Lab 8

Steven Boyd

11/18/2021

Quantile functions

You've seen quantile functions in the lecture notes, but you will be expected to use them in the problem set. Thus far, you've primarily encountered the `qnorm` function (i.e. "quantile of the normal"). Recall how it works:

```
qnorm_995 <- qnorm(.995)

qnorm_005 <- qnorm(.005)
```

What do the values `qnorm995` and `qnorm005` represent? What else do you know about the distribution used to generate these values (i.e. what is the mean and standard deviation)? Check the documentation to find the null arguments if you are unsure.

[YOUR ANSWER HERE]

So, this works if the normal distribution is a suitable approximation of the null distribution for our hypothesis test, but this won't always be the case. Luckily, we can use the `quantile` function with any distribution! See this example:

```
unif_sample <- runif(100, min = 0, max = 100)

quantile(unif_sample, probs = .9)
```

```
##      90%
## 84.85308
```

```
quantile(unif_sample, probs = c(.05, .95))
```

```
##        5%       95%
##  5.815369 95.168740
```

What do these values represent? Are they symmetric about the mean like the values produced by `qnorm` above?

[YOUR ANSWER HERE]

Now, use `qunif` to find the .05 and .95 quantiles of the same uniform distribution used to generate the sample above.

How do the values compare to the quantiles of the sample? When you increase the sample size, what happens to the sample quantiles relative to the distribution quantiles?

[YOUR ANSWER HERE]

More on Bootstrapping

Last week we practiced bootstrapping from a specific sample, which was a vector. But, we can bootstrap from dataframes too! To do so, we'll use the `slice_sample()` function. Rather than randomly sample values from a vector, it randomly samples rows. This is helpful if you want to do bootstrapping and there are multiple variables that you want to keep from your data. Why might you want to do so?

[YOUR ANSWER HERE]

Run the following chunk. What is the structure of the output?

```
mt_cars_boot_1 <- map(1:1000, ~slice_sample(mtcars))

head(mt_cars_boot_1)
```

```
## [[1]]
##             mpg cyl  disp  hp drat   wt qsec vs am gear carb
## Merc 450SE 16.4   8 275.8 180 3.07 4.07 17.4  0  0    3    3
##
## [[2]]
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Pontiac Firebird 19.2   8  400 175 3.08 3.845 17.05  0  0    3    2
##
## [[3]]
##                       mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Lincoln Continental 10.4   8  460 215    3 5.424 17.82  0  0    3    4
##
## [[4]]
##                     mpg cyl disp  hp drat   wt  qsec vs am gear carb
## Cadillac Fleetwood 10.4   8  472 205 2.93 5.25 17.98  0  0    3    4
##
## [[5]]
##                     mpg cyl disp  hp drat   wt  qsec vs am gear carb
## Cadillac Fleetwood 10.4   8  472 205 2.93 5.25 17.98  0  0    3    4
##
## [[6]]
##                    mpg cyl disp  hp drat   wt  qsec vs am gear carb
## Dodge Challenger 15.5   8  318 150 2.76 3.52 16.87  0  0    3    2
```

Alter the code to take 1000 bootstrapped versions of `mtcars`, where each bootstrapped dataframe has the same number of rows as the original. Save the output as an object called `mt_cars_boot_2`.

What does the output look like now?

Iterating linear models

One thing we can do with these bootstrapped dataframes is feed them into map and fit a linear model to each. Run the following code in a chunk:

boot_lm <- map(mt_cars_boot_2, ~lm_robust(mpg ~ cyl + disp, data = .) %>% coef())

head(boot_lm)

What does the output look like? Alter the code to iterate a linear model of your own design over the bootstrapped dataframes. Output the regression coefficients on each variable as a *matrix* (hint: use `bind_rows`).