Stat 230 - Example for 4-7

A.S. Wagaman

MLR: Bootstrap for Regression (4.7)

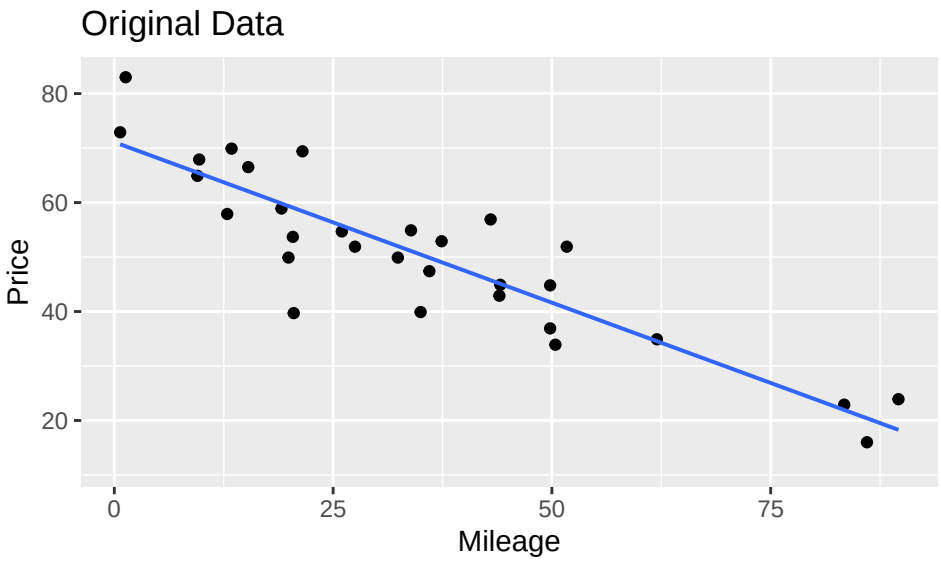
We have discussed randomization/permutation tests. A similar idea is the bootstrap. Here, we will repeatedly sample cases from our data set, *with replacement*, and construct the distribution of a statistic. The key idea here is that even though the solution to the regression model is deterministic, the data itself is assumed to be randomly sampled, and so all of the estimates that we make in a regression model are **random**. If the data changes, then the estimates change as well. The bootstrap gives us a non-parametric understanding of the distribution of those estimates. Once again, the advantage to this method is that we can construct meaningful confidence intervals for, say, the slope of the regression line, without having to assume that the residuals are normally distributed.

In this example we are examining the prices of Porsches. We want to estimate the *Price* as a function of the *Mileage*. Our sample of 30 cars looks like this:

```
PP <- data(PorschePrice)
PP <- PorschePrice #renamed to be shorter
head(PP)
```

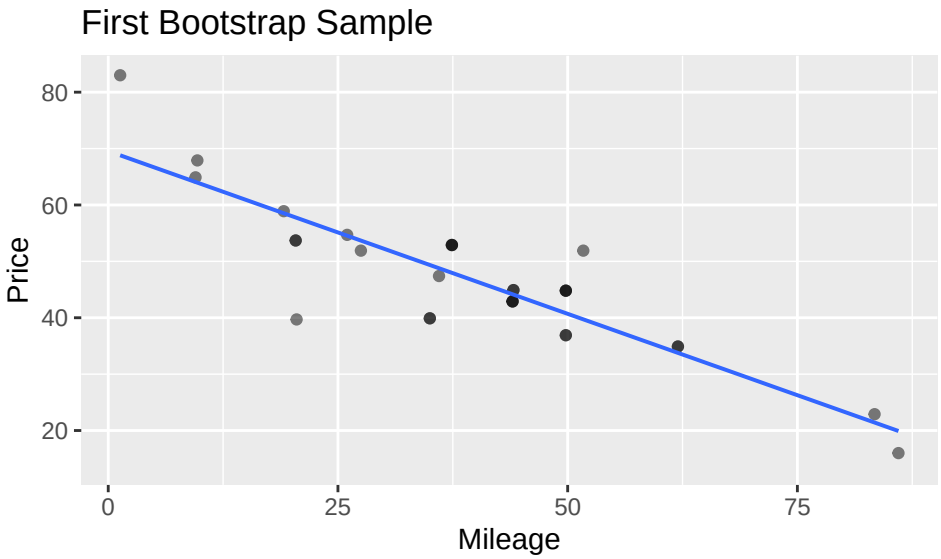
```
##   Price Age Mileage
## 1  69.4   3    21.5
## 2  56.9   3    43.0
## 3  49.9   2    19.9
## 4  47.4   4    36.0
## 5  42.9   4    44.0
## 6  36.9   6    49.8
```

```
gf_point(Price ~ Mileage, data = PP, title = "Original Data") %>%
  gf_lm()
```



To create a bootstrap sample, we select rows from the data frame uniformly at random, but with replacement. In other words, you create a data set the same size as the original, but some observations repeat (and can repeat several times), and others are left out.

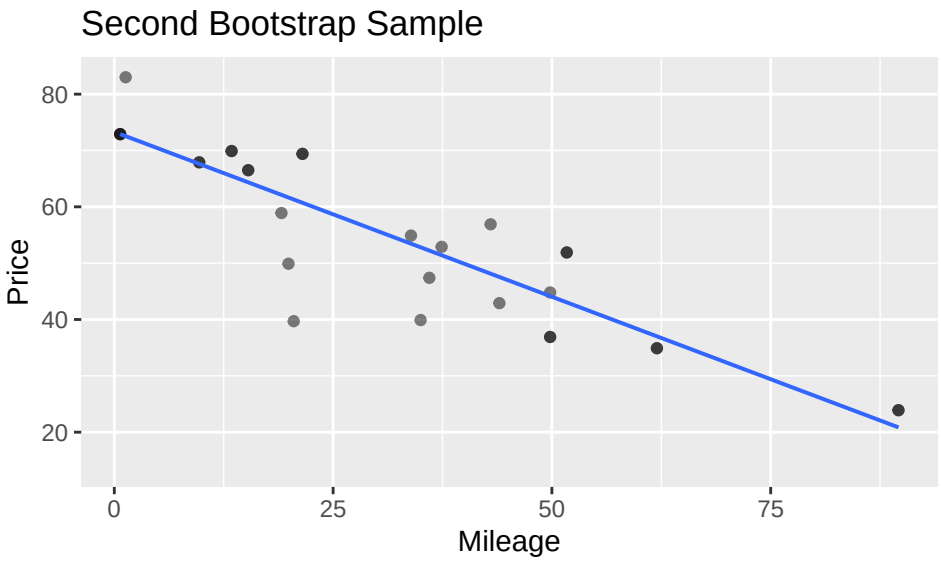
```
set.seed(201)
gf_point(Price ~ Mileage, data = resample(PP), title = "First Bootstrap Sample", alpha = 0.5) %>%
  gf_lm()
```



Note the differences between this plot and the previous one. What do you notice?

Let's try it again.

```
set.seed(231)
gf_point(Price ~ Mileage, data = resample(PP), title = "Second Bootstrap Sample", alpha = 0.5) %>%
  gf_lm()
```



The **resample** function is what allows you to create bootstrap samples. You could work with each sample, but usually, we are more interested in studying the distribution of some value saved from each bootstrapped sample, such as the distribution of bootstrapped slope coefficients for some predictor. Thus, we repeat the process, generating a collection of values to examine. This is different from the randomization procedure though because here, we are not destroying existing relationships, though we will get data sets with some of the original observations repeated and some left out entirely.

Confidence Intervals Using the Bootstrap The primary advantage of the bootstrap is that it allows us to construct confidence intervals for the slope coefficient that are not nearly as dependent upon the conditions for linear regression being met.

The original confidence intervals for our SLR model depend upon the regression conditions being met.

```
fm <- lm(Price ~ Mileage, data = PP)
confint(fm)
```

```
##                2.5 %    97.5 %
## (Intercept) 66.23602 75.944887
## Mileage     -0.70544 -0.473362
```

Now let's create a bootstrap distribution for the regression coefficients. Note the resample command is used to resample the data from itself.

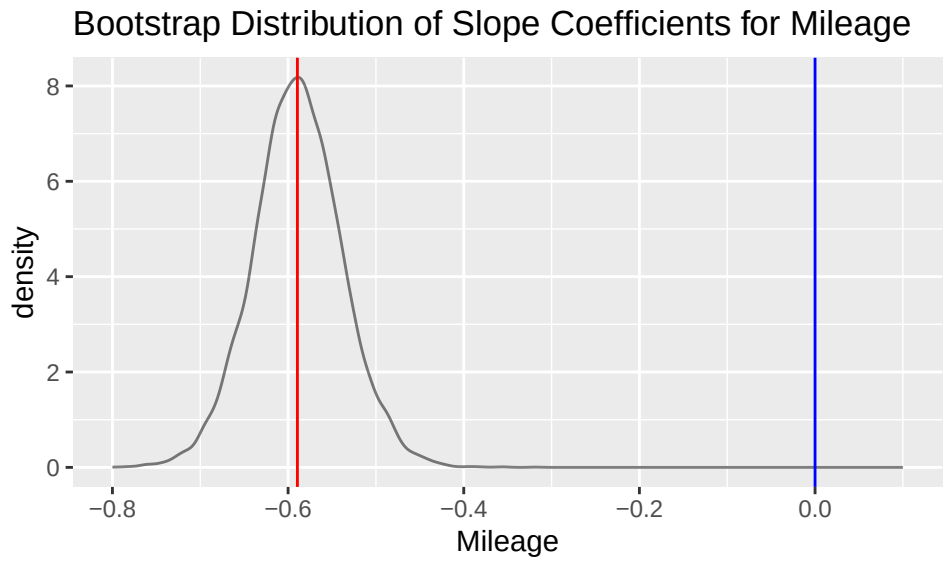
```
set.seed(500)
bootstrap <- do(10000) * lm(Price ~ Mileage, data = resample(PP))$coefficients # might take a second to

names(bootstrap) #saved slope coefficients
```

```
## [1] "Intercept" "Mileage"
```

```
gf_dens(~ Mileage, data = bootstrap) %>%
  gf_labs(title = "Bootstrap Distribution of Slope Coefficients for Mileage") %>%
  gf_lims(x = c(-0.8, 0.1)) %>%
  gf_vline(xintercept = ~ coef(fm)["Mileage"], color = "red") %>%
  gf_vline(xintercept = ~ 0, color = "blue")
```

```
## Warning: Removed 3 rows containing non-finite values (stat_density).
```



This gives us a plot of the coefficient for Mileage based on 10000 bootstrapped coefficients. In contrast to permutation test results, these are slopes we WOULD expect to get though some values are more common than others. We can use this distribution to construct CIs. In fact, there are three methods for constructing confidence intervals from the bootstrapped distribution of the statistic (the final one for skewed distributions takes a little bit more work).

Method #1 The first method does not require that the bootstrap distribution be normal, but only works well when it is roughly symmetric. In this case we simply use the percentiles of the bootstrap distribution to build confidence intervals. We choose the cutoffs to attain a 95% interval for the slope coefficient for Mileage, putting 2.5 percent in each tail (so 95 percent in between).

```
qdata(~ Mileage, c(0.025, 0.975), data = bootstrap)
```

```
##      2.5%     97.5%
## -0.689787 -0.486799
```

Think about how you would adjust this to create a 90% interval instead of a 95% interval.

Method #2

The second method assumes that the bootstrap distribution is **normal**. In this case we can use the standard deviation of the bootstrap distribution to construct a confidence interval for the slope coefficient. 1.96 is the critical value for a 95% interval assuming normality.

```
mycoef <- coef(fm)["Mileage"]
mycoef + c(-1.96, 1.96) * sd(bootstrap$Mileage) #95% CI
```

```
## [1] -0.689287 -0.489515
```

You can adjust to other confidence levels by changing the critical value.

```
qnorm(.95) #90% critical value. 5% in upper tail + 5% in lower tail would yield 90% CI
```

```
## [1] 1.64485
```

So, you'd replace (-1.96, 1.96) with (-1.645, 1.645) if you wanted a 90% CI instead of a 95% CI.

In previous editions of the textbook, Method 2 was listed as Method 1, and there was a progression of CIs - from distributions that looked normal, then to those that were symmetric, then to skewed ones. With modern

computation though, there's less need to focus on normality, and the (current) Method 1 can be used even when the distribution is normal, so there's less need for this (current) Method 2.

Method #3 If the bootstrap distribution is skewed, then we need to do a bit more work. Basically, we use similar ideas to Method 2, but we generate the critical values ourselves, and they may not be equidistant from 0 (depending on the skew). This approach takes a lot more work because we need to construct not just the bootstrap distribution for the slope coefficients but also that of the t-statistics comparing the bootstrap slopes to the original slope.

First we define a function to compute the T* value for any bootstrap sample. When you run the chunk below, this will create a function in your workspace to use. It would need inserted into any future .Rmd where you need to do this as well. (This function was written by your textbook authors, there are more elegant ways to do this as well.)

```
tstar <- function(bootmodel,origmodel, pos=2){
  SE <- summary(bootmodel)$coeff[2*pos]
  slope <- summary(bootmodel)$coeff[pos]
  origslope <- summary(origmodel)$coeff[pos]
  ts <- (slope-origslope)/SE
  return(c(slope,ts))
}
```

Now, we redo our bootstrap procedure, but this time, it will save the bootstrapped slopes and t-statistics we want to use to generate the critical values. The pos = 2 argument is important to pay attention to. For Mileage, since this is an SLR, it's coefficient will be in position 2. If you are doing a different model, such as an MLR, you may need to change the pos argument.

```
set.seed(500)
manyTs <- do(10000)*tstar(lm(Price ~ Mileage, resample(PP)), fm)
names(manyTs)
```

```
## [1] "V1" "V2"
```

The result here doesn't have nice names for the variables. The first variable is the bootstrapped slopes, and second is the desired t-star values.

We can get the bootstrap slopes and t*-values and store them in variables.

```
bootslopes <- manyTs[ ,1]
Ts <- manyTs[ ,2]
```

Now to find the quantiles we want, we find quantiles of the T* distribution to match our chosen confidence level.

```
qtl <- quantile(Ts, 0.025)
qtu <- quantile(Ts, 0.975)
c(qtl,qtu) #to see what they are
```

```
##     2.5%    97.5%
## -1.84909  1.71118
```

Here, these do not particularly demonstrate that our distribution was skewed - they are nearly symmetric around 0.

Now we can find the CI using our found T* quantiles as multipliers. Remember due to the skewness, the *upper* multiplier is from the lower tail of the t-star distribution, and vice versa.

```
SE <- sd(bootslopes) #should be the same as sd(bootstrap$Mileage) because used the same seed
SE
```

```
## [1] 0.0509624
```

```
sd(bootstrap$Mileage)
```

```
## [1] 0.0509624
```

```
c(mycoef - qtu*SE, mycoef - qtl*SE)
```

```
##   Mileage   Mileage
## -0.676607 -0.495167
```

Here again, the distribution was not really skewed, so this CI closely matches that from Method 1 and 2.

Return to GPA Example In this example, suppose we are trying to explain a college student's *GPA* as a function of their score on the verbal section of the SAT, their high school GPA, and whether or not they are a first generation college student.

```
data(FirstYearGPA)
GPA <- FirstYearGPA
# Use only the non-null entries
GPA <- subset(GPA, !is.na(GPA)) #careful, this removes ALL observations with missing values
#even if some variables you aren't planning to use are causing the missingness issues
```

Previously, we determined that HSGPA does have a significant relationship with GPA when SATV and FirstGen are in the model, using a randomization test. But what are reasonable values of the slope coefficient? The randomization test doesn't give us a range of reasonable values - instead it gave us a range of values that would indicate NO relationship. We can use the bootstrap to obtain CIs for the slope coefficient of HSGPA for predicting GPA when SATV and FirstGen are in the model.

```
fm2 <- lm(GPA ~ HSGPA + SATV + FirstGen, data = GPA)
summary(fm2)
```

```
##
## Call:
## lm(formula = GPA ~ HSGPA + SATV + FirstGen, data = GPA)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.0077 -0.2660  0.0293  0.2970  0.8356
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.715988   0.295003    2.43    0.016 *
## HSGPA        0.518540   0.074956    6.92  5.2e-11 ***
## SATV         0.001012   0.000348    2.91    0.004 **
## FirstGen    -0.199837   0.089154   -2.24    0.026 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.402 on 215 degrees of freedom
## Multiple R-squared:  0.263,  Adjusted R-squared:  0.253
## F-statistic: 25.6 on 3 and 215 DF,  p-value: 3.35e-14
```

```
confint(fm2)
```

```
##                    2.5 %      97.5 %
## (Intercept)  0.134519969  1.29745550
## HSGPA        0.370798152  0.66628193
```

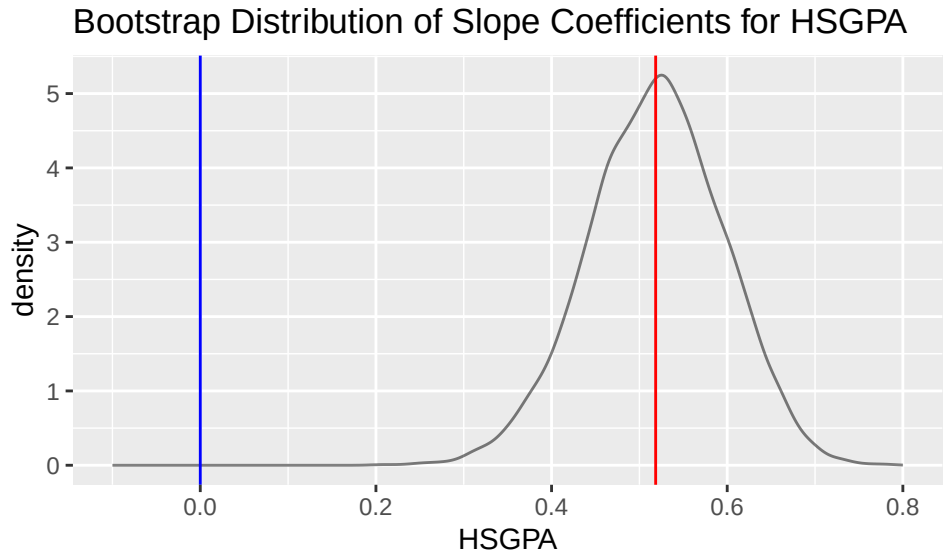
```
## SATV          0.000326881  0.00169803
## FirstGen     -0.375564386 -0.02411032
```

We would trust these CIs if the regression conditions are met. But let's see what the bootstrap CIs are anyway.

```
set.seed(118)
bootstrap <- do(10000) * lm(GPA ~ HSGPA + SATV + FirstGen, data = resample(GPA))$coefficients

gf_dens(~ HSGPA, data = bootstrap) %>%
  gf_labs(title = "Bootstrap Distribution of Slope Coefficients for HSGPA") %>%
  gf_lims(x = c(-0.1, 0.8)) %>%
  gf_vline(xintercept = ~ coef(fm2)["HSGPA"], color = "red") %>%
  gf_vline(xintercept = ~ 0, color = "blue")
```

```
## Warning: Removed 2 rows containing non-finite values (stat_density).
```



Now we generate CIs using the same methods as above:

Method #1 The first method assumes the bootstrap distribution is roughly symmetric.

```
qdata(~ HSGPA, c(0.025, 0.975), data = bootstrap)
```

```
##     2.5%    97.5%
## 0.363122 0.663599
```

Method #2

The second method assumes that the bootstrap distribution is **normal**.

```
mycoef <- coef(fm2)["HSGPA"]
mycoef + c(-1.96, 1.96) * sd(bootstrap$HSGPA) #95% CI
```

```
## [1] 0.367083 0.669997
```

Method #3 This final method is useful if the bootstrapped coefficients show a very skewed distribution. Here, that is not the case, so we'd expect to get something similar to what we obtained with Method 1 and 2. Remember, this means getting those t-star values so there is more work involved here.

```
set.seed(118) #use same seed
manyTs <- do(10000)*tstar(lm(GPA ~ HSGPA + SATV + FirstGen, data = resample(GPA)), fm2)
bootslopes <- manyTs[ ,1]
Ts <- manyTs[ ,2]
```

Now to find the quantiles we want, we find quantiles of the T* distribution to match our chosen confidence level.

```
qtl <- quantile(Ts, 0.025)
qtu <- quantile(Ts, 0.975)
c(qtl,qtu) #to see what they are
```

```
##     2.5%    97.5%
## -1.06145  1.49731
```

Here, these do not particularly demonstrate that our distribution was skewed - they are nearly symmetric around 0.

Now we can find the CI using our found T* quantiles as multipliers. Remember due to the skewness, the *upper* multiplier is from the lower tail of the t-star distribution, and vice versa.

```
SE <- sd(bootslopes)
c(mycoef - qtu*SE, mycoef - qtl*SE)
```

```
##    HSGPA    HSGPA
## 0.402837 0.600562
```

The interval is similar but shifted up and is a little narrower than that from Method 1 and 2.

You can bootstrap statistics besides slopes. For example, correlation coefficients are often bootstrapped, because their sampling distributions are often skewed.