Chapter 3 R Bootstrap Examples

Bret Larget

February 19, 2014

Abstract

This document shows examples of how to use R to construct bootstrap confidence intervals to accompany Chapter 3 of the Lock 5 textbook. It also highlights the use of the R package `ggplot2` for graphics. A quick introduction to the package `boot` is included at the end. However, when learning the bootstrap and the R language, it is useful to learn how to apply the bootstrap "from scratch" without a package to understand better how R works and to strengthen the conceptual understanding of the bootstrap.

1 Bootstrap Confidence Intervals with Standard Errors

The textbook describes how to construct a confidence interval for a population parameter by constructing an interval centered at a point estimate with a margin of error equal to twice the standard error. Here, we will estimate the size of the standard error by applying the bootstrap and sampling many samples *with replacement* from the original sample, each the same size as the original sample, computing a point estimate for each, and finding the standard deviation of this distribution of bootstrap statistics.

1.1 Atlanta Commute Times

The data set **CommuteAtlanta** from the textbook contains variables about a sample of 500 commuters in the Atlanta area.

```
library(Lock5Data)
data(CommuteAtlanta)
str(CommuteAtlanta)

## 'data.frame': 500 obs. of  5 variables:
##  $ City    : Factor w/ 1 level "Atlanta": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Age     : int  19 55 48 45 48 43 48 41 47 39 ...
##  $ Distance: int  10 45 12 4 15 33 15 4 25 1 ...
##  $ Time    : int  15 60 45 10 30 60 45 10 25 15 ...
##  $ Sex     : Factor w/ 2 levels "F","M": 2 2 2 1 1 2 2 1 2 1 ...
```

These commuters were all part of the US Census Bureau's American Housing Survey and none worked from home. You can find more details from the help page of the data set.

```
?CommuteAtlanta
```

To construct the confidence interval for the mean commute time in Atlanta, we need to find the point estimate (sample mean) from the original sample.

```
time.mean = with(CommuteAtlanta, mean(Time))
time.mean

## [1] 29.11
```

To find the standard error, we will create a huge matrix with 1000 rows (one for each bootstrap sample) and 500 columns (one for each sampled value, to match the original sample size). We will then use `apply()` to apply `mean()` to each row of the matrix. This approach differs from the example in the author R guide that uses a for loop, but we can show this approach later as well.

First create a large matrix to store all of the samples.

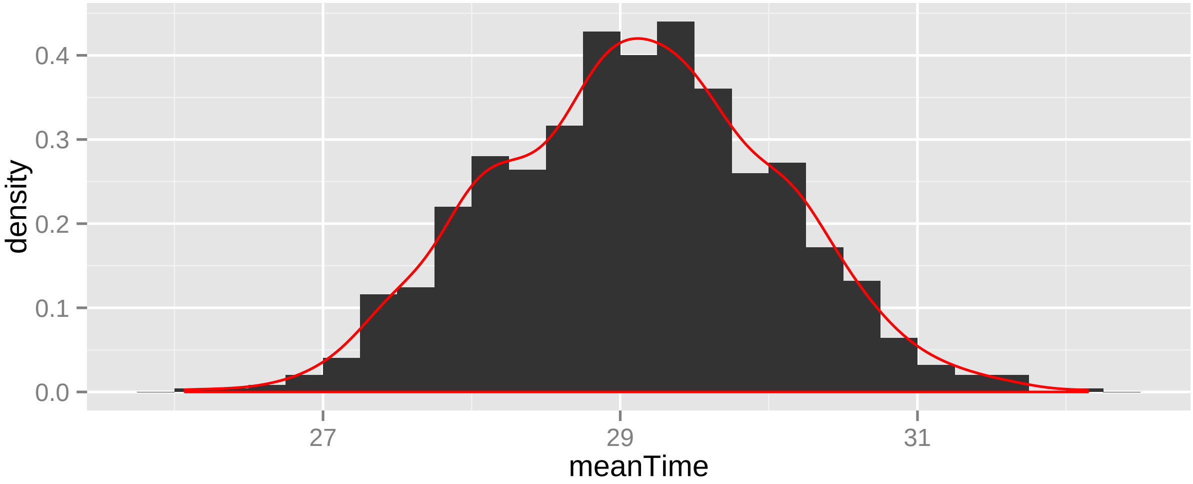
```
B = 1000
n = nrow(CommuteAtlanta)
boot.samples = matrix(sample(CommuteAtlanta$Time, size = B * n, replace = TRUE),
    B, n)
```

Notice that we used `sample()` to sample from the values in `CommuteAtlanta$Time` and we sampled with replacement (`replace=TRUE`) to fill all `B*n` elements in the matrix. Next, the three arguments to `apply()` are the object on which parts will applied the function, the number 1 to indicate the function should be applied to each row (use 2 for columns), and the function name `mean`.

```
boot.statistics = apply(boot.samples, 1, mean)
```

Graph this with a density plot overlaying a histogram for something different. Here, `ggplot()` requires a data frame with the input data, so we use `data.frame()` to create one with the only variable of interest.

```
require(ggplot2)
ggplot(data.frame(meanTime = boot.statistics),aes(x=meanTime)) +
  geom_histogram(binwidth=0.25,aes(y=..density..)) +
  geom_density(color="red")
```



We see a distribution that is not too asymmetric and is bell-shaped, more or less. The standard deviation of this distribution is as follows.

```
time.se = sd(boot.statistics)
time.se

## [1] 0.9414
```

Finally, construct the confidence interval. Here, I round the margin of error up and to one decimal place so that it has two significant digits, and I am being cautious when rounding not to make the interval too small.

```
me = ceiling(10 * 2 * time.se)/10
round(time.mean, 1) + c(-1, 1) * me

## [1] 27.2 31.0
```

Now interpret in context.

> We are 95% confident that the mean commute time in Atlanta among commuters who do not work at home is in the interval from 27.2 to 31 minutes.

The accuracy of this inference depends on the original sample being representative from the population of interest. As the US Census Bureau has likely used accurate sampling methods, this seems reasonable. We could examine the references to gain more understanding about the conditions associated with the sampling procedure (in what year, in what season, what wording was used to collect the data).

1.2 Writing a Function

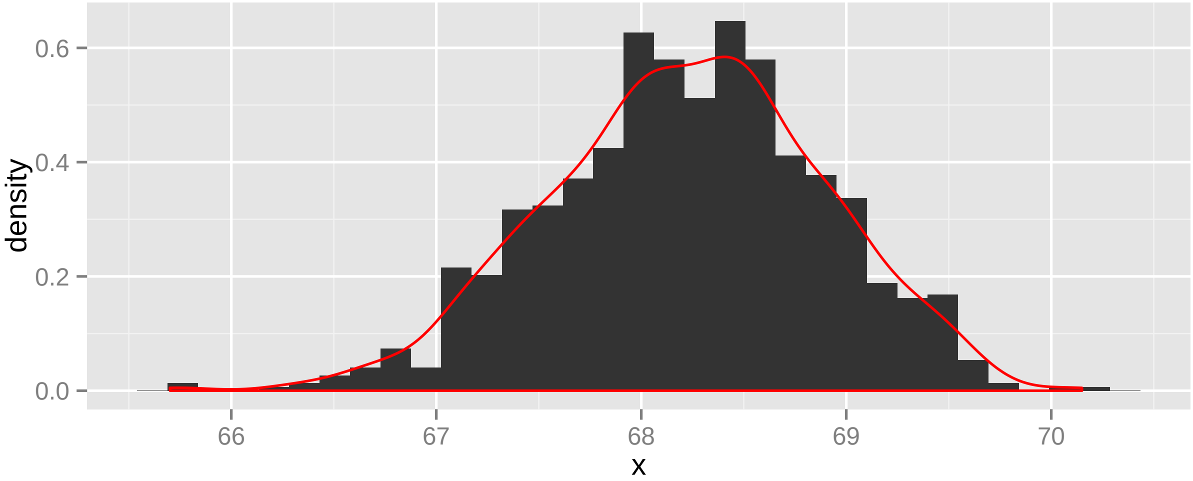
As there were several complicated steps, it would be useful to have a function to do them all so that in the future, we can source in the function and then just call it. Here is an example function that takes an argument `x` which is assumed to be a numerical sample and does the bootstrap `B` times. The function will print useful information to the console, make a plot of the bootstrap distribution, and return the bootstrap statistics, the interval, the standard error and the graph all togther as a `list`.

```
# Bret Larget
# January 10, 2014
# A quick bootstrap function for a confidence interval for the mean

# x is a single quantitative sample
# B is the desired number of bootstrap samples to take
# binwidth is passed on to geom_histogram()
boot.mean = function(x,B,binwidth=NULL) {
  n = length(x)
  boot.samples = matrix( sample(x,size=n*B,replace=TRUE), B, n)
  boot.statistics = apply(boot.samples,1,mean)
  se = sd(boot.statistics)
  require(ggplot2)
  if ( is.null(binwidth) )
    binwidth = diff(range(boot.statistics))/30
  p = ggplot(data.frame(x=boot.statistics),aes(x=x)) +
    geom_histogram(aes(y=..density..),binwidth=binwidth) + geom_density(color="red")
  plot(p)
  interval = mean(x) + c(-1,1)*2*se
  print( interval )
  return( list(boot.statistics = boot.statistics, interval=interval, se=se, plot=p) )
}
```

Here is how to use the function for the heights of students in our course.

```
students = read.csv("students.csv")
out = with(students, boot.mean(Height, B = 1000))
```



```
## [1] 66.90 69.56

out$interval

## [1] 66.90 69.56
```

We would need to be cautious, however, before using this interval to conclude that the mean height of UW students is between 66.9 and 69.6 inches as the sample of students is not random, but is a sample of convenience from our class. Here are two possible confounding variables: sex and country of origin. Is the sex of a student associated with height and with the decision to take Statistics 302? What about country of origin?

1.3 for loops

The R guide from the authors implements the bootsrap using a for loop. Rather than taking all the samples at once, the for loop just takes samples one at a time. Usually, R code that uses `apply()` is more efficient than code that uses for loops. Try both out for a large number of bootstrap replicates!

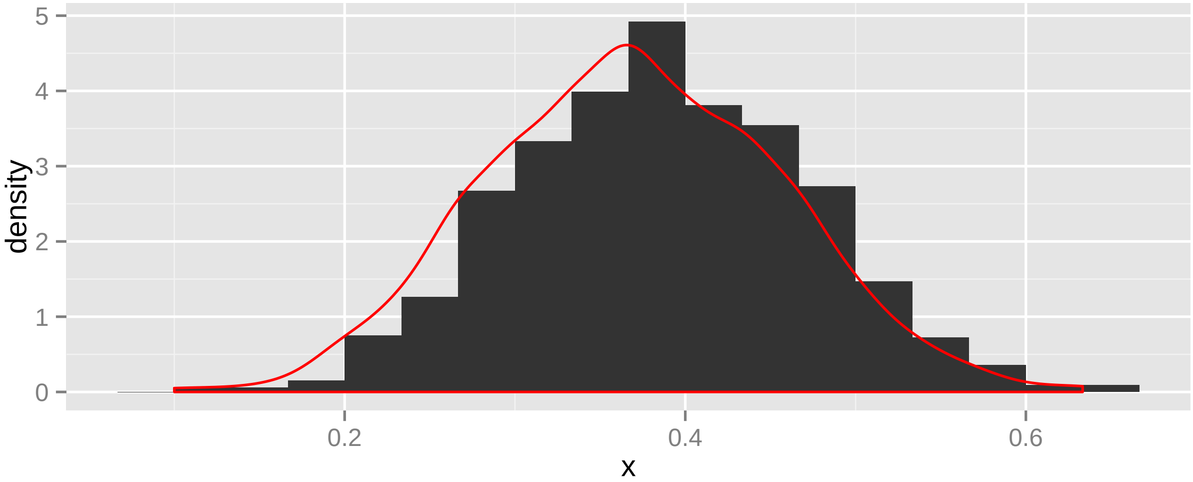
```
n = length(students$Height)
B = 1000
result = rep(NA, B)
for (i in 1:B) {
    boot.sample = sample(n, replace = TRUE)
    result[i] = mean(students$Height[boot.sample])
}
with(students, mean(Height) + c(-1, 1) * 2 * sd(result))

## [1] 66.89 69.58
```

1.4 Proportions

Consider the problem of estimating the proportion of orange Reese's pieces. I picked one of the student samples at random (really!) and chose a student with 11 orange and 19 nonorange candies. Let's use the bootstrap to find a 95% confidence interval for the proportion of orange Reese's pieces. The simplest thing to do is to represent the sample data as a vector with 11 1s and 19 0s and use the same machinery as before with the sample mean.

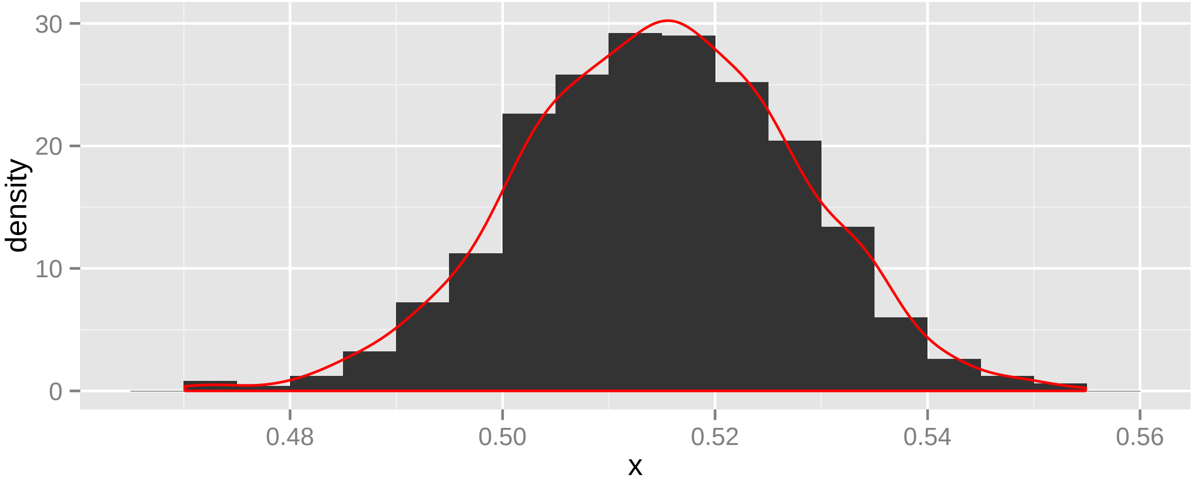
```
reeses = c(rep(1, 11), rep(0, 19))
reeses.boot = boot.mean(reeses, 1000, binwidth = 1/30)
```



```
## [1] 0.1947 0.5386
```

So, based on only this single sample, we are 95% confident that the true proportion of orange Reese's pieces is between 0.19 and 0.54. Had we combined all 48 samples together in one big sample, we could redo the problem. There were a total of 741 orange candies and 699 nonorange ones for an observed proportion of 0.515.

```
reeses = c(rep(1, 741), rep(0, 699))
reeses.boot = boot.mean(reeses, 1000, binwidth = 0.005)
```



```
## [1] 0.4888 0.5404
```

With this much larger sample size, the SE is much smaller and we have a narrower confidence interval. Anyone else think that the true proportion is 0.5?

1.5 Differences in Means

I will use the **StudentSurvey** data set from the textbook to illustrate using the bootstrap to estimate differences in means. On interesting variable is *Exercise*, the number of hours per week each student exercises.

```
data(StudentSurvey)
with(StudentSurvey, summary(Exercise))

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##    0.00    5.00    8.00    9.05   12.00   40.00       1

with(StudentSurvey, summary(Gender))

##   F   M
## 169 193

with(StudentSurvey, by(Exercise, Gender, mean, na.rm = TRUE))

## Gender: F
## [1] 8.11
## ---------------------------------------------------------
## Gender: M
## [1] 9.876
```

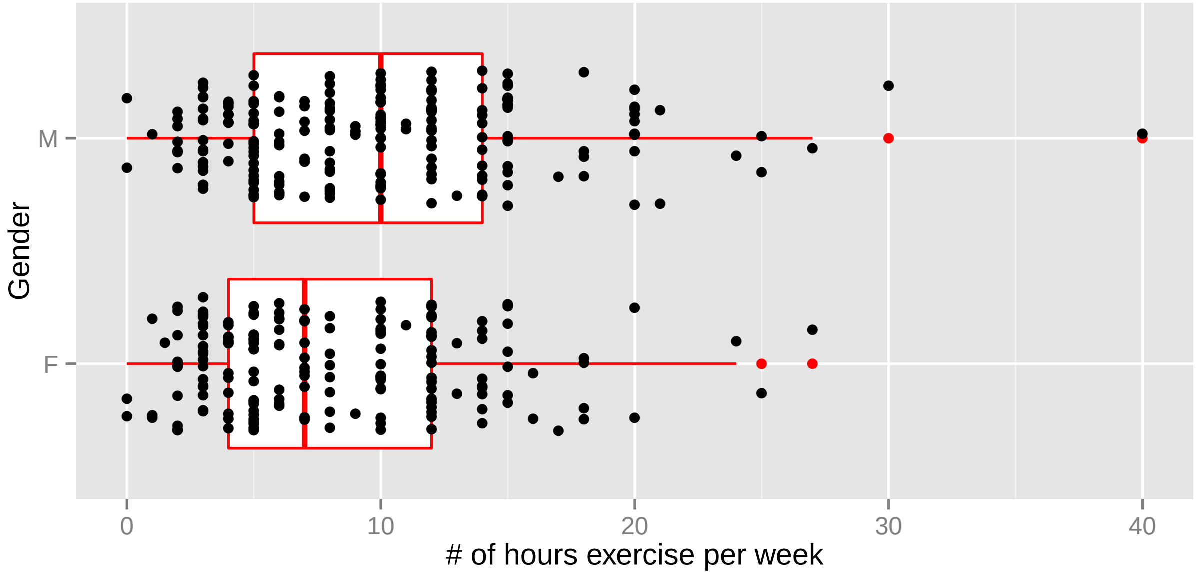
We see from this summary that in the sample, men exercise more hours per week than women. If we treat this sample of students as randomly chosen from a population of college students, we can estimate the difference in time spent exercising for each sex. Note that without more information about the sampling process, any such inference may be prone to bias is the students in the sample differ substantially from the population.

One student did not report a value for exercise. We will create a new data frame to use that eliminates this individual. Note the use of the square brackets to take a subset of the rows (and all of the columns), and the use of `!is.na()` to find all cases where the Exercise variable was *not* missing (NA).

```
newStudent = with(StudentSurvey, StudentSurvey[!is.na(Exercise), ])
```

Before constructing the confidence interval, here is a graph of the two distributions. I am using side-by-side boxplots with the actual data points plotted in a layer above, but jittered to avoid overplotting. I used `coord_flip()` to make it easier to compare the distributions and because a short and wide plot fits the page better. I outlined the boxplot in red and used red for the outlier colors so that these can be distinguished from the plotted data points.

```
ggplot(newStudent, aes(x=Gender,y=Exercise)) +
  geom_boxplot(color="red",outlier.colour="red") +
  geom_point(position = position_jitter(h=0,w=0.3)) +
  ylab('# of hours exercise per week') +
  coord_flip()
```



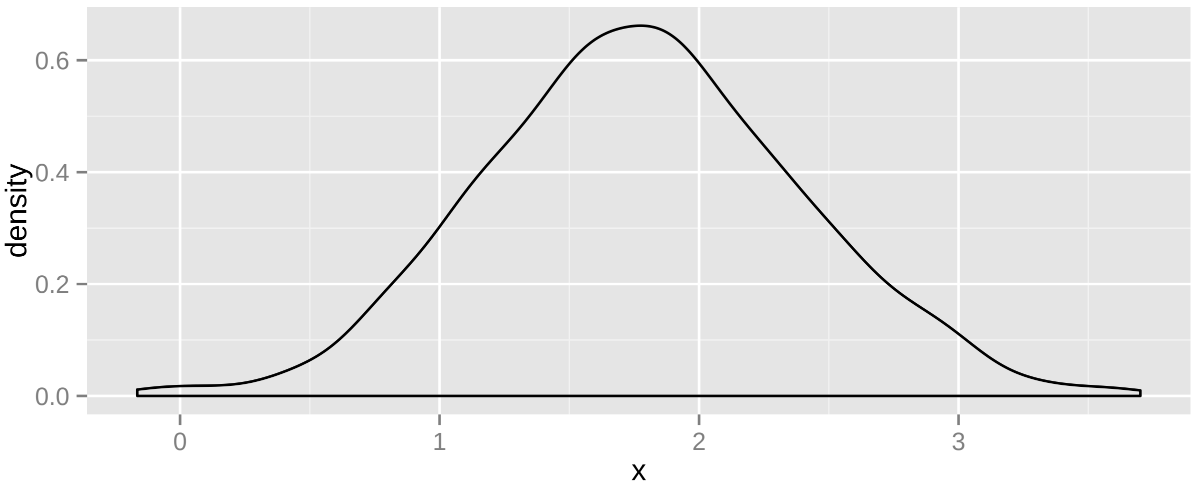
We use `length()` to find the sample size of each group. Note that the number of females is `n[1]` and the number of males is `n[2]`.

```
n = with(newStudent, by(Exercise, Gender, length))
n

## Gender: F
## [1] 168
## --------------------------------------------------------
## Gender: M
## [1] 193
```

The next block of code creates a matrix for the males and one for the females of samples with replacement of the same size for each. We then use `apply()` to fine the mean of each sample and take the differences (male minus female) to get the distribution of bootstrap statistics. We graph this to check for symmetry and a bell shape. All looks fine.

```
B = 1000
female.samples = with(newStudent, matrix(sample(Exercise[Gender == "F"], size = n[1] *
    B, replace = TRUE), B, n[1]))
male.samples = with(newStudent, matrix(sample(Exercise[Gender == "M"], size = n[2] *
    B, replace = TRUE), B, n[2]))
female.means = apply(female.samples, 1, mean)
male.means = apply(male.samples, 1, mean)
boot.stat = male.means - female.means
ggplot(data.frame(x = boot.stat), aes(x = x)) + geom_density()
```



To finish, take the point estimate (the difference in sample means) and add and subtract twice the standard error. After looking at the unrounded version, round to one decimal place for two significant figures.

```
xbars = with(newStudent, by(Exercise, Gender, mean))
me = 2 * sd(boot.stat)
(xbars[2] - xbars[1]) + c(-1, 1) * me

## [1] 0.5397 2.9914

round((xbars[2] - xbars[1]) + c(-1, 1) * me, 1)

## [1] 0.5 3.0
```

1.6 The boot package

There is a package `boot` with a function `boot()` that does the bootstrap for many situations. I will revisit the Atlanta Commute Times example. The function `boot()` requires three arguments: (1) the data from the original sample (a data frame or a matrix or an array); (2) a function to compute the statistics from the data where the first argument is the data and the second argument is the indices of the observations in the bootstrap sample; (3) the number of bootstrap replicates. An example will help you to understand.

```
library(boot)
data(CommuteAtlanta)
my.mean = function(x, indices) {
  return( mean( x[indices] ) )
}
time.boot = boot(CommuteAtlanta$Time, my.mean, 10000)
```

Notice that `my.mean(CommuteAtlanta$Time,1:length(CommuteAtlanta$Time)` computes the mean of the original sample.

The object `time.boot` is a list with many elements. One is `time.boot$t0` which is the sample mean of the original data. Another is `time.boot$t` which is the collection of bootstrap statistics that can be used as above. But the built-in function `boot.ci()` will calculate bootstrap confidence intervals using multiple methods.

```
boot.ci(time.boot)

## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 10000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = time.boot)
##
## Intervals :
## Level      Normal              Basic
## 95%   (27.28, 30.96 )   (27.24, 30.92 )
##
## Level     Percentile            BCa
## 95%   (27.30, 30.98 )   (27.40, 31.13 )
## Calculations and Intervals on Original Scale
```

Basic uses the estimated standard error. Percentile uses percentiles. BCa also uses percentiles, but adjusted to account for bias and skewness.

2 Percentile Bootstrap to come . . .

The idea of confidence intervals using percentiles from the bootstrap is to pick the endpoints from the middle of the bootstrap distribution corresponding to the desired confidence level. Here is an example using Exercise 3.109.

```
require(Lock5Data)
data(ImmuneTea)
tea = with(ImmuneTea, InterferonGamma[Drink=="Tea"])
coffee = with(ImmuneTea, InterferonGamma[Drink=="Coffee"])
tea.mean = mean(tea)
coffee.mean = mean(coffee)
tea.n = length(tea)
coffee.n = length(coffee)
B = 100000
# create empty arrays for the means of each sample
tea.boot = numeric(B)
coffee.boot = numeric(B)
# Use a for loop to take the samples
for ( i in 1:B ) {
        tea.boot[i] = mean(sample(tea,size=tea.n,replace=TRUE))
        coffee.boot[i] = mean(sample(coffee,size=coffee.n,replace=TRUE))
}
```

```
boot.stat = tea.boot - coffee.boot
# Find endpoints for 90%, 95%, and 99% bootstrap confidence intervals using percentiles.
quantile(boot.stat,c(0.05,0.95))

##     5%    95%
##  4.018 29.764

quantile(boot.stat,c(0.025,0.975))

##   2.5%  97.5%
##  1.491 32.045

quantile(boot.stat,c(0.005,0.995))

##   0.5%  99.5%
## -3.745 36.264
```