Notes 09 - Using infer for Confidence Intervals

STS 2300 Introduction to Data Analytics

Updated: 2025-04-03

Table of Contents

[Reading for Notes 09 1](#_Toc194609329)

[Learning Goals for Notes 09 1](#_Toc194609330)

[The infer package 2](#_Toc194609331)

[House of Representatives Example 4](#_Toc194609332)

[Three approaches to confidence intervals 6](#_Toc194609333)

[Theory-based approach 6](#_Toc194609334)

[The Standard Error Method 6](#_Toc194609335)

[The percentile method 7](#_Toc194609336)

[Comparing the methods 8](#_Toc194609337)

[Graphing confidence intervals 8](#_Toc194609338)

[Revisiting the Learning Goals for Notes 09 9](#_Toc194609339)

# Reading for Notes 09

Read [Section 8.4](https://moderndive.com/8-confidence-intervals.html#bootstrap-process) of the Modern Dive textbook.

# Learning Goals for Notes 09

* Understand the general steps of using the infer package for creating bootstrap distributions.
* Understand how the percentile method and the standard error method can be used to create confidence intervals with bootstrap distributions (and how these compare to theory-based methods)

I’ll be using the following packages in this set of notes, so I’ll load them before I get started.

library(infer)  
library(dplyr)  
library(ggplot2)

# The infer package

We will use the infer package to help us do the following

* generate bootstrap resampling distributions,
* visualize those distributions, and
* calculate confidence intervals based on those distributions.

There are four basic steps in the first two bullet points of this process. Each step is associated with a different function in the infer package.

1. specify()

First we want to **specify** what variable or relationship we’re studying. I will use what is known as formula notation to provide some consistency between different examples. You’ll notice this is similar to what we did with the lm() function when creating linear regression models. The code to specify this relationship will look something like:

data |>   
 specify(response ~ explanatory)

The response variable will be the one we’re studying or interested in, and the explanatory variable will be the one that helps explain differences in the response variable. In many cases, this will be the variable that splits our data into groups.

If our response variable is categorical, we will also need to specify which level of the response variable we are interested in using the success argument. This will look like specify(response ~ explanatory, success = "\_\_\_").

When we run just these lines, it won’t look like much is changing, but R will be storing information about our analysis plan as “metadata”.

1. generate()

In this step, we will use resampling to **generate** many new samples *with replacement* from our original sample. This is what we previously did with the rep\_sample\_n() function. The generate() function will have two main arguments, reps and type.

* The reps argument controls how many new samples we generate. We will use 1,000 for most examples in our class, but technically you could use more or less if you wanted. We want enough repetitions that we get a clear shape for our distribution but not so many that it takes a long time to run.
* The type argument describes how to do the resampling. We will stick with using the bootstrap method for now.

Below is some generic code to generate 1,000 samples with replacement from our original data.

data |>   
 specify(response ~ explanatory) |>   
 generate(reps = 1000, type = "bootstrap")

The output of this will be a data frame with 1,000 times as many rows as our original sample. It will contain all 1,000 of our new samples from the re-sampling process. A variable called replicate is added to the data frame to denote which rows belong to which sample.

1. calculate()

In this step, we will **calculate** a summary statistic that we want to use to estimate our parameter of interest (e.g., a sample proportion or a difference between two sample means). This step completes our process of generating a bootstrap distribution based on our original sample. Our created object will now have 1000 rows (or whatever we entered for reps).

bootstrap\_dist <- data |>   
 specify(response ~ explanatory) |>   
 generate(reps = 1000, type = "bootstrap") |>   
 calculate(stat = "ourstat")

There are many statistics we can calculate using the calculate() function. We will learn more in later sets of notes (or you can look at the help file for the calculate() function).

1. visualize()

Once we have our bootstrap distribution of resamples we can **visualize** the results in a histogram. This function takes the place of having to use ggplot() and geom\_histogram() every time. The only thing we need to put into the function is the object containing our bootstrap distribution.

visualize(bootstrap\_dist)

Later, when we’ve calculated confidence intervals, there are ways to add those to our graph if we’d like. The visualize() function has some similar arguments as geom\_histogram() that we can alter (like fill and color). We can also add other ggplot2 geoms to the graph (like labs(), geom\_vline(), or a theme) if we’d like to supplement our graph.

# House of Representatives Example

Let’s test this out on our House of Representatives example from Notes 08. Below is code to read our population into R, to take a sample of 30 seats from the House of Representatives, and to view the party breakdown in our sample.

house\_of\_reps <- read.csv("https://raw.githubusercontent.com/nbussberg/STS2300-Spring2025/refs/heads/main/Data/house\_of\_reps.csv")  
  
set.seed(82720)  
mysamp <- sample\_n(house\_of\_reps, size = 30)  
table(mysamp$party)

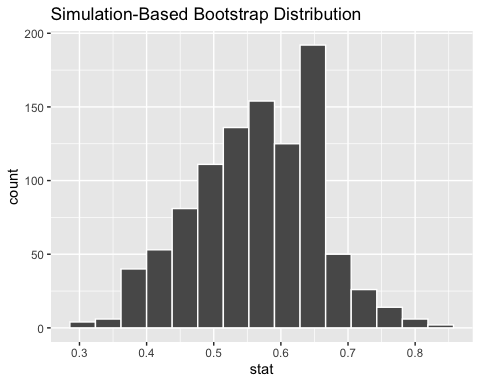
##   
## Democratic Republican   
## 17 13

Next let’s take our code from above. How can we customize this for our specific example?

bootstrap\_dist <- data |>   
 specify(response ~ explanatory) |>   
 generate(reps = 1000, type = "bootstrap") |>   
 calculate(stat = "ourstat")  
visualize(bootstrap\_dist)

* First, we might choose to rename bootstrap\_dist to be more specific. We’ll need to replace data with the object containing our sample.
* In the specify() function:
  + the variable we are interested in (our response) is party.
  + We don’t have any other variables explaining someone’s party, so we replace explanatory with NULL.
  + Since our response is categorical, we need to tell the function which category we are interested in by using the success = "\_\_" argument. Here we can replace the blank with “Democratic”
* We don’t need to change anything in the generate() function.
* In the calculate() function, we need to calculate a proportion, so we replace “ourstat” with “prop”.
* If we changed the name of our object, we will put the new name inside the visualize() function.

Once we do all of this, we should get something like this:



**Note:** If your sample included some Vacant seats, you may have gotten an error telling you that we need to have only two categories for our response variable. We can get around this by using the ifelse() function to update the party variable to only consist of “Democratic” and “Not Democratic” categories. This function has three arguments:

* a condition to check (whether party is equivalent to “Democratic”),
* what to do if the condition is true (assign “Democratic” as the value), and
* what to do if the condition is false (assign “Not Democratic” as the value).

Below is an example:

mysamp <- mysamp |>   
 mutate(party = ifelse(party == "Democratic", "Democratic", "Not Democratic"))

# Three approaches to confidence intervals

## Theory-based approach

In previous statistics classes, you likely focused on theory-based methods for confidence intervals. These methods require certain assumptions for you to be able to say that the sampling distribution follows a well known theoretical distribution. Theory-based confidence intervals tend to have three pieces:

For example, a theory-based 90% confidence interval for a population proportion would look like:

In these intervals, the estimate is our best guess for the population proportion. The critical value is based on how confident we want to be that our interval contains the true value of interest. The critical value comes from a theoretical sampling distribution. Recall that the standard error is how much our estiamtes vary from sample to sample.

When the assumptions required for the theory-based interval above are true, the method works because our sample proportion will be within standard errors 90% of the time.

For my data above, my interval would look like:

phat <- mean(mysamp$party == "Democratic")  
phat

## [1] 0.5666667

ci\_theory <- c(phat - 1.645 \* sqrt(phat \* (1 - phat) / 30),  
 phat + 1.645 \* sqrt(phat \* (1 - phat) / 30))  
ci\_theory

## [1] 0.4178402 0.7154931

We will learn later how to calculate theory-based intervals using various R functions.

## The Standard Error Method

Remember from our previous notes that the bootstrap distribution is like our best guess at the sampling distribution. An alternative approach to creating a confience interval is to use our bootstrap distribution to approximate the standard error in the equation above. Essentially we are just replacing with the standard deviation of our bootstrap distribution.

**Note**: This method should only be used if our bootstrap distribution is bell-shaped since we are using a critical value from a standard normal distribution.

For my data above, my interval would look like:

boot\_sd <- sd(HOR\_boot$stat)  
  
c(phat - 1.645 \* boot\_sd, phat + 1.645 \* boot\_sd)

## [1] 0.4160464 0.7172870

We can use the get\_ci() function from the infer package to do this calculation for us. We need to input our bootstrap distribution and specify level, type, and point\_estimate arguments. The level argument will be how confident we want to be (e.g. 0.95 for 95%), the type would be “se”, and the point\_estimate is our sample proportion.

ci\_se <- HOR\_boot |>   
 get\_ci(level = 0.9, type = "se", point\_estimate = phat)  
ci\_se

## # A tibble: 1 × 2  
## lower\_ci upper\_ci  
## <dbl> <dbl>  
## 1 0.416 0.717

Notice that I get **very** similar endpoints to the theory-based method. This is because both methods are essentially doing the same thing. The only difference is how they estimate the standard error for the sampling distribution.

## The percentile method

A third approach to creating confidence intervals is using our bootstrap distribution along with something called the percentile method. This has the added bonus that it can be used when the bootstrap distribution is not normally distributed. To use this method, we select the middle \_\_% of our distribution as our confidence interval. Again, this can be done with the get\_ci() function by specifying type = "percentile". When using this method, we do not need to specify a point estimate because we aren’t using a formula like we are for the two previous methods. This method also has the advantage of being applicable in some cases where unmet assumptions might prevent us from using a different method.

ci\_perc <- HOR\_boot |>   
 get\_ci(level = 0.9, type = "percentile")  
ci\_perc

## # A tibble: 1 × 2  
## lower\_ci upper\_ci  
## <dbl> <dbl>  
## 1 0.432 0.7

Once again, I get pretty similar answers.

## Comparing the methods

Putting this all together, we can see that in this case all three of our intervals were pretty similar.

## method lower\_ci upper\_ci  
## 1 Theory-based 0.418 0.715  
## 2 Standard Error 0.416 0.717  
## 3 Percentile 0.432 0.700

The theory-based and standard error methods will always be centered on our estimate of our parameter (e.g., sample proportion). These two methods both come up with estimates for the standard error based on our sample (either using a formula or using our bootstrap distribution).

The percentile method, like the standard error method, uses our bootstrap distribution to guess how far statistics are likely to be from the true parameter, but it uses percentiles instead of the standard deviation of the bootstrap distribution to create the interval.

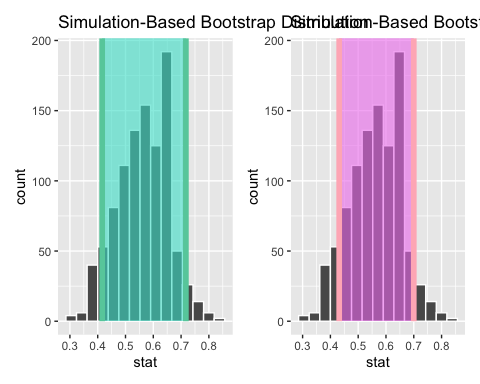
# Graphing confidence intervals

If you use either the standard error or percentile method, you can add the interval to your bootstrap distribution graph by adding a shade\_ci() function to your visualize() function. For example:

library(patchwork)

## Warning: package 'patchwork' was built under R version 4.3.3

se <- visualize(HOR\_boot) +  
 shade\_ci(ci\_se)  
perc <- visualize(HOR\_boot) +  
 shade\_ci(ci\_perc, fill = "violet", color = "lightpink")  
  
se + perc



In the next couple sets of notes we will talk specifically about how to make confidence intervals for different parameters with each method along with how to interpret those intervals.

# Revisiting the Learning Goals for Notes 09

* Understand the general steps of using the infer package for creating bootstrap distributions.
* Understand how the percentile method and the standard error method can be used to create confidence intervals with bootstrap distributions (and how these compare to theory-based methods)