Basic intro to bootstrapping in R

Rosie Aboody

2022-07-11 23:21:36

Contents

Bootstrapping over the mean . . . 1
What's the logic? . . . 1
But how do we actually do it? . . . 2
How do we plot it? . . . 5
How do we interpret it? . . . 6
Additional example: Bootstrapping over a correlation . . . 6
Difference bootstraps . . . 7
What's the logic? . . . 7
How do we do it? . . . 8
How do we interpret it? . . . 9
Additional example: bootstrapping over a correlation difference . . . 10
Other resources: tidyboot! . . . 11
References . . . 12

Setup: load bootstrap package & tidyverse (for plotting). Install the packages first if you don't have them.

```
install.packages("boot")
install.packages("tidyverse")
install.packages("tidyboot")
install.packages("glue")

library(boot)
library(tidyverse)
library(tidyboot)
library(glue)
```

Bootstrapping over the mean

What's the logic?

The basic logic behind bootstrapping is pretty simple. Let's say you have a bunch of data. Often as developmentalists this data is binary, so for our main example let's use binary data—but note that all the same holds for continuous data (see the extra example below). Maybe you ran 30 kids in a binary choice task, and 24 picked the right answer and 6 picked the wrong answer. Assuming this result is *representative* of the true world state (that 80% of all kids in the world would get your task right), we can simulate what might happen if we re-ran this experiment a bunch of times. . . just re-sampling from the data we already got!

To do that, we basically say: Let's put each one of our data points in a bag, and draw 30 new data points. Importantly, this sampling is *with replacement* (once we draw a data point we put it back in the bag), so the sample you get is going to be representative of your initial sample, but with some amount of noise. If your effect is large, and LOTS more 1's than 0's went into the bag, then you'll probably get more 1's than 0's most times you draw a new sample of participants. To get a really good sense of the range of results you could have gotten if you repeated your experiment (again, assuming that 80% of kids actually succeed on

your task), we can re-sample this way a bunch of times, and record the average proportion of kids succeeding in each sample. These averages are our bootstrapped means. The confidence interval just gives you the range (min and max) of the middle 95% of these means (so, cutting off the lowest 2.5% and the highest 2.5%). In essence, this range is telling you: If you re-ran your study again, what is the range of results (here, the range of means) you'd expect to get 95% of the time?

There is some variation in whether people choose 1,000, 10,000, or even 100,000 replications, but I generally resample 10,000 times. 10,000 samples is enough to get a good idea of the range and distribution of means you'd get if you replicated your experiment (assuming your result is representative).

But how do we actually do it?

The code is actually really simple! The `boot` package is going to take care of pretty much everything for us. Specifically, the `boot` package has a `boot()` function. This function takes in a data frame with your data, and generates a random set of numbers, with replacement, ranging from 1:n (n being the number of data points you have). To get a random sample from our data, we can use these numbers as indices. So each number corresponds to a row in your data frame (or position in your vector), and by taking the data at each index the `boot()` function provides, we can build a new sample from your original data. And then we just calculate the mean of that sample. And then we do it a bunch more times to generate the range of means you might get if you repeated your experiment many times (assuming your data is representative of the population).

At its most basic, the `boot()` function requires three inputs. First, your data. Second, a function to take the indices it provides, use that to sample from your data, and calculate the mean. And finally, the number of resamples you want to take. As mentioned above, 10,000 is my standard.

The first step is easy. Let's generate our fake data as a data frame:

```
our_data_frame = data.frame("fours" = c(rep(0, 6), rep (1, 24)), "age" =  4,
                            "month_tested" = "december")
head(our_data_frame, 10)
```

```
##    fours age month_tested
## 1      0   4     december
## 2      0   4     december
## 3      0   4     december
## 4      0   4     december
## 5      0   4     december
## 6      0   4     december
## 7      1   4     december
## 8      1   4     december
## 9      1   4     december
## 10     1   4     december
```

The second step is actually really easy too. All we need is a function that takes in your data, and a list of indices (which will be provided by the `boot()` function for us), selects the data that's at the provided indices, and returns the mean.

Here is the code—you can see that it's only a few lines! In the first line, we define a function, `GetMean`, which takes in two argments: your data, and a list of indices (provided by the `boot()` function). On the second line, we create a new data sample, by indexing your original data according to the random indices the `boot()` function has selected for us. And finally, on the third line we return the mean of that new sample.

```
Get_Mean = function(my_data, indices_from_boot){
  data_sample = my_data[indices_from_boot]
  return(mean(data_sample))
}
```

Finally, we need to set our seed. Any time you want R to do anything involving random numbers, you always

want to set your seed. Sampling truly random numbers is hard, so R only usually samples pseudo-random numbers. This is really good though, because setting the seed essentially tells R where to start sampling numbers from its list of pseudo-random numbers, ensuring everyone else's "random" sampling is the same as yours. So the indices the `boot()` function will sample will be the same for everyone running your code, thus making sure that your results are reproducible (this might not make sense yet, but just continue reading and hopefully it will!)

You can choose any seed you want.

```
set.seed(2020)
```

So now we have data, and we have a function that can select our data at the provided indices, and return the mean, and we set the seed so our results are reproducible. Now we just need to plug those things into the boot function!

Let's do it! Note that if we use a dataframe, we have to tell R which column contains the data. If you don't, you'll get an error.

```
data_frame_bootstrapsamples = boot(our_data_frame$fours,Get_Mean,R=10000)
```

If we look at the first few rows, you can see that we have a bunch of means. Each one was generated from one resample of our data.

```
head(data_frame_bootstrapsamples$t)
```

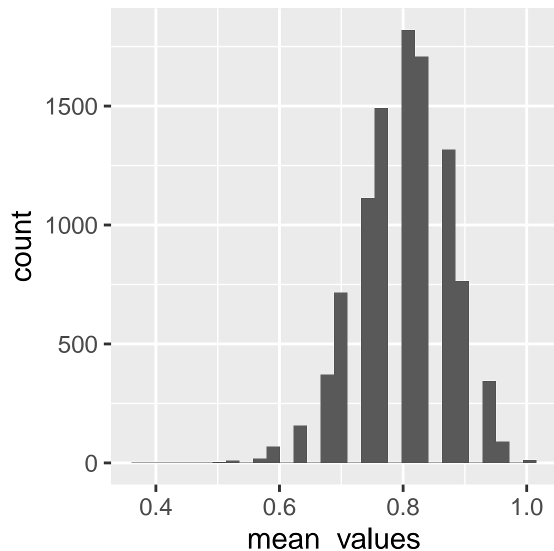
```
##           [,1]
## [1,] 0.8666667
## [2,] 0.8333333
## [3,] 0.6666667
## [4,] 0.8333333
## [5,] 0.7666667
## [6,] 0.8333333
```

If we're curious, we can plot these means. Since our effect is pretty large (with 80% of kids passing) we should see that the majority of the sample means fall within a reasonable range of 80%

```
sample_means = data.frame("mean_values" = data_frame_bootstrapsamples$t)

ggplot(sample_means, aes(x = mean_values)) +
  geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Right off the bat, it should be clear that pretty much every sample mean was between 60% and 100%.

The last thing we need to do is cut off the top and bottom 2.5% of means, to produce the 95% confidence interval range we care about. We can just use the `boot.ci()` function to do this for us!

```
boot.ci(data_frame_bootstrapsamples, type = "basic")
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 10000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = data_frame_bootstrapsamples, type = "basic")
##
## Intervals :
## Level      Basic
## 95%   ( 0.6667,  0.9667 )
## Calculations and Intervals on Original Scale
```

Note that we asked for a "basic" 95% confidence interval. There are a few different kinds you can calculate. I think basic is pretty common but they should all give you pretty much the same answer, so I'd pick one and stick to it. If you write `"all"` instead of `"basic"`, it will compute all 5 types of confidence intervals. If you want to do this just to play around with it, that's great, but then choose a type a priori before running it on any real data.

Putting all this together in one place:

```
Get_Mean = function(myData, selectedIDs){
  tempdata = myData[selectedIDs]
  return(mean(tempdata))
}

data_frame_bootstrapsamples = boot(our_data_frame$fours,GetMean,R=10000)
boot.ci(data_frame_bootstrapsamples, type = "basic")
```

Although I kept saying earlier that the boot function provides indices for a data frame, R also knows how to index your data if you don't make a proper data frame out of it but just use a vector with no rows. Usually your data will be in the form of a data frame (like if you loaded it from a csv file), but using a vector is great for lazy calculations (like if you're just prepping a quick plot for a meeting and just have the number of kids who succeeded and failed). I'll show you the vector now so you can see it works:

Creating a vector of just successes and failures:

```
our_vector = c(rep(0, 6), rep(1,24))
our_vector
```

```
##  [1] 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Plugging them into our function:

```
vector_bootstrapsamples = boot(our_vector,Get_Mean,R=10000)
head(vector_bootstrapsamples$t)
```

```
##           [,1]
## [1,] 0.8000000
## [2,] 0.8333333
## [3,] 0.7333333
## [4,] 0.9000000
## [5,] 0.7333333
## [6,] 0.9000000
```

```
boot.ci(vector_bootstrapsamples, type = "basic")
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 10000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = vector_bootstrapsamples, type = "basic")
##
## Intervals :
## Level      Basic
## 95%   ( 0.6667,  0.9333 )
## Calculations and Intervals on Original Scale
```

Or if you're lazy like me, you can wrap the last 2 lines into their own function. So then anywhere you have to run a bootstrap you can do it in one line instead of two:

```
Get_Mean = function(my_data, indices_from_boot){
  data_sample = my_data[indices_from_boot]
  return(mean(data_sample))
}

Get_Mean_Bootstrap = function(my_data){
  bootstrap_samples = boot(my_data,Get_Mean,R=10000)
  return(boot.ci(bootstrap_samples, type = "basic"))
}

Get_Mean_Bootstrap(our_vector)
```

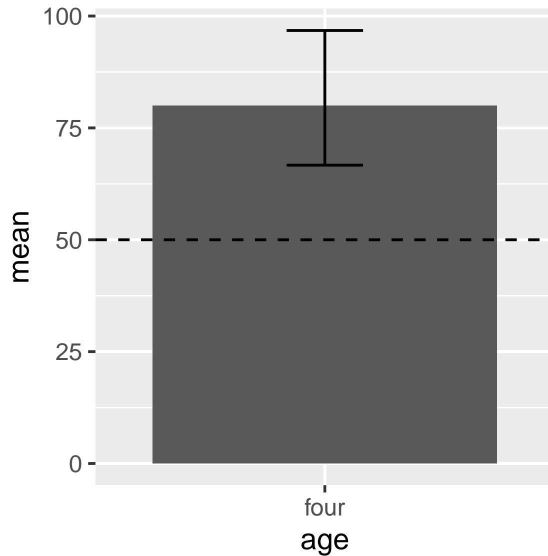
How do we plot it?

Easily! When I have to generate a bunch of bootstrapped confidence intervals, I'll sometimes write a function to do it for me (on top of all the other functions). But if it's only a few, here's how you can manually enter the values.

```
plot_data = data.frame(age = "four", mean = 80, upper = 96.77, lower = 66.67)

ggplot(plot_data, aes(x = age, y = mean, ymin = upper, ymax = lower))  +
  geom_bar(stat = "identity") +
```

```
  geom_errorbar(width = 0.2) +
  geom_hline(yintercept = 50, linetype = "dashed")
```



How do we interpret it?

Pretty simple too! If the confidence interval is completely above 50% (chance) and does not cross it, we say kids were reliably above chance. If the confidence interval is completely below chance and does not cross it, we say kids were reliably below chance.

Remember: bootstrapping gets rid of a lot of the assumptions of parametric tests... but its one *big* assumption is that your data is representative of the world state. It's worth remembering this: for instance, if you don't have enough data, you may have no clue if your data is truly representative of the world or just a fluke.

If you would like an example of how to report 95% bootstrapped confidence intervals in a manuscript, see Aboody, Zhou & Jara-Ettinger, 2021, *Child Dev.*

Additional example: Bootstrapping over a correlation

There are lots of ways to use bootstraps to generate confidence intervals. Here's just one more example. Let's say you computed a correlation, and you'd like a confidence interval over that correlation. That's pretty easy to do too.

Let's make up some fake data. It won't be particularly well correlated because it will be generated independently.

```
fake_data_2 = data.frame(Measure1 = rnorm(50,50,sd=5), Measure2 = rnorm(50,50,sd=5))
head(fake_data_2)
```

```
##   Measure1 Measure2
## 1 52.20445 44.12047
## 2 49.89804 46.10119
## 3 56.33705 40.47714
## 4 45.55366 57.80007
## 5 51.06558 39.21630
## 6 51.02725 47.37160
```

nevertheless, we can test the correlation

```
cor(fake_data_2$Measure1, fake_data_2$Measure2, method = "pearson")
```

```
## [1] -0.01156916
```

Now we can calculate a bootstrapped confidence interval over that correlation. Importantly, your data frame can only have two columns for this to work out of the box, because of the way we calculate the correlation inside the function. You can edit the code to work if you prefer to do something else though.

Essentially, we will resample pairs of data, and then perform a correlation over each resample. We then return the middle 95% range of correlations, which constitutes our 95% bootstrapped confidence interval.

`data[indices,]` extracts the first row passed in by the `boot()` function; `data[,1]` extracts the first column; `data[,2]` extracts the second.

```
Get_Correlation = function(data_frame, indices_from_boot){
  sampled_data = data_frame[indices_from_boot,]
  correlation = cor(sampled_data[,1], sampled_data[,2], method = "pearson")
  return(correlation)
}
```

Here is our wrapper to return the confidence interval:

```
Get_Correlation_Bootstrap = function(data_frame){
  bootstrap_samples = boot(data_frame,Get_Correlation,R=10000)
  return(boot.ci(bootstrap_samples, type = "basic"))
}
```

Let's run it!

```
Get_Correlation_Bootstrap(fake_data_2)
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 10000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = bootstrap_samples, type = "basic")
##
## Intervals :
## Level      Basic
## 95%   (-0.3122,  0.2862 )
## Calculations and Intervals on Original Scale
```

You can see that there's a pretty wide range of correlations we could've reasonably gotten if we repeated this experiment again, including both negative and positive correlations (which makes sense since we have no reason to think this data is actually reliably correlated). If you wanted to report a confidence interval over a correlation, you could do it this way. If you would like an example of how to report this in a manuscript, see Aboody, Davis, Dunham & Jara-Ettinger, 2021, *CogSci*.

Difference bootstraps

What's the logic?

Sometimes you want to see if two samples are different. If you calculate a bootstrap over each sample's mean and the 95% confidence intervals do *not* overlap, the two samples reliably differ (although you would still want to compute a difference bootstrap when reporting this lack of a difference). But, if they *do* overlap, it is important to understand that your samples could still be reliably different from one another.

Think about it this way: as you saw when we made our plot, we got a distribution of sample means from our 10,000 samples. And then we chopped off the bottom and top 2.5%. Austin & Hux (2002) explain

this well: "The probability that a sample mean would lie in the lower 2.5th percentile or the upper 2.5th percentile is 5%. However, when one compares two means, the probability that one mean would lie in the upper 2.5th percentile of that means sampling distribution, while the other simultaneously lies in the lower 2.5th percentile of its sampling distribution, is substantially less than 5%. Hence, despite having overlapping 95% confidence intervals, one can reject the null hypothesis with a P value that is substantially less than .05. In comparison of two groups, the confidence intervals may overlap yet the means may be significantly different from one another."

In essence: given any reasonable effect size, it's actually pretty unlikely for a mean to fall at the ends of the range. So even if two confidence intervals overlap a bit, it could be so unlikely that both means would fall at the far end of the range of sample means that we ought to conclude that the samples *are* actually different. You can't eyeball this: the only way to actually know the answer is to check.

How do we check? This is where the difference bootstrap comes in. The logic is basically the same. The only difference is that we sample data from group A, we sample data from group B, and then we return the difference in the means of each group (e.g., mean A - mean B). We do this 10,000x. And then we get a distribution over the magnitude of the difference between each group's resampled means.

How do we do it?

Let's make up some new data. Let's say that 16 out of 30 four-year-olds answered your test question correctly, and 22 out of 30 five-year-olds did. Was there a reliable difference between age groups? Let's find out!

```
fake_data_frame = data.frame("fours" = c(rep(1, 16), rep(0, 14)),
                             "fives" = c(rep(1, 22), rep(0, 8)))
head(fake_data_frame)
```

```
##   fours fives
## 1     1     1
## 2     1     1
## 3     1     1
## 4     1     1
## 5     1     1
## 6     1     1
```

Now let's write our bootstrap function. It's a little more complicated, but only superficially. The first thing we're going to do is re-shuffle the data in each one of our columns. That's because we sample our random data by indexing by row. So if the `boot()` function tells us that that we are sampling data at index 1, we are going to both sample the first data point from Group A, and the first data point in Group B. The groups will always be paired, and that's REALLY not what we want (since presumably there's no reason that the first four-year-old should be paired with the first five-year-old, and vice versa). The way to get around this is to use the `sample()` function (confusing name, I know) to shuffle the data in the columns before we generate each one of our 10,000 samples.

After we shuffle the data in each column, we re-sample data at the indices the `boot()` function will later supply. Then we calculate the average of each re-sampled group (A and B), and return the difference between the two.

Note that the way we wrote this function means it will NOT work with two vectors, and will ONLY work with a dataframe the way we set this up here. That's because the function is written to take in 1 dataframe with 2 columns, and that's what it'll take in. If you prefer to use vectors you could definitely modify it though!

```
Get_Difference = function(data_frame, indices_from_boot){
  data_frame$Column1 = sample(data_frame[,1])
  data_frame$Column2 = sample(data_frame[,2])
  ResampledData = data_frame[indices_from_boot,]
  Average1 = mean(ResampledData$Column1)
  Average2 = mean(ResampledData$Column2)
```

```
  return(Average2 - Average1)
}
```

Now, I'm going to do something slightly different with my wrapper. I'm adding 3 lines of code. The first two added lines just create variables which store your column names. When this function is run, the third line causes a reminder to print to the console, helping users remember the order your column means are being subtracted when computing the confidence interval.

Why is this helpful? Well, imagine you subtract the means of two of your columns to calculate the difference between the two, and then you decide to calculate a bootstrap over that raw difference. You want to make sure you're subtracting consistently here: so if you calculated raw scores via `Column_A - Column_B`, you'd want to calculate your difference bootstrap the same. Unfortunately, if you've forgotten how your function actually works, you might end up calculating the raw difference one way (e.g., `Column_A - Column_B`), and your confidence interval the opposite way (e.g., `Column_B - Column_A`). Oops. If you haven't touched your code for a while, this is an easy mistake to make (don't ask me how I know), so the error message is an attempt to mitigate this possibility!

```
Get_Difference_Bootstrap = function(data_frame){
  bootstrap_samples = boot(data_frame,Get_Difference,R=10000)
  Col1 = colnames(data_frame)[1]
  Col2 = colnames(data_frame)[2]
  message(glue("REMINDER: difference is calculated over {Col2} - {Col1}"))
  return(boot.ci(bootstrap_samples, type = "basic"))
}
```

Important note: if you change this code to suit your own needs, you need to make sure the warning accurately reflects what's happening in your bootstrap function. E.g., if you edit `Get_Difference` to return `Average1-Average2`, then you need to edit the order these columns are referenced in `Get_Difference_Bootstrap`.

Now let's run the difference bootstrap on our data! (Notice the reminder at the top?)

```
Get_Difference_Bootstrap(fake_data_frame)
```

```
## REMINDER: difference is calculated over fives - fours

## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 10000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = bootstrap_samples, type = "basic")
##
## Intervals :
## Level      Basic
## 95%   (-0.0333,  0.4333 )
## Calculations and Intervals on Original Scale
```

How do we interpret it?

Here, 0 is the point of "no difference" (if two means are the same and you subtract one from the other you'll get 0). So if your confidence interval is entirely above and does not cross 0, you can conclude that group A was reliably better than group B (remember, you subtracted B from A, so if the confidence interval is above 0 it must mean that A performed reliably better than B). And if your confidence interval is entirely below and does not cross 0, you can conclude that group B performed reliably better than group A. Finally, if your confidence interval crosses 0, you can conclude that there was no reliable difference between groups.

We can see that our confidence interval crosses 0, so there is no reliable difference between groups. However, the nice thing is that confidence intervals show us the magnitude of the effect: so even though there's no

reliable difference, we can still see that 5yo's did a bit better than 4yo's. The reminder should always help you interpret the direction of the effect: by reminding you that we're subtracting four-year-olds' mean performance from that of five-year-olds, we can see that a confidence interval above and not crossing zero suggests five-year-olds are better at answering our test question; conversely, if four-year-olds were better, both values should've been negative.

Additional example: bootstrapping over a correlation difference

Sometimes you want to test whether two correlations are different. For instance, maybe you have predictions from two separate models, and want to compute wheher one is more highly correlated with participant judgments, for instance. Let's create that data.

```
difference_data = data.frame("Model1" = rnorm(50,50,5), "Model2" = rnorm(50,50,5))
difference_data$Participants = difference_data$Model1 + sample(1:5, 1)
head(difference_data)
```

```
##     Model1   Model2 Participants
## 1 44.41519 48.45972     45.41519
## 2 45.12238 49.75612     46.12238
## 3 51.87029 54.65388     52.87029
## 4 47.38308 43.98390     48.38308
## 5 51.67295 47.23698     52.67295
## 6 59.57399 51.31274     60.57399
```

Now let's write our function. Again, our function takes two inputs: the data we're bootstrapping over, and the indices to use (provided by `boot()`). The first thing we do is take our samples, by using the indices provided by `boot()`. Note that we aren't shuffling the data: that's because in this case, assuming that each row corresponds to a different question or trial or measure, we really care about the correlation between these values, not across different trials or measures. So we want to preserve the relationships we have within each row, simply resampling rows with replacement to compute our statistic.

After we've indexed the rows fed in by `boot()`, we use the resulting dataframe to calculate the two correlations we need. In this example, that would be the correlation between each model and participants. Finally, we return the difference between the two correlations. The order of the two correlations matters: Here, we're subtracting correlation two from correlation one. That means that if our 95% bootstrapped correlation difference is below and does not cross 0 (e.g., both values are negative), correlation two was larger (participants and model two were better correlated). And if it is above and does not cross 0 (e.g., both values are positive), correlation one was larger (participants and model one were better correlated). As before, if the correlation difference crosses 0, this means that in some cases model one was better correlated with participant judgments, and in some cases model two was — revealing no reliable difference in correlation.

Just to recap the logic: we are resampling different rows (here, each row represents a different trial) with replacement until we have constructed a dataframe with the same number of rows our original data had. Then, we observe how well-correlated each model is with participant data. By doing this many times over and over again, we can get a measure of how much these correlations might differ if we repeated the experiment again (assuming that our results are reflective of the true world state). This gives us a confidence interval over the range of correlations we'd expect, and by subtracting the correlations from each other this can reveal whether one correlation was reliably larger than the other.

Note that we have hard-coded our column names; I thought it might be clearest this way for demo purposes; you can either modify the names to take on those of your variables, or you could modify this function to work based on position (index) instead of column name, which would be more general.

```
Get_Correlation_Difference = function(data_frame, indices_from_boot){
  sampled_data = data_frame[indices_from_boot,]
  correlation_one = cor(sampled_data$Model1, sampled_data$Participants)
  correlation_two = cor(sampled_data$Model2, sampled_data$Participants)
```

```
  return(correlation_one - correlation_two)
}
```

Finally, let's write our wrapper (with warning, although here the order of the columns in the warning is flipped, because you can see the order of the calculations in the function is flipped!).

```
Get_Correlation_Difference_Bootstrap = function(data_frame){
  Bootstrapsamples = boot(data_frame, Get_Correlation_Difference,R=10000)
  Col1 = colnames(data_frame)[1]
  Col2 = colnames(data_frame)[2]
  message(glue("REMINDER: difference is calculated over {Col1} - {Col2}"))
  return(boot.ci(Bootstrapsamples, type = "basic"))
}
```

Let's run it!

```
Get_Correlation_Difference_Bootstrap(difference_data)
```

```
## REMINDER: difference is calculated over Model1 - Model2

## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 10000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = Bootstrapsamples, type = "basic")
##
## Intervals :
## Level      Basic
## 95%   ( 0.5851,  1.1358 )
## Calculations and Intervals on Original Scale
```

Again, we interpret this essentially the same way as any other difference bootstrap: we were subtracting Model 2's correlation from Model 1's correlation. If the confidence interval crosses 0, we know the two are not reliably different. Here, you can see that Model 1 is reliably better correlated with participant data than Model 2: if the values are positive, then we know our first value is larger than the second (and in this case, the first value is the correlation between Model 1 and participant data). Of course this shouldn't be surprising, as we created participant data based on the data of Model 1!

Other resources: tidyboot!

The `tidyboot()` package is really great, and can help you streamline some of your bootstrapping needs. It's nice to know how to write your own in case there are things you want that the package can't yet do (like the examples that dealt with correlations), but it as an excellent out of the box solution that can probably handle most people's needs.

```
install.packages("tidyboot")
library(tidyboot)
```

How do you use `tidyboot`? Let's make up a fake data frame. This time, it'll be stacked: the four- and five-year-old data will be in the same column. `tidyboot` interfaces really conveniently with `dplyr`: we are just going to `group_by()` age, and then `tidyboot` will compute bootstraps separately for each age group!

Note: tidyboot computes 1,000 samples by default. 10,000 samples took a long time to compute, so I am not changing the default for our purposes here.

```
tidyboot_data = data.frame("age" = c(rep("fours", 30), rep("fives", 30)),
                           "answer"= c(rep(1, 16), rep(0, 14), rep(1, 22), rep(0, 8)))
```

```
tidyboot_data %>%
  group_by(age) %>%
  tidyboot_mean(answer, nboot = 1000)
```

```
## # A tibble: 2 x 6
##   age       n empirical_stat ci_lower  mean ci_upper
##   <chr> <int>          <dbl>    <dbl> <dbl>    <dbl>
## 1 fives    30          0.733    0.577 0.730    0.871
## 2 fours    30          0.533    0.346 0.534    0.714
```

References

Austin, P. C., & Hux, J. E. (2002). A brief note on overlapping confidence intervals. Journal of Vascular Surgery, 36(1), 194-195.

Tidyboot on CRAN: https://cran.r-project.org/web/packages/tidyboot/tidyboot.pdf

Tidyboot — a more user-friendly guide: https://github.com/langcog/tidyboot