R Tutorial-10

T. Scofield

You may click here to access the .qmd file.

In this issue, we investigate applying operations involving one variable of a data frame in a manner that slices data based on the value of another variable. Specifically, we

- see what it means to construct a bootstrap distribution
- constructing confidence intervals using bootstrapping

Producing bootstrap means

Any collection of numbers can be used to calculate a mean, and through the `c()` and `mean()`, we can get R to produce the result. The mean of

$$17,\ \ 52,\ \ 44,\ \ 28,\ \ 39$$

is as straightforward as the commands

```
listOfNumbers = c(17, 52, 44, 28, 39)
mean(listOfNumbers)
```

```
[1] 36
```

If we use `resample()` without the `size` switch, the mean we obtain is of five numbers, chosen with replacement from the list. We call that a bootstrap mean. Here, I use `replicate()` to produce 3 of them.

```
replicate(3, mean( resample(listOfNumbers) ))
```

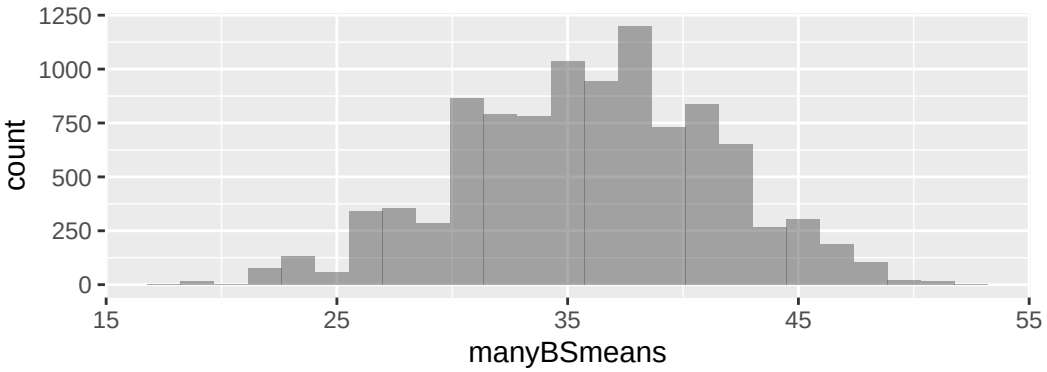
```
[1] 30.2 36.0 36.0
```

Bootstrap means rely on

- the original list of n numbers
- the particular n numbers in the **bootstrap sample** (the ones chosen by `resample()`)

One must do this much more often than 3 times to see the bootstrap distribution of means emerge:

```
manyBSmeans = replicate(10000, mean( resample(listOfNumbers) ))
gf_histogram(~manyBSmeans)
```



Producing other bootstrap statistics

Let's take an iid sample of values of size $n = 20$ of $X $ Exp($\lambda = 1/100$).

```
iidSample = rexp(20, rate=1/100)
```

Bootstrap distribution of 0.2-quantiles

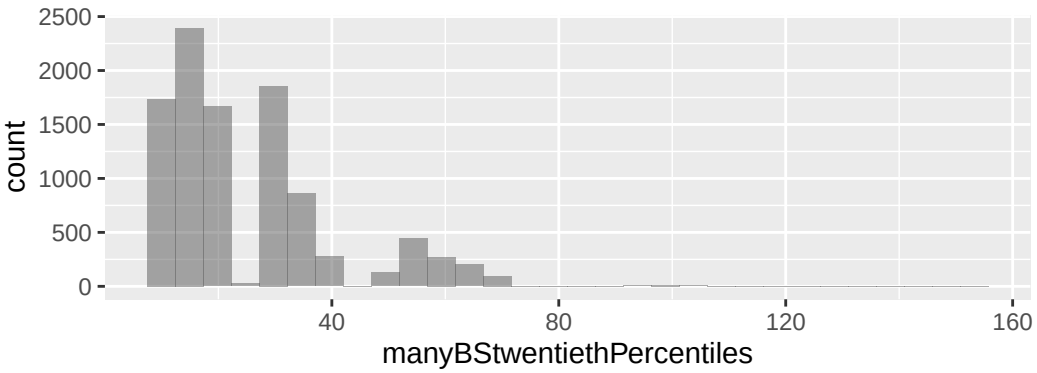
Our sample has a 0.2-quantile:

```
quantile(iidSample, probs=0.2)
```

```
    20%
17.4334
```

The bootstrap distribution of 0.2-quantile arising from this sample is generated much like a bootstrap mean:

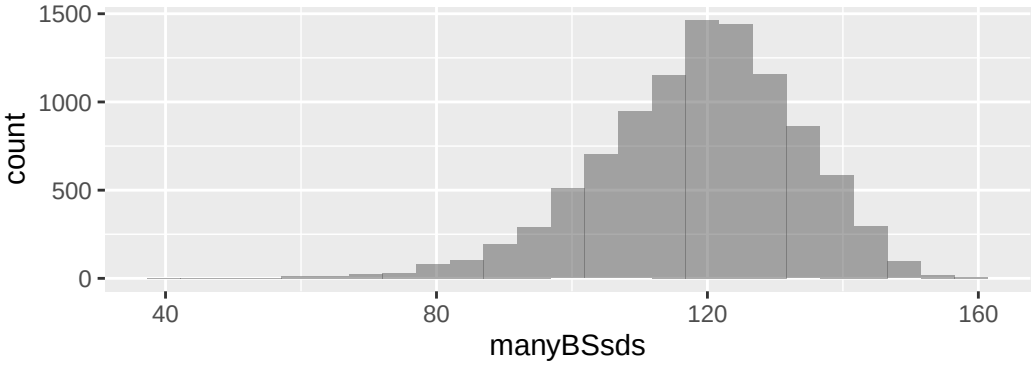
```
manyBStwentiethPercentiles = replicate(10000, quantile( resample(iidSample), probs=0.2 ))
gf_histogram(~manyBStwentiethPercentiles, bins=30)
```



Bootstrap distribution of S

Continuing to use the iid sample taken from Exp($\lambda = 1/100$) above,

```
manyBSsds = replicate(10000, sd( resample(iidSample) ))
gf_histogram(~manyBSsds)
```



Using bootstrapping to construct a confidence interval

Method 1: Use the standard deviation

Up to this point, we have constructed confidence intervals according to the method

$$(\text{point estimate}) \pm (t^*)(\text{SE})$$

Perhaps the standard deviation of the relevant bootstrap distribution can be used as an approximation to the standard error. I illustrate this using the `WeightLoss` values for the `Control` group in the `WeightLossIncentive` data frame:

```
require(Stat2Data)
```

```
Loading required package: Stat2Data
```

```
data(WeightLossIncentive)
controlGroupCases = WeightLossIncentive |> filter(Group=="Control")
mean(~WeightLoss, data=controlGroupCases)
```

```
[1] 3.921053
```

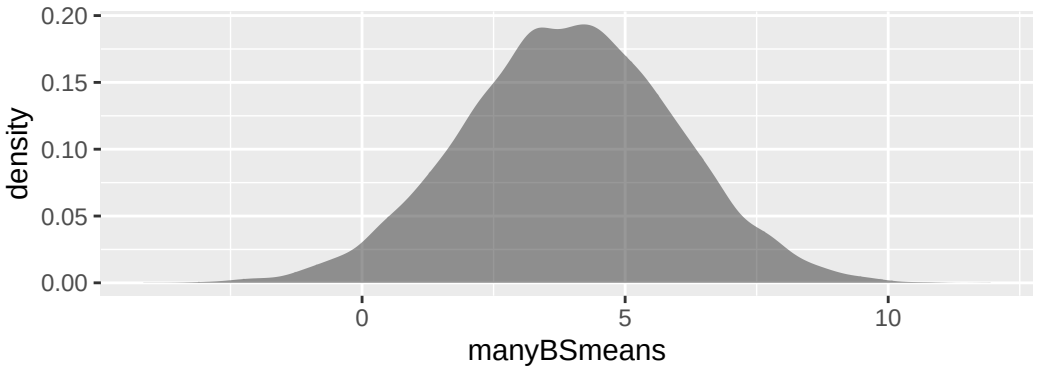
The relevant participants have been isolated in `controlGroupCases`. The mean for these is 3.921. The `WeightLoss` column gives us a this mean, but also a list of numbers we can use to produce bootstrap means. Here are three of them:

```
replicate(3, mean(~WeightLoss, data=resample(controlGroupCases)))
```

```
[1] 2.736842 6.236842 7.868421
```

The (approximate) bootstrap distribution for the sample mean:

```
manyBSmeans = replicate(10000, mean(~WeightLoss, data=resample(controlGroupCases)))
gf_density(~manyBSmeans)
```



The standard deviation of this bootstrap distribution is computed much as any sample standard deviation:

```
sd(~manyBSmeans)
```

```
[1] 2.022532
```

We might construct a 90% confidence interval for the true mean weight loss μ of `Control` participants finding t^* as before (the sample size is 19), and using the last computed value as our approximate standard error:

```
qt(0.95, df=18)
```

```
[1] 1.734064
```

$$3.921 \pm (1.734)(2.022532), \qquad \text{or} \qquad (0.414, 7.428).$$

As this is a different approach, we should not be surprised to find it differs from the one produced using 1-sample t methods:

```
t.test(~WeightLoss, data=controlGroupCases, conf.level=.9)$conf.int
```

```
[1] 0.297780 7.544325
attr(,"conf.level")
[1] 0.9
```

Method 2: Use percentiles as CI boundaries

A variant on Method 1 is to simply use percentiles of a bootstrap distribution as lower and upper bounds on the confidence interval. In the case of 90% confidence, the central 90% of values lie between the 5th and 95th percentile. An 84% confidence interval constructed this way from a bootstrap distribution would lie between the 8th and 92nd percentile.

Using this method, our 90% confidence interval for mean weight loss would be

```
quantile(~manyBSmeans, probs=c(0.05, 0.95))
```

```
       5%       95%
0.5526316 7.2368421
```

Comparing success rates for the various methods

For the `Control` cases, we have obtained 90% confidence intervals for μ using three approaches:

- 1-sample t method: $(0.298, 7.544)$
- Bootstrapping, method 1: $(0.414, 7.428)$
- Bootstrapping, method 2: $(0.553, 7.237)$

They obviously do not match. It seems possible that, had early statisticians been able to harness the power of modern computers, 1-sample t methods may never have been discovered, and one should resist the temptation to call it the *right* way to do things.

Testing the three approaches

The measure of a method is whether it performs, under the basic assumptions, at its advertised rate. A 95% confidence interval should capture the parameter within its bounds 95% of the time, in the long run. We should look at the three approaches and compare how they do.

To test the 1-sample t method, we will repeatedly draw an iid from a normal population (using `rnorm()`), use it to construct a 95% confidence interval, and see if the mean lies inside. Doing so once will not tell us much, but doing it 100 times starts to give a sense of the empirical success rate.

```
mu = 40
n = 10
result = replicate(100, {
 iid = rnorm(n, mu, 1)
 xbar = mean(iid)
 s = sd(iid)
 tstar = qt(0.975, df=n-1)
 CI_1st = xbar + c(-1,1) * tstar * s / sqrt(n)
 (CI_1st[1] < mu) & (mu < CI_1st[2])   # boolean result
}
)
sum(result) / length(result)
```

```
[1] 0.94
```

The one hundred 1-sample t confidence intervals, built choosing t^* so that they are supposed to have a 95% success rate, succeeded 94 out of 100 times. If you run this code in the console, you will find it sometimes succeeds at or above 95 times out of a hundred, but is always in the ballpark of 95.

Tasks for you

1. Write code that draws an iid sample from Norm$(40, 1)$, generates a bootstrap distribution, employs Method 1 (described above) to generate a 95% confidence interval, and then checks whether $\mu = 40$ lies inside. Then use `replicate()` to repeat this 100 times and report the success rate for capturing μ. Does Method 1 appear to perform at the level it is supposed to get?
2. Write code that draws an iid sample from Norm$(40, 1)$, generates a bootstrap distribution, employs Method 2 (described above) to generate a 95% confidence interval, and then checks whether $\mu = 40$ lies inside. Then use `replicate()` to repeat this 100 times and report the success rate for capturing μ. Does Method 2 appear to perform at the level it is supposed to get?

If our samples are pulled from a non-normal population, the size of n may play a role in whether the empirical success rate is close to the advertised rate.

3. Return to the 1-sample t approach, drawing iid samples from Unif$(8, 12)$, constructing 95% confidence intervals 100 times, and observing the success rate. Do this for different sample sizes n, starting at $n = 2$, and increasing from there. Identify the smallest n at which it seems to you the empirical success rate is reliably 95%.
4. Do the same as in Number 3, but drawing the iid samples from Exp$(\lambda = 1/100)$.
5. Do as in Number 4 (iid samples come from Exp$(\lambda = 1/100)$), but construct your 95% confidence intervals using Method 1 based on bootstrap distributions.

Permutation Testing for $\mu_2 - \mu_1$, independent samples

We explore another technique that is computationally intensive, **permutation testing**, which might be used instead of a 2-sample t hypothesis test. To compare, we use data familiar from earlier class periods when we were learning 2-sample t techniques. Specifically, we carried out a hypothesis test between

$$\mathbf{H}_0 : \mu_2 - \mu_1 = 0 \qquad \text{and} \qquad \mathbf{H}_a : \mu_2 - \mu_1 > 0,$$

where μ_1 is the average amount of weight lost by individuals in the control group, μ_2 is the average of weight lost by those incentivized, in a study like that giving the data in `WeightLossIncentive`. The sample means for the two groups are

```
filteredDat = filter(WeightLossIncentive, !is.na(WeightLoss))    # remove NA values
mean(WeightLoss ~ Group, data=filteredDat)
```

```
  Control Incentive
 3.921053 15.676471
```

When this result is handed to the `diff()` command, it subtracts the two values $15.676 - 3.921$:

```
mean(WeightLoss ~ Group, data=filteredDat) |> diff()
```

```
Incentive
 11.75542
```

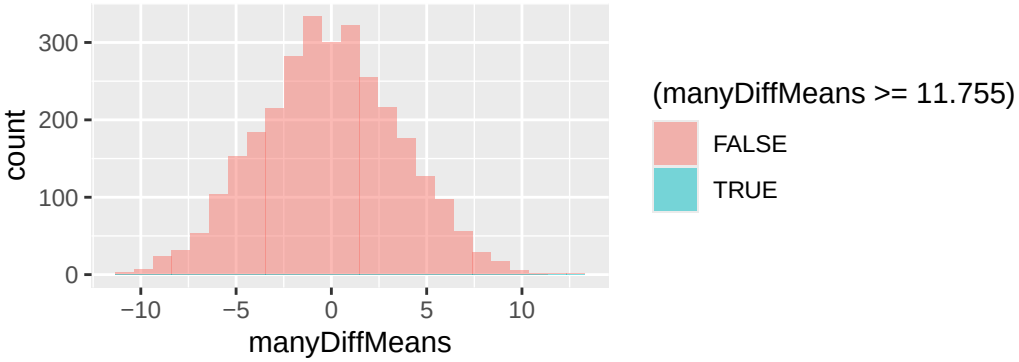
The null hypothesis of our test makes the assertion that there is no association between the variables `Group` and `WeightLoss`; the result of the response (`WeightLoss`) is independent of which `Group` label a participant has. Permutation testing simulates this independence by computing many differences such as the one above, but shuffling the attachments between explanatory and response values. To do this three times produces three different differences of (sample) means:

```
replicate(3,
  mean(WeightLoss ~ shuffle(Group), data=filteredDat) |> diff()
)
```

```
 Incentive  Incentive  Incentive
-2.6222910  0.8885449 -3.0123839
```

Repeating this *many* times gives us a simulated sampling distribution of values `\overline X_2 - \overline X_1` under the null hypothesis that the two variables have no association.

```
manyDiffMeans = replicate(3000,
  mean(WeightLoss ~ shuffle(Group), data=filteredDat) |> diff()
)
gf_histogram(~manyDiffMeans, fill=~(manyDiffMeans >= 11.755))
```




```
prop(~(manyDiffMeans >= 11.755))
```

```
   prop_TRUE
0.0006666667
```

The distribution of differences shows what is possible when the null hypothesis is true. We attempt to shade those differences beginning at the value 11.755 seen in our original sample, along with every difference at least that extreme (but further out in the right tail only). There are so few results this extreme, the second color is invisible to our eyes.

In our simulation, 2 out of 3000 resamples ($2/3000 = 0.0006667$) produced differences at least as extreme as our result. Taking this as an (approximate) P-value, we reject the null hypothesis. Compare to the 2-sample t result produced using `t.test()`:

```
t.test(WeightLoss ~ Group, alternative="less", data=filteredDat)
```

```
    Welch Two Sample t-test

data:  WeightLoss by Group
t = -3.7982, df = 33.276, p-value = 0.0002944
alternative hypothesis: true difference in means between group Control and group Incentive is l
95 percent confidence interval:
     -Inf -6.51882
sample estimates:
  mean in group Control mean in group Incentive
               3.921053                15.676471
```