A Comparision of Bootstrap Methods

Justin Papagelis *
Department of Mathematics and Statistics, Amherst College

December 2, 2022

Abstract

For my project, I plan to explore different bootstrap methods for creating confidence intervals and then perform comparisons between a couple of the methods. My paper will re-introduce the idea of bootstrap to my peers and give some background on constructing confidence intervals. We will also go deeper into the theory behind the construction of confidence intervals from bootstrapped data. These various methods to create a confidence interval from a bootstrap can include the percentile method, bias-corrected method, accelerated method, and the studentized method, as well as others. I will demonstrate how these bootstrap methods work using "toy examples," which will be datasets in which a specific bootstrap method is appropriate. To demonstrate my understanding of the methods, I will write a simulation to compare a few of them and determine how they perform against each other. I will show my understanding of the different methods of creating confidence intervals from bootstrapped data by communicating the statistical theory in a concise and accessible way to my peers. Writing the simulation and sharing conclusions will demonstrate my ability to implement statistical methods in practice as well as my ability to analyze the results of the simulation.

Keywords: confidence interval, simulation, percentile, bias-corrected, acceleration, studentized

*The authors gratefully acknowledge ...

1 Introduction

Bootstrapping is an essential tool in Statistics, even more so now that there is access to higher computing power. We use bootstrapping to estimate the desired population parameter from a given sample without making assumptions about any underlying distributions of the sample. Different bootstrap techniques are developed, but we will focus on the non-parametric bootstrap for our purposes. One way to make a statistical inference is through confidence intervals which give a range of estimates for the unknown population parameter at a certain confidence level. We can use bootstrapping to create accurate approximate confidence intervals for our population parameter even without its underlying distributions. There are many different ways to create intervals, and we will go through a couple of them: The Standard Method, The Percentile Method, The Bias-Corrected (BC) Method, The Bias-Corrected with Acceleration (BCa) Method, and finally, The Studentized Method. Additionally, we will go through an example of creating a bootstrapped confidence interval for each method. Then, we will perform a simulation to evaluate the performance of the bootstrapped confidence interval methods on different distribution types.

2 Exposition

2.1 The Non-Parametric Bootstrap

Bootstrapping is a statistical method of resampling that allows the estimation of a test statistic from an unknown distribution. In particular, bootstrapping is a computational heavy method which is useful for many different situations.

First, we introduce the non-parametric bootstrap. Suppose we have a random sample $X = (x_1, x_2, \ldots, x_n)$ from our unknown distribution, F and a statistic of interest $\hat{\theta} = \hat{\theta}(X)$ (Efron & Hastie 2021). Ideally, the desired test statistic could be found by repeatedly sampling new reproductions of X from F. However F is unknown, so this is not possible. The non-parametric bootstrap creates an estimate $\hat{F}$ from F using our sample X without making any parametric assumptions about F (such as its distribution type). Therefore, the bootstrap sample could be represented as $X^* = (x_1^*, x_2^*, \ldots, x_n^*)$ where each x_i^* is sam-

pled randomly with equal probability and with replacement from $\{x_1, x_2, \ldots, x_n\}$. From this bootstrap sample, a bootstrap replication of the test statistic can be computed using $\hat{\theta}^* = \hat{\theta}(X^*)$. A large number, B, of bootstrap samples are drawn independently and the corresponding bootstrap replication of the test statistic is calculated.

$$\hat{\theta}^{*b} = \hat{\theta}(X^{*b}) \text{ for } b = 1, 2, \ldots, B.$$

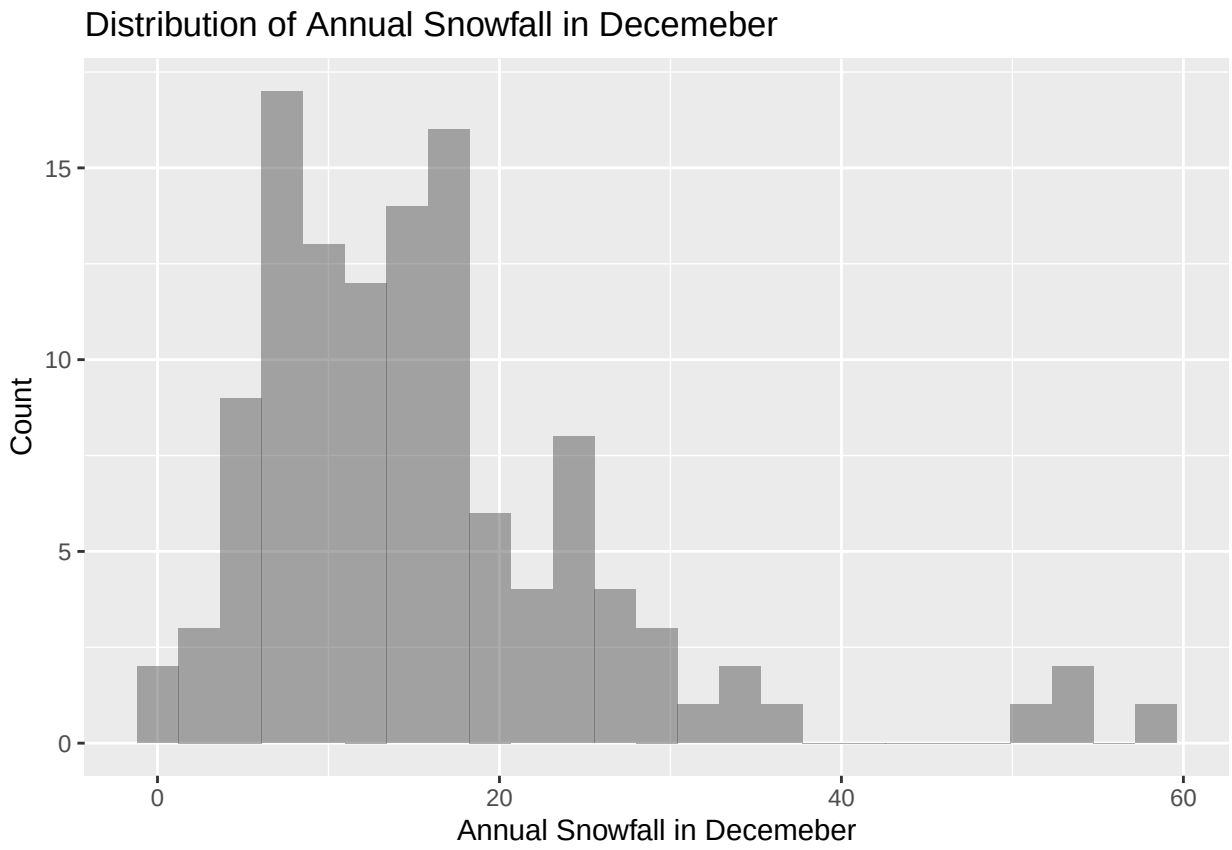
The bootstrap estimate of the test statistic is the empirical value of the test statistic from all of the $\hat{\theta}^{*b}$ replications. As B increases, $\hat{F}$ approaches F which means that the test statistic of interest approaches its true value as well.

We demonstrate performing a non-parametric bootstrap below using `SnowGR` which gives the official snowfall dataset by month for Grand Rapids, MI, starting in 1893. We will be using the `Dec` variable which gives the number of inches of snow that fell in December of each year. In particular, we are interested in the sample mean and later finding confidence intervals for population mean of `Dec`. The distribution is below.

```
data(SnowGR)
favstats(~ Dec, data = SnowGR)
```

```
##  min   Q1 median    Q3  max     mean       sd   n missing
##  0.8 8.35   13.8 18.65 59.2 15.75798 10.58382 119       0
```

```
gf_histogram(~ Dec, data = SnowGR) +
  labs(x = "Annual Snowfall in Decemeber",
       title = "Distribution of Annual Snowfall in Decemeber", y = "Count")
```

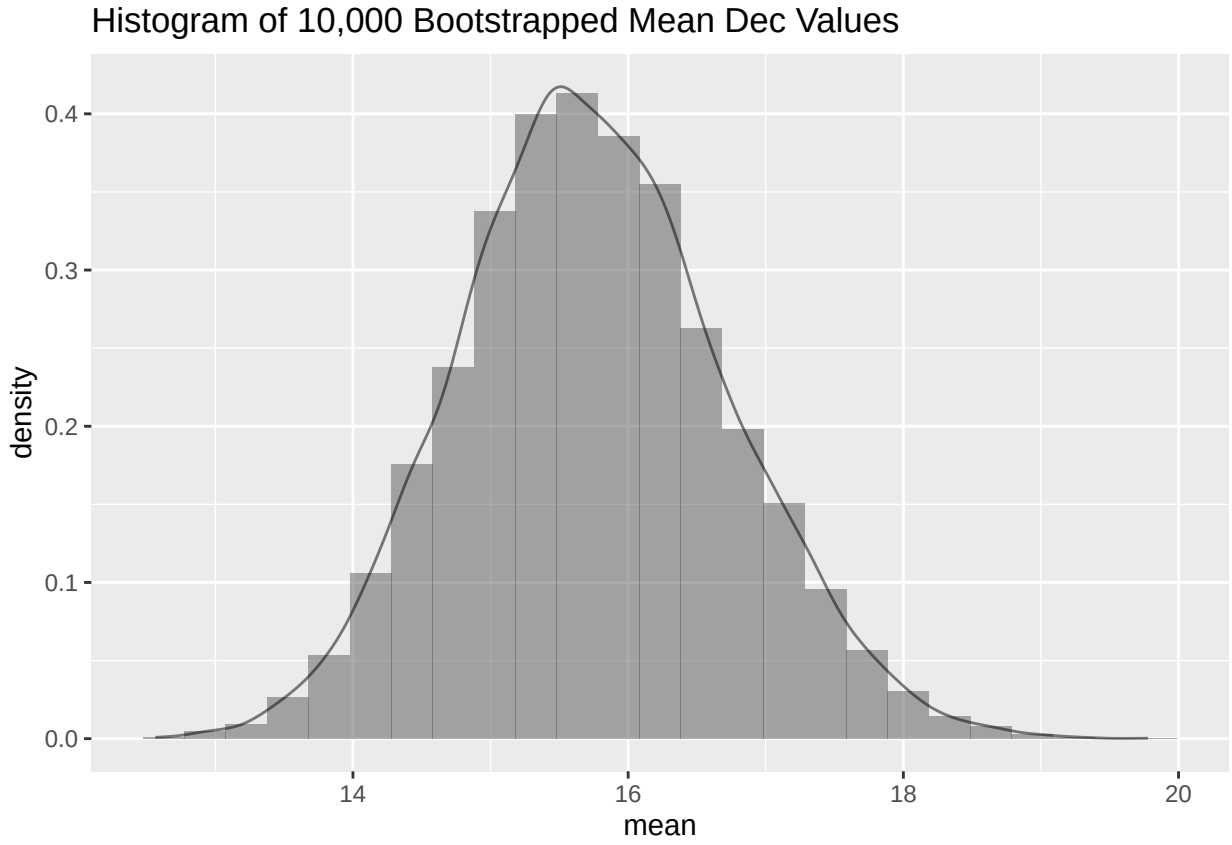


Next, we will perform the non-parametric bootstrap using 10,000 replications.

```
orig_mean <- mean(~ Dec, data = SnowGR) # theta hat

set.seed(495)
nboot <- 10000

mean <- rep(0,nboot)
sd <- rep(0,nboot)
se <- rep(0,nboot)
t_stat <- rep(0,nboot)

for (i in 1:nboot) {
  resampled <- as.data.frame(mosaic::resample(SnowGR, replace = TRUE))
  mean[i] <- mean(~Dec, data = resampled)
```

```
  sd[i] <- sd(~Dec, data = resampled)
}

dec_means <- as.data.frame(mean)
```

```
gf_dhistogram(~ mean, data = dec_means) %>%
  gf_dens() %>%
  gf_labs(title = "Histogram of 10,000 Bootstrapped Mean Dec Values")
```



2.2 Standard Confidence Interval

Confidence intervals are tools that are used to estimate a parameter. Specifically, a confidence interval gives a range of values in which the true value of the parameter may lie. An α-level standard confidence interval is given by

$$\hat{\theta}_S[\alpha] = \hat{\theta} \pm z_\alpha \hat{\sigma},$$

where $\hat{\theta}$ is a point estimate of the parameter of interest θ, $\hat{\sigma}$ is the estimate of the standard deviation of $\hat{\theta}$ and z_α is the $(100*\alpha)$th percentile of the normal deviation (Efron & Tibshirani 1986). We say that the confidence interval constructed in this manner has a chance of capturing the true parameter with a probability of α.

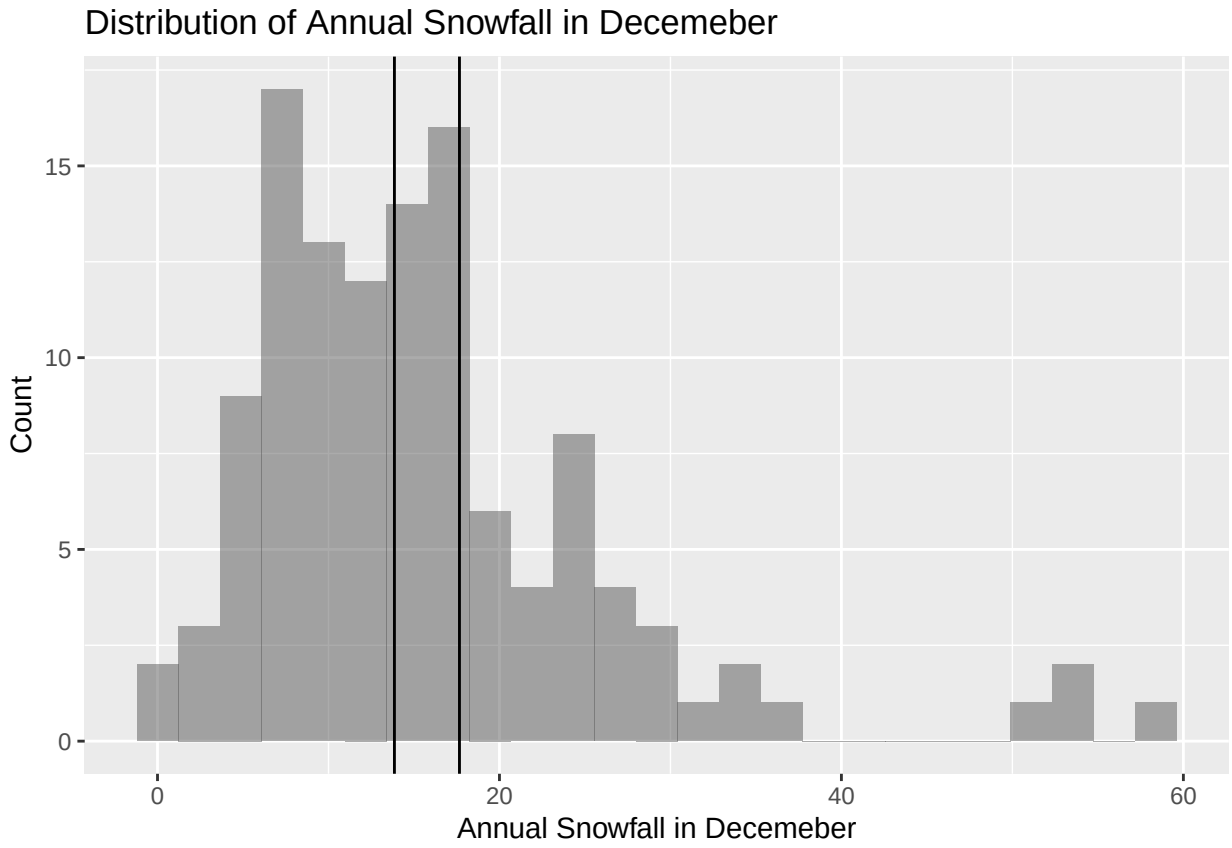
The standard confidence interval is built based on the assumption that the distribution from which we are sampling is Normal. This means that for an unknown distribution, the standard confidence interval could present an incorrect range. However, the same process can be used with bootstrap sampling to form the bootstrap percentile method. This means that an approximate bootstrap confidence interval will be created in the same automatic way that the standard confidence interval was created. For bootstrapped confidence intervals, the number of bootstrap replications B must be large (around 2000) due to the nature of confidence intervals requiring greater accuracy (Efron & Tibshirani 1986).

Using our example, to find a 95% confidence interval for the population mean of `Dec`, we would get the following:

```
mean(~ Dec, data = SnowGR) + qnorm(c(0.025, 0.975)) *
  sd(~ Dec, data = SnowGR)/sqrt(nrow(SnowGR))
```

```
## [1] 13.85639 17.65957
```

```
gf_histogram(~ Dec, data = SnowGR) +
  labs(x = "Annual Snowfall in Decemeber",
       title = "Distribution of Annual Snowfall in Decemeber", y = "Count") +
  geom_vline(aes(xintercept= 13.85639)) +
  geom_vline(aes(xintercept= 17.65957))
```



2.3 The Percentile Method

The percentile method interval is defined as the interval between the $100 * \alpha$ and the $100(1-\alpha)$ percentiles of the bootstrap distribution of $\hat{\theta}$. That is, the $(1-2\alpha)$ coverage interval can be defined as $[\hat{\theta}^*_\alpha, \hat{\theta}^*_{1-\alpha}]$ (Efron & Tibshirani 1986, Efron & Hastie (2021)). To go further, we can define $\hat{G}(t)$ as the bootstrap cdf, or the proportion of bootstrap samples less than t:

$$\hat{G}(t) = \frac{\#\{\hat{\theta^{*b}} \leq t\}}{B}.$$

Thus the αth percentile point of the distribution is given by

$$\hat{\theta}_p[\alpha] = \hat{\theta^*_\alpha} = \hat{G}^{-1}(\alpha).$$

It follows that the percentile interval can be represented as

$$\left[\hat{G}^{-1}(\alpha), \hat{G}^{-1}(1-\alpha)\right].$$

In the case that the bootstrap distribution of $\hat{\theta}^* \sim N(\hat{\theta}, \hat{\sigma}^2)$, the corresponding percentile interval would be equivalent to the standard interval. However, this is not usually the case. When the bootstrap distribution is non-normal, we can suppose that there exists, for all θ,

$$\hat{\phi} \sim N(\phi, \tau^2),$$

for some monotone transformation $\hat{\phi} = g(\hat{\theta}), \phi = g(\theta)$, and τ is a constant. In other words, this transformation perfectly normalizes the distribution of $\hat{\theta}$. This transformation invariant can be applied to the bootstrap replications such that

$$\hat{\phi}^{*b} = g\left(\hat{\theta}^{*b}\right) \text{ for } b = 1, 2, \ldots, B.$$

The corresponding percentiles of the distribution transform similarly, $\hat{\phi}^*_\alpha = g\left(\hat{\theta}^*_\alpha\right)$. Or we can say that the $(1 - 2\alpha)$ percentile interval is $\hat{\phi} \pm \tau z_\alpha$ which can also be represented as $[\hat{\phi}^*_\alpha, \hat{\phi}^*_{1-\alpha}]$. This means that the interval on the θ scale can be defined as

$$\hat{\theta}^*_\alpha = g^{-1}(\hat{\phi} \pm \tau z_\alpha).$$

This also can be represented as an interval,

$$\left[g^{-1}(\hat{\phi} \pm \tau z_{1-\alpha}), g^{-1}(\hat{\phi} \pm \tau z_\alpha)\right].$$

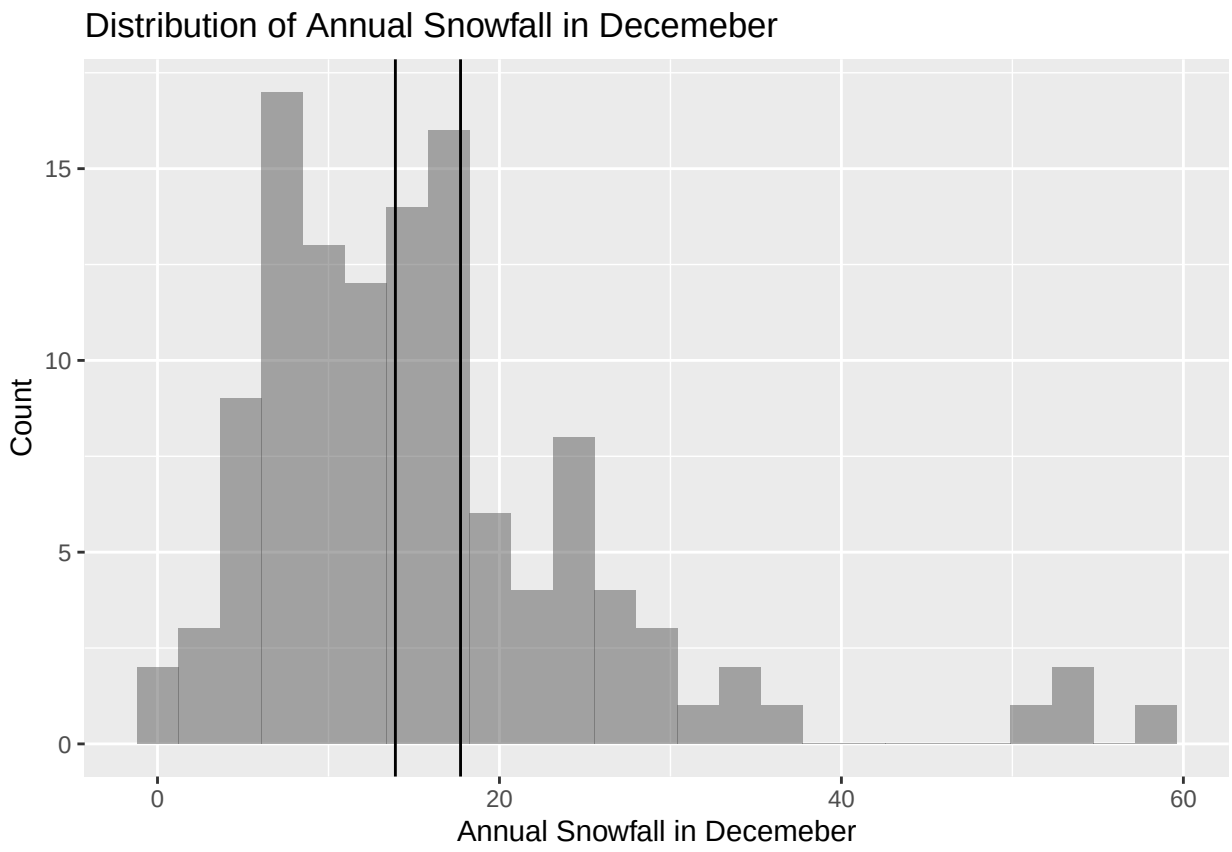
Therefore, the percentile method produces a correct interval for ϕ and due to the transformation invariance, also produces a correct percentile interval for θ. This method assumes the existence of some monotone normalizing mapping $\hat{\phi} = g(\hat{\theta}), \phi = g(\theta)$ and relies on that to create a correct interval. Since the process is automatic, we do not need to know the transformation itself, only that it exists. However, in some cases, no monotone normalizing mapping will exist (Efron & Tibshirani 1986).

Finding a 95% confidence interval for the true population mean of `Dec` using the percentile method, we get:

```
qdata(~ mean, c(0.025, 0.975), data = dec_means)
```

```
##     2.5%    97.5%
## 13.91170 17.72353
```

```
gf_histogram(~ Dec, data = SnowGR) +
  labs(x = "Annual Snowfall in Decemeber",
       title = "Distribution of Annual Snowfall in Decemeber", y = "Count") +
  geom_vline(aes(xintercept= 13.91170)) +
  geom_vline(aes(xintercept= 17.72353))
```



This interval is similar to the one created using the Standard method, but shifted slightly to the right.

2.4 The Bias-Corrected (BC) Method

The next method we will be looking at is the bias-corrected percentile method (BC method) which is an improvement upon the previous percentile method because we now take into account the possibility of bias. It can be shown that $\hat{\theta}$ is biased upwards relative to θ which means that the confidence intervals should be adjusted downwards (Efron & Tibshirani 1986, Efron & Hastie (2021)). From our simulated bootstrap replications $\hat{\theta}^{*1}, \hat{\theta}^{*2}, \ldots, \hat{\theta}^{*B}$,

define

$$p_0 = \frac{\#\{\theta^{*b} \hat{\leq} \theta\}}{B},$$

and define the bias-correction value

$$z_0 = \Phi^{-1}(p_0),$$

where Φ^{-1} is the inverse function of the standard normal cdf. Thus we define a transformation $\hat{\phi} = g(\hat{\theta}), \phi = g(\theta)$ such that for any θ,

$$\hat{\phi} \sim N(\phi - z_0\tau, \tau^2),$$

with z_0 and τ constants. This means that we can say the bias corrected method has an α-level endpoint can be represented as

$$\hat{\theta}_{BC}[\alpha] = \hat{G}^{-1}\left[\Phi\left(2z_0 + z_\alpha\right)\right].$$

If $\hat{G} = 0.50$, then half of the bootstrap distribution is less than $\hat{\theta}$ and our bias-correction value $z_0 = 0$. In this case, the confidence interval produced by BC would be the same interval produced by the percentile method.

Using this method to create a 95% confidence interval has a couple more steps because we need to find the bias-correction value. First we calculate the sample mean.

```
sample_mean <- mean(~Dec, data = SnowGR); sample_mean
```

```
## [1] 15.75798
```

Then we find the proportion of bootstrap replications that have a sample mean less than the original sample mean.

```
less_than_sample_mean <- sum(ifelse(dec_means <= sample_mean, 1, 0))/10000
less_than_sample_mean
```

```
## [1] 0.5208
```

From this, we can calculate the bias-correction value, z_0:

```r
z0 <- qnorm(less_than_sample_mean); z0
```

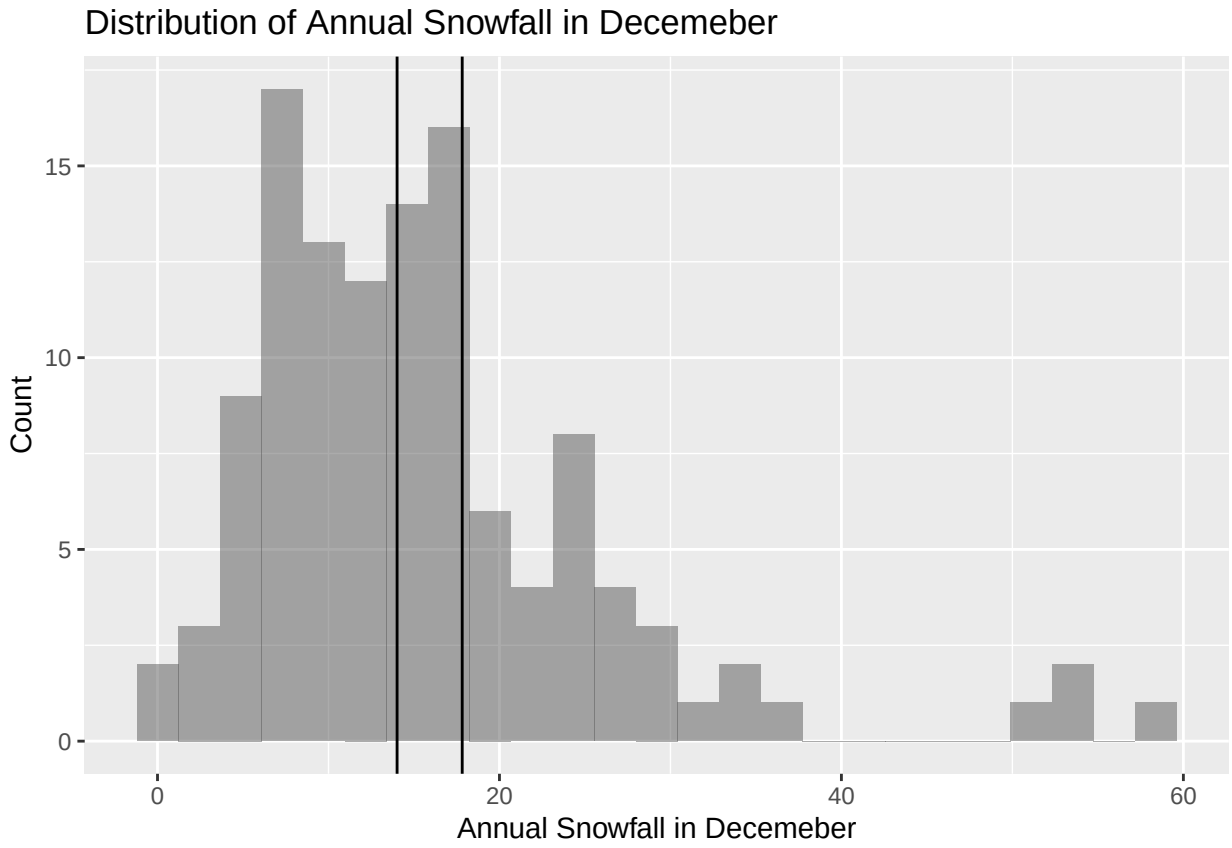
```
## [1] 0.05216151
```

Then we can find the modified percentiles:

```r
alphalow <- 0.025; alphahigh <- 0.975

newlow <- pnorm(2*z0 + qnorm(alphalow)); #newlow
newhigh <- pnorm(2*z0 + qnorm(alphahigh)); #newhigh

qdata(~ mean, c(newlow, newhigh), data = dec_means)
```

```
## 3.175238% 98.05047%
##  14.01172  17.82017
```

```r
gf_histogram(~ Dec, data = SnowGR) +
  labs(x = "Annual Snowfall in Decemeber",
       title = "Distribution of Annual Snowfall in Decemeber", y = "Count") +
  geom_vline(aes(xintercept= 14.01172)) +
  geom_vline(aes(xintercept= 17.82017))
```



2.5 The Bias-Corrected and Accelerated (BCa) Method

A further modification upon the BC interval is the bias corrected and accelerated method (BCa). For this method, we do not assume the the standard error, τ is constant as we did in the BC interval (Efron & Tibshirani 1986, Efron & Hastie (2021)). Rather, we assume the existence of a monotone transformation $\hat{\phi} = g(\hat{\theta}), \phi = g(\theta)$ such that for any θ,

$$\hat{\phi} \sim N(\phi - z_0\tau_\phi, \tau_\phi^2) \text{ where } \tau_\phi = 1 + a\phi.$$

The a is known as the acceleration and is a constant that describes how the standard deviation of $\hat{\phi}$ varies with ϕ. In other words, a is proportional to the skewness of the bootstrap distribution. For example, for one-parameter exponential families, $a = z_0$, however, there are many different algorithms to compute and estimate a (Flowers-Cano et al. 2018). Now, our α-level endpoint from BCa is

$$\hat{\theta}_{BCa}[\alpha] = \hat{G}^{-1}\left[\Phi\left(z_0 + \frac{z_0 + z_\alpha}{1 - a(z_0 + z_a)}\right)\right].$$

If $a = 0$, then $\hat{\theta}_{BCa}[\alpha] = \hat{\theta}_{BC}[\alpha]$. When calculating a BCa interval, the acceleration value a is not a function of the bootstrap distribution and must be calculated separately, however the process is algorithmic and can be calculated without too much work. Each of the three previous methods (percentile, BC, and BCa) all build upon each other and have less restrictive assumptions, however computation increases as we loosen assumptions.

In practice, we can use the jackknife procedure to estimate the acceleration value for this method. Thus, we will run the jackknife procedure:

```
theta <- function(x){mean(x)}
jackknife_results <- bootstrap::jackknife(SnowGR$Dec, theta)
```

We find the mean of the jackknife values below:

```
jackmean <- mean(~ jackknife_results$jack.values); jackmean
```

```
## [1] 15.75798
```

Then we can compute a.

```
estimated_a <- (1/6)*sum((jackknife_results$jack.values - jackmean)^3)/
  (sum((jackknife_results$jack.values - jackmean)^2))^(1.5); estimated_a
```

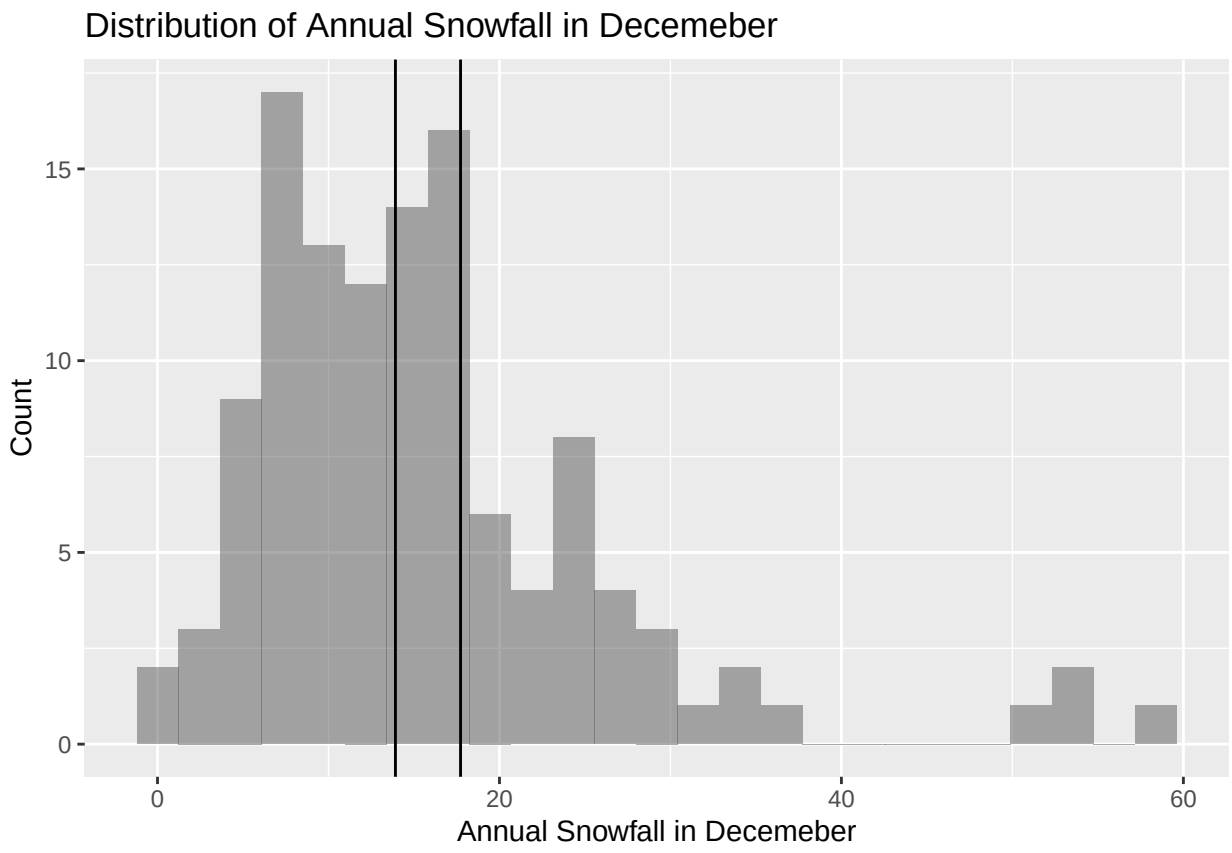
```
## [1] -0.02674911
```

Now, we can find the adjust percentiles and the bias-corrected with acceleration confidence interval.

```
a_newlow <- pnorm(z0 + (z0 + qnorm(alphalow))/
                    (1-estimated_a*(z0 + qnorm(alphalow)))); #a_newlow
a_newhigh <- pnorm(z0 + (z0 + qnorm(alphahigh))/
                     (1-estimated_a*(z0 + qnorm(alphahigh)))); #a_newhigh
qdata(~ mean, c(a_newlow, a_newhigh), data = dec_means)
```

```
## 2.510119% 97.50908%
##  13.91591  17.72353
```

```
gf_histogram(~ Dec, data = SnowGR) +
  labs(x = "Annual Snowfall in Decemeber",
       title = "Distribution of Annual Snowfall in Decemeber", y = "Count") +
  geom_vline(aes(xintercept= 13.91591)) +
  geom_vline(aes(xintercept= 17.72353))
```



2.6 The Bootstrap-t (Studentized) Method

The final method we explain is the bootstrap-t interval (or the studentized method). Recall from earlier, our sample, X from which we can calculate an estimate, $\hat{\theta}(X)$ of the parameter of interest θ (Efron & Tibshirani 1986, Puth et al. (2015)). We can also estimate $\hat{\sigma}(X)$ for the standard error of θ. We can use these parameters to find the Student's t-statistic

defined as

$$T = \frac{\hat{\theta} - \theta}{\hat{\sigma}}.$$

As such, the αth percentile point of a confidence interval of θ would be $\hat{\theta} - \hat{\sigma}T_{(\alpha)}$ where $T_{(\alpha)}$ represents the αth percentile of the t-distribution, T. Unfortunately, the percentiles of the t-distribution are unknown in most cases, but we can use bootstrapping to estimate these percentiles. To do this, we perform a large number, B, of bootstrap samples, from which we can find the bootstrap replications of the parameter of interest, $\hat{\theta^*} = \hat{\theta}(X^*)$, and the standard error, $\hat{\sigma^*} = \hat{\sigma}(X^*)$. From these we can calculate a t-statistic for each bootstrap sample:

$$T^* = \frac{\hat{\theta}^* - \hat{\theta}}{\hat{\sigma}^*}.$$

for each bootstrap sample. Using a large number of these bootstrap samples, we can estimate the percentiles of the t-distribution such that:

$$\hat{T}_{(\alpha)} = B * \alpha\text{th ordered value of all the bootstrap relications of } T^*.$$

This means that for $B = 2000$ and $\alpha = 0.90$, then $\hat{T}_{(\alpha)}$ is the 1,800th ordered point of all of the bootstrap replications of T^*. It follows that the αth studentized confidence interval endpoint can be given with

$$\hat{\theta}_T[\alpha] = \hat{\theta} - \hat{\sigma}T_{(\alpha)},$$

where we can estimate the standard error using

$$\hat{\sigma} = \frac{1 - \hat{\theta}^2}{\sqrt{n}}.$$

One of the main factors why the studentized method is so popular is because we assume that our bootstrapped statistic is pivotal which means that the confidence interval does not depend on any other parameters. Instead, we can calculate the appropriate confidence interval for the parameter of interest specifically from the bootstrapped statistic (Efron & Tibshirani 1986, Puth et al. (2015)).

Following the process above, we can use the following to get our studentized confidence interval:

```r
set.seed(495)

orig_mean <- mean(~ Dec, data = SnowGR) # theta hat
orig_se <- sd(~ Dec, data = SnowGR)/sqrt(nrow(SnowGR))

se <- rep(0,nboot)
t_stat <- rep(0,nboot)

for (i in 1:nboot) {
  se[i] <- sd[i]/sqrt(nrow(SnowGR))
  t_stat[i] <- (mean[i] - orig_mean)/se[i]
}

dec_t_stat <- as.data.frame(t_stat)

q <- unname(quantile(dec_t_stat$t_stat, c(0.025, 0.975)))
lower <- q[1]
upper <- q[2]

c(orig_mean - upper*orig_se, orig_mean - lower*orig_se)
```

```
## [1] 14.03467 18.01709
```

3 Simulation

For the simulation, we will randomly sample from distributions in which we know the true population mean. For example, we will sampling from a Gamma Distribution where we can find the true population mean using the shape and rate parameters of the sample distribution. After we sample from a distribution, we will create a bootstrapped confidence interval for each of the methods we described in the exposition. Since we know the true population mean of the distribution, we will determine if the true population parameter is

contained in the confidence interval for each of the methods. We will perform this process a large number of times (10,000) and then average by the number of iterations to determine the average proportion that each bootstrapped confidence interval method contains the true population mean. We will do this for a couple different distributions as well as a range of sample sizes show how the intervals perform with skewed distributions and larger sample sizes.

First, we create a function that calculates the bootstrapped confidence intervals from a given sample. To do this, we first bootstrap the sample and then perform each method.

```
findCIs <- function(sampleinput, nboot) {

  sample_mean <- mean(~ X, data = sampleinput)
  sample_sd <- sd(~ X, data = sampleinput)
  sample_n <- nrow(sampleinput)
  sample_se <- sample_sd/sqrt(sample_n)

  alphalow <- 0.025
  alphahigh <- 0.975

  #perform bootstrap
  mean <- rep(0,nboot)
  sd <- rep(0,nboot)
  se <- rep(0,nboot)
  t_stat <- rep(0,nboot)

  for (i in 1:nboot) {
    resampled <- as.data.frame(mosaic::resample(sampleinput, replace = TRUE))
    mean[i] <- mean(~ X, data = resampled)
    sd[i] <- sd(~ X, data = resampled)
    se[i] <- sd[i]/sqrt(nrow(resampled))
    t_stat[i] <- (mean[i] - sample_mean)/se[i]
```

```
}

boot_means <- as.data.frame(mean)
boot_t_stat <- as.data.frame(t_stat)

# find the standard confidence interval
standard_interval <- sample_mean +
  qnorm(c(0.025, 0.975)) * sample_sd/sqrt(sample_n)

# find percentile interval
percentile_interval <- qdata(~ mean, c(0.025, 0.975), data = boot_means)

# find BC interval
less_than_sample_mean <- sum(ifelse(boot_means <= sample_mean, 1, 0))/nboot
z0 <- qnorm(less_than_sample_mean)
newlow <- pnorm(2*z0 + qnorm(alphalow))
newhigh <- pnorm(2*z0 + qnorm(alphahigh))

bc_interval <- qdata(~ mean, c(newlow, newhigh), data = boot_means)

# find BCa interval
theta <- function(x){mean(x)}
jackknife_results <- bootstrap::jackknife(sampleinput$X, theta)
jackmean <- mean(~ jackknife_results$jack.values)
estimated_a <- (1/6)*sum((jackknife_results$jack.values - jackmean)^3)/
  (sum((jackknife_results$jack.values - jackmean)^2))^(1.5)
a_newlow <- pnorm(z0 + (z0 + qnorm(alphalow))/
                    (1-estimated_a*(z0 + qnorm(alphalow))))
a_newhigh <- pnorm(z0 + (z0 + qnorm(alphahigh))/
                     (1-estimated_a*(z0 + qnorm(alphahigh))))
```

```
  bca_interval <- qdata(~ mean, c(a_newlow, a_newhigh), data = boot_means)

  # find studentized interval
  t_q <- unname(quantile(boot_t_stat$t_stat, c(0.025, 0.975)))
  lower_q <- t_q[1]
  upper_q <- t_q[2]
  t_interval <- c(sample_mean - (upper_q*sample_se), sample_mean - (lower_q*sample_se))

  output <- list(standard_interval, percentile_interval, bc_interval,
                 bca_interval, t_interval)
  names(output) <- c("Standard Interval", "Percentile Interval", "BC Interval",
                     "BCa Interval", "Studentized Interval")
  output
}
```

Next, we will create the simulation that picks the random sample from a distribution and tests to see if the sample creates bootstrapped confidence intervals that contain the true population parameter. First, we have the simulation of selecting from a Normal Distribution. Since we have a $N(0, 1)$ distribution, out true population mean is 0.

```
run_simulation_normal <- function(simulation_rep){
  trueparameter <- 0

  X <- rnorm(sample_sizes, mean = 0, sd = 1)
  values <- data.frame(X)
  output <- findCIs(values, 2000)
  ci <- matrix(c(trueparameter >= output$`Standard Interval`[1] &
    trueparameter <= output$`Standard Interval`[2],
    trueparameter >= output$`Percentile Interval`[1] &
    trueparameter <= output$`Percentile Interval`[2],
    trueparameter >= output$`BC Interval`[1] &
```

```
    trueparameter <= output$`BC Interval`[2],
    trueparameter >= output$`BCa Interval`[1] &
    trueparameter <= output$`BCa Interval`[2],
    trueparameter >= output$`Studentized Interval`[1] &
    trueparameter <= output$`Studentized Interval`[2]), nrow = 1)
  res <- rbind(data.frame(), ci)
  res
}
```

Second, we select from a Gamma Distribution with a shape parameter of 1 and a scale parameter of 4. This means the true population parameter is 4.

```
run_simulation_gamma <- function(simulation_rep){
  trueparameter <- 4

  X <- rgamma(sample_sizes, shape = 1, scale = 4)
  values <- data.frame(X)
  output <- findCIs(values, 2000)
  ci <- matrix(c(trueparameter >= output$`Standard Interval`[1] &
    trueparameter <= output$`Standard Interval`[2],
    trueparameter >= output$`Percentile Interval`[1] &
    trueparameter <= output$`Percentile Interval`[2],
    trueparameter >= output$`BC Interval`[1] &
    trueparameter <= output$`BC Interval`[2],
    trueparameter >= output$`BCa Interval`[1] &
    trueparameter <= output$`BCa Interval`[2],
    trueparameter >= output$`Studentized Interval`[1] &
    trueparameter <= output$`Studentized Interval`[2]), nrow = 1)
  res <- rbind(data.frame(), ci)
  res
}
```

Third, we sample from a Log-Normal distribution with mean log of 0 and a mean standard deviation of 1. Therefore, the true population mean of this distribution is $e^{\frac{1}{2}}$.

```
run_simulation_log_normal <- function(simulation_rep){
  trueparameter <- exp(1/2)

  X <- rlnorm(sample_sizes, meanlog = 0, sdlog = 1)
  values <- data.frame(X)
  output <- findCIs(values, 2000)
  ci <- matrix(c(trueparameter >= output$`Standard Interval`[1] &
    trueparameter <= output$`Standard Interval`[2],
    trueparameter >= output$`Percentile Interval`[1] &
    trueparameter <= output$`Percentile Interval`[2],
    trueparameter >= output$`BC Interval`[1] &
    trueparameter <= output$`BC Interval`[2],
    trueparameter >= output$`BCa Interval`[1] &
    trueparameter <= output$`BCa Interval`[2],
    trueparameter >= output$`Studentized Interval`[1] &
    trueparameter <= output$`Studentized Interval`[2]), nrow = 1)
  res <- rbind(data.frame(), ci)
  res
}
```

Lastly, we sample from a Beta distribution with shape parameters of 4 and 1 which gives us a true population mean of 0.8.

```
run_simulation_beta <- function(simulation_rep){
  trueparameter <- 0.8

  X <- rbeta(sample_sizes, shape1 = 4, shape2 = 1)
  values <- data.frame(X)
  output <- findCIs(values, 2000)
```

```
  ci <- matrix(c(trueparameter >= output$`Standard Interval`[1] &
    trueparameter <= output$`Standard Interval`[2],
    trueparameter >= output$`Percentile Interval`[1] &
    trueparameter <= output$`Percentile Interval`[2],
    trueparameter >= output$`BC Interval`[1] &
    trueparameter <= output$`BC Interval`[2],
    trueparameter >= output$`BCa Interval`[1] &
    trueparameter <= output$`BCa Interval`[2],
    trueparameter >= output$`Studentized Interval`[1] &
    trueparameter <= output$`Studentized Interval`[2]), nrow = 1)
  res <- rbind(data.frame(), ci)
  res
}
```

After that, these functions will compile all of the information we have created from the previous methods and run the simulations of each method for a variety of sample sizes.

```
collect_data_gamma <- function(sim_reps) {
  data2 <- matrix(nrow = 5, ncol = 6)
  sample_sizes = 50
  for (i in 1:5) {
    simulation_rep <- seq(1,sim_reps)
    results <- mclapply(simulation_rep, run_simulation_gamma, mc.cores = detectCores())
    df <- data.frame(matrix(unlist(results), nrow=length(results), byrow=TRUE))
    ci_prop <- unname(colSums(df)/sim_reps)

    data2[i, 1] <- sample_sizes
    data2[i, 2:6] <- ci_prop
    sample_sizes <- sample_sizes*2
  }
```

```
  gamma_output <- as.data.frame(data2) %>%
    rename("Sample_Size" = V1, "Standard" = V2, "Percentile" = V3, "BC" = V4,
           "BCa" = V5, "Studentized" = V6)
  gamma_output
}
```

```
collect_data_log_normal <- function(sim_reps) {
  data2 <- matrix(nrow = 5, ncol = 6)
  sample_sizes = 50

  for (i in 1:5) {
    simulation_rep <- seq(1,sim_reps)
    results <- mclapply(simulation_rep, run_simulation_log_normal, mc.cores = detectCore
    df <- data.frame(matrix(unlist(results), nrow=length(results), byrow=TRUE))
    ci_prop <- unname(colSums(df)/sim_reps)

    data2[i, 1] <- sample_sizes
    data2[i, 2:6] <- ci_prop
    sample_sizes <- sample_sizes*2
  }

  log_normal_output <- as.data.frame(data2) %>%
    rename("Sample_Size" = V1, "Standard" = V2, "Percentile" = V3, "BC" = V4,
           "BCa" = V5, "Studentized" = V6)
  log_normal_output
}
```

```
collect_data_normal <- function(sim_reps) {
  data2 <- matrix(nrow = 5, ncol = 6)
  sample_sizes = 50
```

```
  for (i in 1:5) {
    simulation_rep <- seq(1,sim_reps)
    results <- mclapply(simulation_rep, run_simulation_normal,
                        mc.cores = detectCores())
    df <- data.frame(matrix(unlist(results), nrow=length(results), byrow=TRUE))
    ci_prop <- unname(colSums(df)/sim_reps)

    data2[i, 1] <- sample_sizes
    data2[i, 2:6] <- ci_prop
    sample_sizes <- sample_sizes*2
  }

  normal_output <- as.data.frame(data2) %>%
    rename("Sample_Size" = V1, "Standard" = V2, "Percentile" = V3, "BC" = V4,
           "BCa" = V5, "Studentized" = V6)
  normal_output
}
```

```
collect_data_beta <- function(sim_reps) {
  data2 <- matrix(nrow = 5, ncol = 6)
  sample_sizes = 50

  for (i in 1:5) {
    simulation_rep <- seq(1,sim_reps)
    results <- mclapply(simulation_rep, run_simulation_beta,
                        mc.cores = detectCores())
    df <- data.frame(matrix(unlist(results), nrow=length(results), byrow=TRUE))
    ci_prop <- unname(colSums(df)/sim_reps)

    data2[i, 1] <- sample_sizes
```

```
    data2[i, 2:6] <- ci_prop
    sample_sizes <- sample_sizes*2
  }

  beta_output <- as.data.frame(data2) %>%
    rename("Sample_Size" = V1, "Standard" = V2, "Percentile" = V3, "BC" = V4,
           "BCa" = V5, "Studentized" = V6)
  beta_output
}
```

```
set.seed(495)
gamma_data1 <- collect_data_gamma(10000)
write.csv(gamma_data1,"gamma_data1_10000.csv", row.names = FALSE)

normal_data1 <- collect_data_normal(10000)
write.csv(normal_data1,"normal_data1_10000.csv", row.names = FALSE)

log_normal_data1 <- collect_data_log_normal(10000)
write.csv(log_normal_data1,"log_normal1_data_10000.csv", row.names = FALSE)

beta_data1 <- collect_data_beta(10000)
write.csv(beta_data1,"beta_data1_10000.csv", row.names = FALSE)
```

```
    gamma_data1 <- gamma_data5 %>%
      pivot_longer(cols = 2:6, names_to = "method", values_to = "prop")

    gamma_fig <- ggplot(data = gamma_data1, aes(x = Sample_Size, y = prop,
                                                color = method))+
      geom_point() +
      geom_line() +
      labs(title = "Proportion of CIs that Contain the True Parameter
```

```
          from Gamma Distibution",
          x = "Sample Size", y = "Proportion (of xyz Simulations)", color = "Method") +
    theme_light()

gamma_table <- gamma_data %>%
    kable(booktabs = TRUE, align = "c", caption = "gamma", col.names = gsub("[_]", " "

gamma_table
gamma_fig
```

References

Efron, B. & Hastie, T. (2021), *Computer Age Statistical Inference, Student Edition: Algorithms, Evidence, and Data Science*, Vol. 6, Cambridge University Press.

Efron, B. & Tibshirani, R. (1986), 'Bootstrap methods for standard errors, confidence intervals, and other measures of statistical accuracy', *Statistical Science* **1**(1), 54–75.
URL: *http://www.jstor.org/stable/2245500*

Flowers-Cano, R. S., Ortiz-Gómez, R., León-Jiménez, J. E., López Rivera, R. & Perera Cruz, L. A. (2018), 'Comparison of bootstrap confidence intervals using monte carlo simulations', *Water* **10**(2).
URL: *https://www.mdpi.com/2073-4441/10/2/166*

Puth, M.-T., Neuhäuser, M. & Ruxton, G. D. (2015), 'On the variety of methods for calculating confidence intervals by bootstrapping', *Journal of Animal Ecology* **84**(4), 892–897.
URL: *https://besjournals.onlinelibrary.wiley.com/doi/abs/10.1111/1365-2656.12382*