Lab 8 Solutions

11/18/2022

```
## Warning: package 'tidyverse' was built under R version 4.2.1
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.3.6     v purrr   0.3.4
## v tibble  3.1.7     v dplyr   1.0.9
## v tidyr   1.2.0     v stringr 1.4.0
## v readr   2.1.2     v forcats 0.5.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
## Warning: package 'estimatr' was built under R version 4.2.1
```

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.

A: + The values represent the value below which 99.5% and .5% of the normal distribution lie respectively. + In other words, the two values mark the upper and lower bounds of a 99% confidence interval. + Looking at the documentation, the null arguments for qnorm are `mean = 0` and `sd = 1`, which means the distribution used to generate these quantiles is the *standard* normal (just like the other `norm` functions). + It makes sense that `qnorm95 = -qnorm5` because the standard normal is symmetric about 0.

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 numeric vector! See this example:

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

# The generic function quantile produces sample quantiles corresponding to the given probabilities.
quantile(unif_sample, probs = .9)
```

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

```r
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?

A + The values represent the point below which 90% of the sample lies and the bounds of the middle 90% of the sample respectively. + They are not symmetric because the quantiles were calculated on a random *sample* from the uniform distribution, not the uniform distribution itself (we can think of it as an empirical CDF). + The sampling process introduces randomness.

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

```r
# qunif gives the quantile function
qunif(p = c(.05, .95), min = 0, max = 100)
```

```
## [1]  5 95
```

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?

```r
unif_sample_2 <- runif(n = 1000, min = 0, max = 100)

unif_sample_3 <- runif(n = 5000, min = 0, max = 100)

quantile(unif_sample_2, probs = c(.05, .95))
```

```
##        5%       95%
##  4.269144 94.712080
```

```r
quantile(unif_sample_3, probs = c(.05, .95))
```

```
##        5%       95%
##  4.555576 95.032843
```

A The upper and lower sample quantiles are close, but not exactly the same as the distribution quantiles, but as we increase the sample size, they appear to converge to the true values (5 and 95 respectively).

More on Bootstrapping

Last week we practiced bootstrapping from a sample, which was a numeric 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 there are *multiple variables that you want to keep from your data when you bootstrap*. Why might you want to do so?

A As we will do below, this is helpful because you can use these bootstrapped samples and `map` to run the same linear model specification repeatedly to estimate certain parameters (like robust standard errors).

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
## Porsche 914-2  26   4 120.3 91 4.43 2.14 16.7  0  1    5    2
##
## [[2]]
##               mpg cyl disp  hp drat   wt qsec vs am gear carb
## Maserati Bora  15   8  301 335 3.54 3.57 14.6  0  1    5    8
##
## [[3]]
##               mpg cyl  disp hp drat   wt qsec vs am gear carb
## Porsche 914-2  26   4 120.3 91 4.43 2.14 16.7  0  1    5    2
##
## [[4]]
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Chrysler Imperial 14.7   8  440 230 3.23 5.345 17.42  0  0    3    4
##
## [[5]]
##              mpg cyl disp hp drat    wt  qsec vs am gear carb
## Honda Civic 30.4   4 75.7 52 4.93 1.615 18.52  1  1    4    2
##
## [[6]]
##                    mpg cyl disp  hp drat   wt  qsec vs am gear carb
## Hornet Sportabout 18.7   8  360 175 3.15 3.44 17.02  0  0    3    2
```

A This code produces a list of 1000 dataframes. Each dataframe contains only one randomly sampled row from `mtcars`. This is because the default value of `n` in `slice_sample` is 1. So, every iteration generates a single random row from the original dataframe.

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`.

```
# Creating 1000 simulated samples with 32 rows
mt_cars_boot_2 <- map(1:1000, ~ slice_sample(.data = mtcars, replace = TRUE, n = nrow(mtcars)))

# How it looks like:
View(mt_cars_boot_2)

# Checking two samples
head(mt_cars_boot_2, 2)
```

```
## [[1]]
##                      mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Merc 450SL...2      17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Hornet 4 Drive...4  21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Toyota Corolla...6  33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Datsun 710...7      22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
```

```
## Porsche 914-2...8   26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## AMC Javelin...10    15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Datsun 710...12     22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Merc 450SL...13     17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Ford Pantera L...15 15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Hornet 4 Drive...16 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Ford Pantera L...18 15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Porsche 914-2...19  26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Valiant...24        18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Toyota Corolla...25 33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SL...27     17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## AMC Javelin...28    15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
## Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Datsun 710...31     22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Valiant...32        18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
##
## [[2]]
##                        mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Ford Pantera L...1    15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Camaro Z28            13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Cadillac Fleetwood    10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Toyota Corona         21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Ford Pantera L...5    15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## AMC Javelin...6       15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Lotus Europa          30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Mazda RX4             21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Maserati Bora...9     15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Dodge Challenger      15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## Merc 240D...11        24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 450SL...12       17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Maserati Bora...13    15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Datsun 710            22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Toyota Corolla        33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Duster 360            14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D...17        24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Volvo 142E...18       21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
## Pontiac Firebird...19 19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Porsche 914-2...20    26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Porsche 914-2...21    26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Maserati Bora...22    15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Porsche 914-2...23    26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Pontiac Firebird...24 19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Merc 450SL...25       17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Volvo 142E...26       21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

```
## Fiat 128              32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Ford Pantera L...28   15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Honda Civic           30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Hornet 4 Drive        21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## AMC Javelin...31      15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Lincoln Continental   10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
```

What does the output look like now?

A The output is still a list of 1000 dataframes, but this time each dataframe has 32 rows (the same number of rows as the original `mtcars` data). Also notice that some rows are repeated. If we do not specify `replace = TRUE` then each sample will contain exactly the same observations as the original data, but in a random order. For our purposes, that's not helpful because the summary statistics like mean and variance of each column would be exactly the same as in the original data. Note that rows that are chosen more than once take on the format `[carname]..[rownumber]`.

Iterating linear models

One thing we can do with these bootstrapped dataframes is feed them into map and run a linear regression on each sample. Run the following chunk:

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

head(boot_lm)
```

```
## [[1]]
## (Intercept)         cyl        disp
## 32.50284812 -1.01148324 -0.02634963
##
## [[2]]
## (Intercept)         cyl        disp
## 33.56615131 -1.02294739 -0.02925947
##
## [[3]]
## (Intercept)         cyl        disp
## 38.36862343 -2.33637369 -0.01549278
##
## [[4]]
##  (Intercept)          cyl         disp
## 32.888230842 -1.917850112 -0.004845247
##
## [[5]]
## (Intercept)         cyl        disp
## 37.34623923 -2.00926797 -0.02164827
##
## [[6]]
##  (Intercept)          cyl         disp
## 33.206138701 -2.420643104  0.009271054
```

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

```r
boot_lm_2 <- map(mt_cars_boot_2, ~lm_robust(mpg ~ hp*am, data = .) %>%
                   coef()) %>%
  bind_rows

head(boot_lm_2, 10)
```

```
## # A tibble: 10 x 4
##    `(Intercept)`      hp    am   `hp:am`
##            <dbl>   <dbl> <dbl>     <dbl>
##  1          26.2 -0.0553 5.38  -0.00307
##  2          27.6 -0.0648 3.79   0.0109
##  3          25.4 -0.0505 6.65  -0.0136
##  4          25.8 -0.0503 3.64   0.00318
##  5          26.0 -0.0574 5.82   0.000430
##  6          27.2 -0.0560 0.947  0.0124
##  7          27.6 -0.0630 6.78  -0.0105
##  8          26.0 -0.0582 6.39   0.00267
##  9          25.0 -0.0499 2.74   0.00821
## 10          25.8 -0.0546 6.87  -0.00171
```