can be found at

https://github.com/ryanjtaylor97/bios731_hw2_taylor.

Problem 1.1: ADEMP Structure

- *A*: The goal of this simulation study is to evaluate the accuracy of a linear regression to estimate a binary treatment effect. We also want to compare the coverage and computational efficiency of three methods of estimating a 95% confidence interval around the treatment effect estimates: a Wald confidence interval, a bootstrap percentile interval, and a bootstrap t interval. All of these methods should generate intervals that cover the true treatment effect in at least 95% of simulations.
- *D*: We generate data under the assumption

$$Y_i = \beta_0 + \beta_{treat} X_i + \epsilon_i$$

for $i = 1, \dots, n$.

In all scenarios, X_i is a binary variable that we randomly generate with $P(X_i = 1) = 0.5$, and we set $\beta_0 = 1$.

We set up a full factorial simulation design with the alternatives:

$n = 10,\ 50,$ or 500;

$\beta_{treat} = 0,\ 0.5,$ or 2;

$\epsilon_i \overset{iid}{\sim} N(0, 2)$ or $\epsilon_i = u * \sqrt{2 * \frac{\nu - 2}{\nu}},$ where $u \overset{iid}{\sim} t_\nu$ and $\nu = 3$.

Therefore, we include **18** total scenarios.

- *E*: We are trying to learn about one estimand through this study: $\hat{\beta}_{treat}$ from a linear regression.
- *M*: We are evaluating one method for a point estimate $\hat{\beta}_{treat}$, which is linear regression. We are then comparing three methods for a confidence interval around this estimand: a Wald confidence interval, a bootstrap percentile confidence interval, and a bootstrap t confidence interval.
- *P*: We will evaluate $\hat{\beta}_{treat}$ using its bias. We will compare the three methods for the confidence interval based on coverage and computation speed.

Problem 1.2: nSim

```
# Define desired coverage and max Monte Carlo error
ci_pct <- 0.95
mcse_max <- 0.01

# Calculate necessary simulations
nSim <- ci_pct * (1 - ci_pct) / (mcse_max)^2
```

Based on desired coverage of 95% with a maximum Monte Carlo standard error of 1%, we should perform 475 simulations for each scenario.

Problem 1.3: Implementation

Parameters

First, we set up the full factorial design, defining all of the parameters that change between scenarios and expanding them into a data frame of all combinations. We also define the number of outer and inner bootstrap resamples we will run. We use the recommended numbers from the homework: 500 outer bootstrap resamples and 100 inner bootstrap resamples.

```
# Define sample size, treatment effect, error distribution
data_n <- c(10, 50, 100)
beta_true <- c(0, 0.5, 2)
error_dist <- c("gauss", "t")

# Combine these into table
params <- expand_grid(data_n, beta_true, error_dist)

# Set bootstrap numbers
boot_outer <- 500
boot_inner <- 100
```

Setting seeds

Next, we define the seeds we will use for these simulations. We use a different seed for each simulation and for each outer bootstrap resample in each scenario. We generate these seeds by setting one seed arbitrarily, then randomly generating numbers from 1 to the total number of seeds we need. Then, we organize these into a nested list by scenario and by simulation number.

```
### Define seeds

# Count number of randomization steps we will perform
# -> For each scenario: n simulations, 1 data set,
#                       B_outer bootstrap samples per data set,
#                       B_inner resamples per outer sample
num_sims <- nrow(params) * nSim * (1 + boot_outer * boot_inner)

# -> We will use 1 seed per bootstrap
```

```r
num_seeds <- nrow(params) * nSim * (1 + boot_outer)

# Set initial seed, outside of next range
set.seed(num_seeds + 413)

# Randomize order of other seeds
seeds <- sample(1:num_seeds, num_seeds)

# Organize these by simulation step
seed_list <- split(
  seeds,
  cut(1:num_seeds, breaks = nrow(params))
) %>%
  map(~split(.x, cut(1:length(.x), breaks = nSim)))
```

Comparing possible regression functions

Then, before running these simulations, we try to improve the computational efficiency of our process. For both bootstrap methods of generating a confidence interval, we do not need the estimated standard error of our estimated β_{treat}; we only need the coefficient itself. While this is most easily generated by the function `lm`, it can also be calculated using the function `lm.fit`. We check that these two methods return the same coefficient, and also use `benchmark` to test their computation speeds against each other. We find that `lm.fit` is significantly quicker, so we use it in both of our bootstrap functions.

```r
set.seed(num_seeds + 978)

test_df <- get_simdata(n = max(data_n),
                       beta_treat = max(beta_true),
                       variance = 2,
                       error_dist = "gauss")

tibble(
  lm = coef(fit_model_lm(test_df)),
  lm.fit = coef(fit_model_lm_fit(x_mat = matrix(cbind(1, test_df$x),
                                                nrow = nrow(test_df)),
                                 y_vec = test_df$y))
)
```

lm	lm.fit
0.8763628	0.8763628
1.7465801	1.7465801

```
bench <- microbenchmark::microbenchmark(
  "coef_lm" = coef(fit_model_lm(test_df))[2],
  "coef_lm.fit" =  coef(fit_model_lm_fit(x_mat = matrix(cbind(1, test_df$x),
                                                        nrow = nrow(test_df)),
                                         y_vec = test_df$y))[2]
  )

print(bench)
```

```
Unit: microseconds
        expr     min       lq      mean   median       uq     max neval cld
     coef_lm 120.253 125.1730 139.50619 130.5235 140.1995 617.583   100  a
  coef_lm.fit  14.186  15.6415  22.27612  16.3590  17.6710 538.863   100   b
```

Running simulation scenarios

Finally, we run simulations for all 18 scenarios listed. Results are saved in a nested list: for each scenario, we save one dataset of results for each simulation and one dataset of timing logs.

This process takes about 30 minutes total, so we set up a switch to load the output from these simulations if we are not running this code for the first time.

```
# Determine number of cores available
num_cores <- detectCores() - 2

if(DO_SIMULATIONS){

  # Make cluster for parallelization
  cl <- makeCluster(num_cores)

  registerDoParallel(cl)

  # Combine results and timing from all scenarios
  simulation_output <- foreach(
    i = 1:nrow(params),
    .packages = c("tidyverse", "magrittr", "tictoc", "broom"),
```

```
    .inorder = F
  ) %dopar% {

    # Move all necessary variables to global environment
    .GlobalEnv$params <- params
    .GlobalEnv$nSim <- nSim
    .GlobalEnv$seed_list <- seed_list
    .GlobalEnv$boot_outer <- boot_outer
    .GlobalEnv$boot_inner <- boot_inner

    run_sim_scenario(index = i)
  }

  stopCluster(cl)

  # Save combined output
  save(simulation_output,
       file = here::here("data", "combined_scenario_output.rda"))

} else {

  # Load output
  load(file = here::here("data", "combined_scenario_output.rda"))
}
```

Problem 1.4: Results summary

We combine the results and timing datasets across all 18 scenarios to summarize the results. We also calculate coverage and bias at this stage.

```
# Collect results and timing into two separate datasets
results_all <- map(simulation_output, ~.[["results"]]) %>% bind_rows()
timing_all <- map(simulation_output, ~.[["timing_log"]]) %>% bind_rows()

# Merge information from parameters
params_merge <- params %>%
  mutate(scenario = 1:n())

# Merge parameters and calculate coverage
results_all %<>%
  left_join(params_merge) %>%
```

```
  mutate(coverage = as.numeric(beta_true >= conf.low & beta_true <= conf.high),
         bias = beta_hat - beta_true)
```

Next, we summarize the timing, average $\hat{\beta}$, coverage percent, confidence interval width, and standard error across all simulations by scenario and interval estimation method. The standard error under the Wald method is taken from the output of `lm`; for the two bootstrap methods, we take the standard error of all bootstrap estimates $\hat{\beta}_1, \ldots, \hat{\beta}_B$. Therefore, the standard error estimates for both bootstrap methods are the same.

```
# Summarize time
scenario_timing <- timing_all %>%
  group_by(scenario, step) %>%
  summarize(time_total = sum(time)) %>%
  ungroup() %>%
  arrange(scenario, desc(time_total)) %>%
  group_by(scenario) %>%
  mutate(scenario_pct = time_total / first(time_total)) %>%
  ungroup()

# Trim down to only the methods we are summarizing
method_timing <- scenario_timing %>%
  filter(str_detect(step, "Boot|Wald")) %>%
  select(scenario, method = step, time_total) %>%
  mutate(method = case_match(method,
                             "Bootstrap t" ~ "Boot t",
                             "Bootstrap percentile" ~ "Boot Pct",
                             "Wald" ~ "Wald"))

# Summarize bias, coverage, SD
scenario_results <- results_all %>%
  group_by(scenario, data_n, beta_true, error_dist, method) %>%
  summarize(beta_hat_avg = mean(beta_hat, na.rm = T),
            coverage_pct = mean(coverage, na.rm = T),
            std_error = mean(std.error, na.rm = T),
            ci_width = mean(conf.high - conf.low, na.rm = T)) %>%
  ungroup() %>%
  mutate(bias = beta_hat_avg - beta_true) %>%
  left_join(method_timing) %>%
  mutate(method = factor(method,
                         levels = c("Wald", "Boot Pct", "Boot t")),
         error_dist = case_match(error_dist,
```

```
                                    "gauss" ~ "Gaussian",
                                    "t" ~ "t-Dist."))
```

Bias

The first performance measure we tracked was the bias of the linear regression estimate. This measure is compared across scenarios in the table below.

```
# Summarize bias
bias_table <- scenario_results %>%
  distinct(data_n, beta_true, error_dist,
           bias, beta_hat_avg) %>%
  arrange(error_dist, beta_true, data_n) %>%
  select("Error Dist." = error_dist,,
         "Beta" = beta_true,
         "N" = data_n,
         "Mean Est." = beta_hat_avg,
         "Bias" = bias)

# Print this table
kbl(bias_table,
    booktabs = T,
    caption = "Beta Estimation",
    digits = 3) %>%
  kable_classic(full_width = F) %>%
  column_spec(1:2, bold = T) %>%
  column_spec(3, italic = T) %>%
  collapse_rows(columns = 1:2,
                valign = "top",
                row_group_label_position = "first",
                latex_hline = "full") %>%
  add_header_above(c("Parameters" = 3,
                     "Performance Measures" = 2))
```

To see more details about the bias across simulations and scenarios, we also plot empirical density functions of the bias across scenarios.

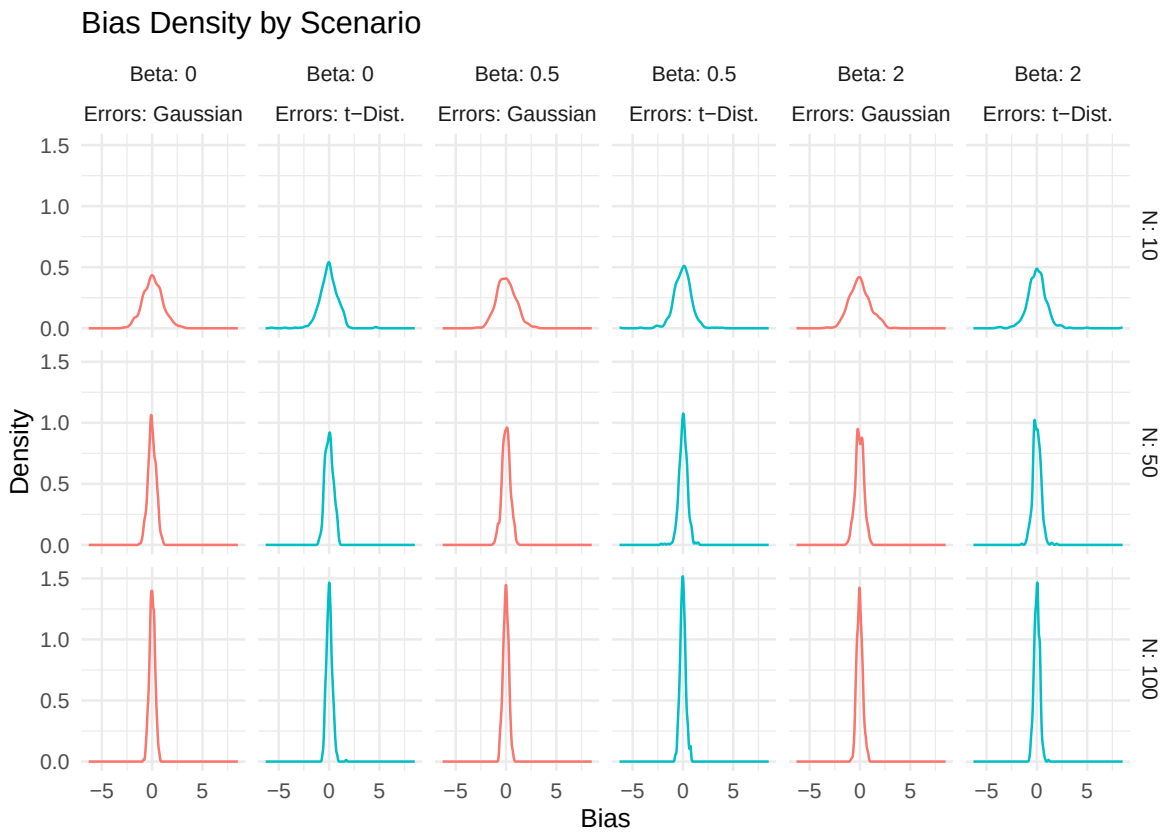
```
bias_plot <- ggplot(results_all %>%
                      mutate(error_dist = case_match(error_dist,
                                    "gauss" ~ "Gaussian",
                                    "t" ~ "t-Dist.")) %>%
```

Table 2: Beta Estimation

Parameters			Performance Measures	
Error Dist.	Beta	N	Mean Est.	Bias
Gaussian	**0.0**	*10*	0.066	0.066
		50	-0.006	-0.006
		100	0.006	0.006
	0.5	*10*	0.525	0.025
		50	0.508	0.008
		100	0.485	-0.015
	2.0	*10*	1.982	-0.018
		50	1.966	-0.034
		100	1.980	-0.020
t-Dist.	**0.0**	*10*	-0.040	-0.040
		50	0.012	0.012
		100	0.028	0.028
	0.5	*10*	0.455	-0.045
		50	0.524	0.024
		100	0.490	-0.010
	2.0	*10*	2.009	0.009
		50	2.002	0.002
		100	2.006	0.006

```
                      rename("N" = data_n,
                             "Beta" = beta_true,
                             "Errors" = error_dist),
                    aes(x = bias,
                        color = Errors)) +
  geom_density() +
  facet_grid(N ~ Beta + Errors, labeller = label_both) +
  labs(x = "Bias", y = "Density", color = "Error Distribution",
       title = "Bias Density by Scenario") +
  theme_minimal() +
  theme(legend.position = "none")

bias_plot
```



Coverage

Next, we compare the three methods of estimating a 95% confidence interval for $\hat{\beta}$. We compare three measures to understand the differences between these methods: average standard error, coverage percentage of the confidence interval, and total computation time over all simulations. These are presented in the table below.

```
# Summarize coverage and standard error
covg_table <- scenario_results %>%
  select(data_n, beta_true, error_dist,
         method,
         std_error, coverage_pct, time_total) %>%
  arrange(error_dist, beta_true, data_n, method) %>%
  select("Error Dist." = error_dist,
         "Beta" = beta_true,
         "N" = data_n,
         se = std_error,
         cvg = coverage_pct,
         tms = time_total,
         method) %>%
  pivot_wider(names_from = method,
              values_from = c(se, cvg, tms)) %>%
  mutate(across(matches("se"),
                ~scales::label_number(accuracy = 0.01)(.)),
         across(matches("cvg"),
                ~scales::label_percent(accuracy = 0.1)(.)),
         across(matches("tms"),
                ~scales::label_number(accuracy = 0.1)(.)))


# Print this table
kbl(covg_table,
    booktabs = T,
    caption = "Coverage Estimation",
    col.names = str_remove_all(names(covg_table), "(.*_)?")) %>%
  kable_classic(full_width = F) %>%
  column_spec(1:2, bold = T) %>%
  column_spec(3, italic = T) %>%
  collapse_rows(columns = 1:2,
                valign = "top",
                row_group_label_position = "first",
                latex_hline = "full") %>%
```

Table 3: Coverage Estimation

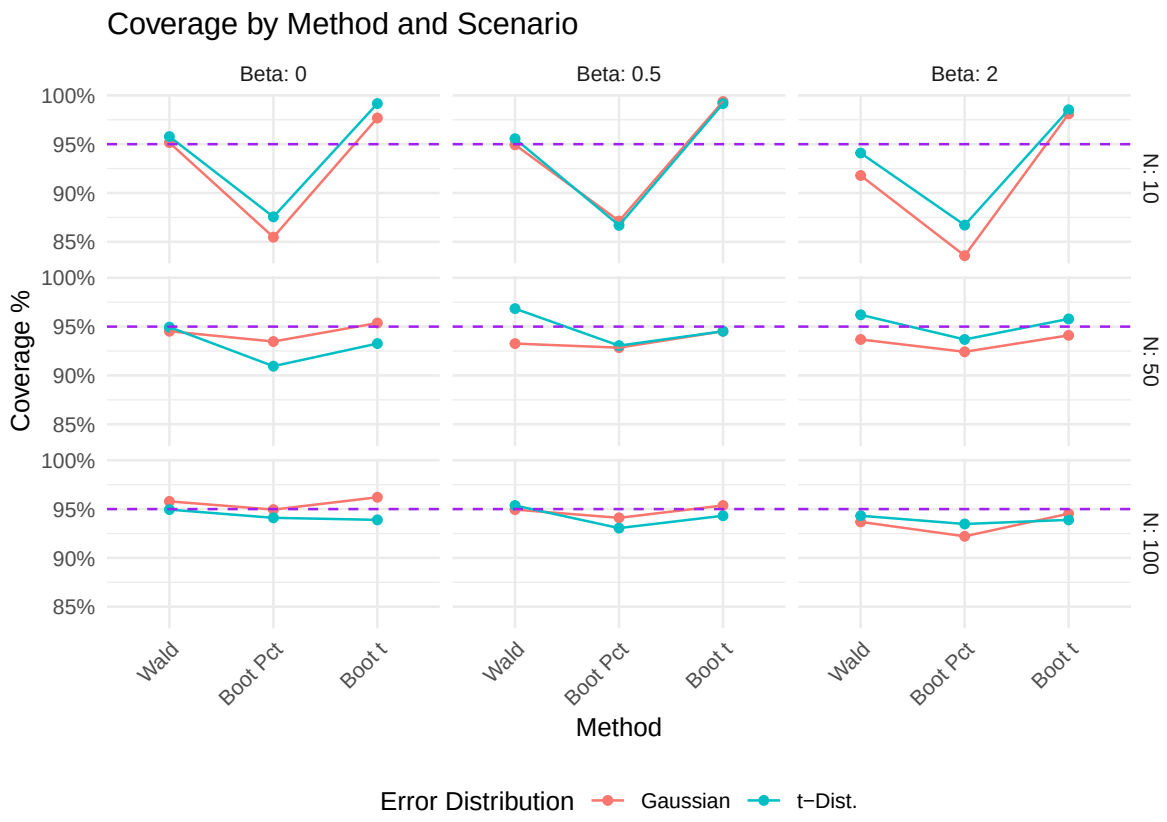
Parameters			Avg. Std. Error			Coverage			Time (Secs.)		
Error Dist.	Beta	N	Wald	Boot Pct	Boot t	Wald	Boot Pct	Boot t	Wald	Boot Pct	Boot t
Gaussian	**0.0**	*10*	0.92	0.81	0.81	95.2%	85.5%	97.7%	2.3	6.2	494.2
		50	0.40	0.40	0.40	94.5%	93.5%	95.4%	2.4	7.2	601.1
		100	0.28	0.28	0.28	95.8%	94.9%	96.2%	2.4	7.9	680.3
	0.5	*10*	0.91	0.83	0.83	94.9%	87.1%	99.4%	2.4	6.2	497.5
		50	0.40	0.40	0.40	93.3%	92.8%	94.5%	2.4	7.1	606.6
		100	0.28	0.28	0.28	94.9%	94.1%	95.4%	2.3	7.5	648.7
	2.0	*10*	0.92	0.81	0.81	91.8%	83.6%	98.1%	2.4	6.3	500.3
		50	0.40	0.40	0.40	93.7%	92.4%	94.1%	2.3	7.0	608.2
		100	0.28	0.28	0.28	93.7%	92.2%	94.5%	1.6	5.5	506.8
t-Dist.	**0.0**	*10*	0.81	0.73	0.73	95.8%	87.6%	99.2%	2.4	6.5	498.9
		50	0.38	0.37	0.37	94.9%	90.9%	93.3%	2.4	7.2	600.9
		100	0.27	0.27	0.27	94.9%	94.1%	93.9%	2.5	7.8	679.4
	0.5	*10*	0.81	0.73	0.73	95.6%	86.7%	99.2%	2.3	6.3	497.8
		50	0.38	0.38	0.38	96.8%	93.1%	94.5%	2.5	7.1	606.8
		100	0.27	0.27	0.27	95.4%	93.1%	94.3%	2.3	7.6	651.2
	2.0	*10*	0.82	0.73	0.73	94.1%	86.7%	98.5%	2.3	6.2	498.5
		50	0.38	0.38	0.38	96.2%	93.7%	95.8%	2.4	7.2	608.0
		100	0.26	0.26	0.26	94.3%	93.5%	93.9%	1.6	5.6	506.3

```
  add_header_above(c("Parameters" = 3,
                     "Avg. Std. Error" = 3,
                     "Coverage" = 3,
                     "Time (Secs.)" = 3))
```

To make these results easier to view and interpret, we plot a comparison of coverage percentage. We also plot a comparison of the average width of the confidence intervals across the three interval calculation methods for each scenario.

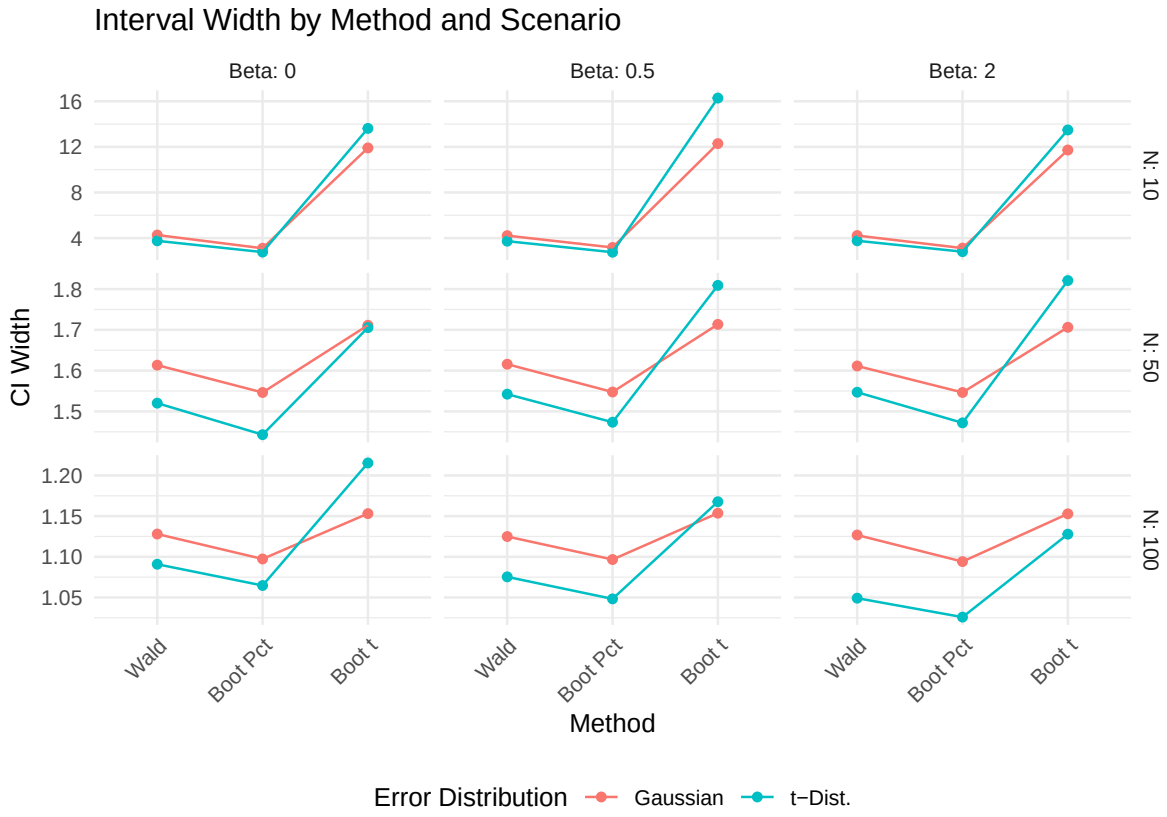
```
covg_plot <- ggplot(scenario_results %>%
                      rename("N" = data_n, "Beta" = beta_true),
                    aes(x = method, y = coverage_pct,
                        color = error_dist, group = error_dist)) +
  geom_point() +
  geom_line() +
  geom_hline(yintercept = 0.95, color = "purple", linetype = 2) +
  facet_grid(N ~ Beta, labeller = label_both) +
  labs(x = "Method", y = "Coverage %", color = "Error Distribution",
       title = "Coverage by Method and Scenario") +
  theme_minimal() +
  scale_y_continuous(labels = scales::label_percent()) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1),
        legend.position = "bottom")

covg_plot
```



```r
width_plot <- ggplot(scenario_results %>%
                       rename("N" = data_n, "Beta" = beta_true),
                     aes(x = method, y = ci_width,
                         color = error_dist, group = error_dist)) +
  geom_point() +
  geom_line() +
  facet_grid(N ~ Beta, labeller = label_both, scales = "free_y") +
  labs(x = "Method", y = "CI Width", color = "Error Distribution",
       title = "Interval Width by Method and Scenario") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1),
        legend.position = "bottom")

width_plot
```



Problem 1.5: Discussion

This simulation study first verifies that linear regression is relatively unbiased when estimating a treatment effect corresponding to a binary variable. We find that the bias of our estimates is low across all scenarios, and is close to 0 more frequently as the data sample size increases. Second, we find that the Wald and bootstrap t methods for constructing a confidence interval have better coverage than the bootstrap percentile method, especially when the data sample size is small. When the data is large, all three methods achieve coverage of at least 92.2%. Notably, however, the bootstrap t confidence intervals are larger on average, and therefore likely have less power.

In terms of computation time, the Wald confidence interval is fastest, and the bootstrap percentile method takes about 3 times as long. The bootstrap t method is by far the slowest, and takes almost 100 times the time of the bootstrap percentile method.

When $\epsilon_i \sim N(0, 2)$, the bootstrap t interval typically provides better coverage. The Wald confidence interval provides slightly less coverage, but is a much more narrow interval.

When the data are generated with heavy-tailed errors, the Wald standard error most commonly provides the best coverage in our simulation. The bootstrap t provides the best coverage when the data sample size is 10. This may indicate that the Central Limit Theorem assumptions behind the Wald interval are not satisfied when the data has only 10 observations.

The bootstrap percentile method performed the worst out of these three methods, but seemed to perform better when the true β value was farther from 0. The bootstrap t confidence interval most commonly achieved at least 95% coverage. However, the Wald interval often achieved only slightly less coverage with significantly less computation time and a smaller average interval width. Therefore, a practical recommendation would be to use the bootstrap t interval if type I error is more of a concern and to use the Wald interval in most other contexts. However, a limitation of this study is that both error distributions used are symmetrical, and bootstrap intervals are often more accurate for asymmetrical distributions.