Important Things about Statistics

Brenton Kenkel — PSCI 8357
January 14, 2016

To kick off the course, I gave you a quiz with the following questions:

1. What is a statistic?
2. What is a parameter?
3. What is an estimator?
4. What is a sampling distribution?
5. What is a standard error?

To do good, honest empirical work, you need to understand these concepts well. Conversely, most of the problems with empirical social science that we will discuss boil down to misunderstandings or neglect of these concepts.

Statistics

A **statistic** is a function of sample data. Let $X = (X_1, \ldots, X_N)$ denote our sample data. The most widely known statistic is the sample mean, defined by the function

$$m(X) = \frac{1}{N} \sum_{i=1}^{N} X_i.$$

Other statistics include the sample median, the sample variance, the minimum and maximum, and so on. Anything that's a function of the sample is a statistic.

I'm going to use a running example to illustrate each of the concepts I asked you about. Assume we have a sample $X = (X_1, \ldots, X_N)$ where each X_i is drawn independently from a uniform distribution on $[0, 1]$: $X_i \overset{i.i.d.}{\sim} U[0, 1]$. For reasons that will become clear later, assume that X is ordered from smallest to largest, so $X_1 \leq X_2 \leq \cdots \leq X_{N-1} \leq X_N$. Finally, assume we also observe a sample of $Y = (Y_1, \ldots, Y_N)$, where

$$Y_i = 1 + 2X_i + \epsilon_i,$$

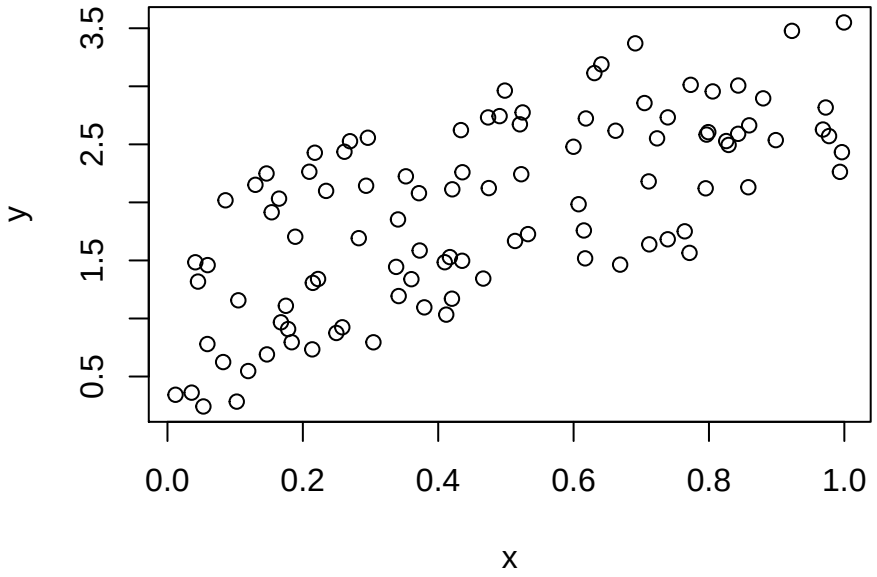
with $\epsilon_i \overset{i.i.d.}{\sim} U[-1, 1]$.

We can draw our very own sample from this distribution using the handy random number generation functions in R.

```
n_obs <- 100
x <- runif(n_obs, min = 0, max = 1)
x <- sort(x)
y <- 1 + 2 * x + runif(n_obs, min = -1, max = 1)
```

Let's take a look at the data we just simulated.

```
plot(x, y)
```



We can compute any manner of statistics from this sample.

```
mean(x)
```

```
## [1] 0.471
```

```
mean(y)
```

```
## [1] 1.94
```

```
sd(x)
```

```
## [1] 0.286
```

Even the scatterplot above is a statistic. Of course, the values of each of these

statistics would have been different if the sample had come out differently. In other words, a statistic is a random variable, just like the sample data itself. The same is not true for parameters, our next concept of interest.

Parameters

A **parameter** is a feature of the population of data. Unlike the value of a statistic, which depends on the particular sample and thus is variable, the value of a parameter is fixed. The most widely known example of a parameter is the population mean, $E[X]$. Another example is the population variance, $E[(X - E[X])^2]$.

In our running example, the population mean of each random draw X_i is $E[X_i] = 0.5$. The population mean of each Y_i is

$$E[Y_i] = E[1 + 2X_i + \epsilon_i] = 1 + 2E[X_i] + E[\epsilon_i] = 2.$$

These don't change with the sample. No matter which data we draw, the parameters remain the same.

In regression analysis, we care about a particular type of parameter: the conditional expectation of Y_i, given some value of X_i. In other words, if we fix the value of the random draw X_i to be some specific quantity x, what is the expected value of the random draw Y_i? In our running example, we have

$$\begin{aligned} E[Y_i \,|\, X_i = x] &= E[1 + 2x + \epsilon_i] \\ &= 1 + 2x + E[\epsilon_i] \\ &= 1 + 2x. \end{aligned}$$

To get ahead of ourselves a bit, let's call $f(x) = E[Y_i \,|\, X_i = x]$ the *regression function*. Specific values of the regression function, like $f(0) = 1$ and $f(1) = 3$, are parameters. So are its derivatives. When the regression function is affine, such as the one in the running example, its derivatives are constant. So we can define yet another parameter by

$$\beta = \frac{df(x)}{dx}.$$

In the running example, we have $\beta = 2$.[1]

Whereas statistics are variable functions of the sample data, parameters are fixed features of the population. We often use statistics to make inferences about parameters. That brings us to the topic of estimators.

Estimators

An **estimator** is a statistic used to estimate a parameter. This definition seems a bit tautological. It is. Take any statistic, say "I am using this statistic to estimate such-and-such parameter," and—ta da! You have an estimator. It won't be a very *good* estimator if you didn't choose your statistic carefully, but it will be an estimator nonetheless. The important part is that an estimator, being a statistic, is a function of sample data.

The best-known estimator is the sample mean $m(X)$ used to estimate the population mean $E[X]$. The sample mean is a great estimator, but not such a great *example* of an estimator. It is simple to calculate, intuitive to understand, and has good statistical properties. It's rare for an estimator to check off all these boxes. For example, you probably saw in introductory statistics that the somewhat unintuitive

$$v(X) = \frac{1}{N-1}\sum_{i=1}^{N}(X_i - m(X))^2$$

is our usual estimator for the population variance.

Let's return to our running example. Suppose we want to estimate β, the coefficient on x in the linear regression function $f(x) = 1 + 2x$, whose true value is 2. In introductory statistics, you learned about the ordinary least squares estimator of β. Today we'll work with a much worse estimator. We're going to connect the dots at the lowest and highest values of X_i, and measure the slope of the resulting line. Formally, this estimator is defined by

$$b(X, Y) = \frac{Y_N - Y_1}{X_N - X_1}.$$

[1] If the regression function were not linear, the derivative would vary across values of x, and thus could not be described by a single constant. For example, consider the model $Y_i = X_i^2 + \epsilon_i$. This yields the regression function $f(x) = x^2$, with derivative $df(x)/dx = 2x$.

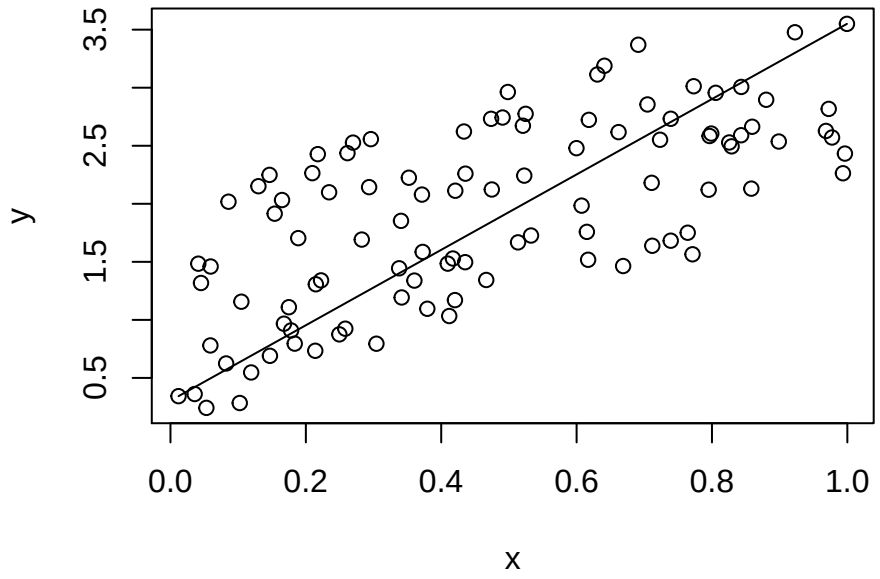
Since we `sorted` `x` as soon as we drew its values, we can easily calculate an estimate using this estimator.

```
b <- (y[n_obs] - y[1]) / (x[n_obs] - x[1])
b
```

```
## [1] 3.25
```

Let's visualize how we arrive at this estimate.

```
plot(x, y)
segments(x[1], y[1], x[n_obs], y[n_obs])
```



Not bad, eh? (Just kidding. It is bad.)

Like any statistic, the value of an estimator will turn out differently depending on the sample data. We like estimators whose average value equals the true parameter value (unbiased), that almost always get close to the true value if we have enough data (consistent), and that have a relatively low average margin of error (efficient). To figure out if these properties hold for a particular estimator, we need to know how the estimator is distributed, given the distribution of the data.

Sampling Distributions

A **sampling distribution** is the probability distribution of a statistic. I think of it like this. The sample data come from a distribution. A statistic is a function of sample data—for each different way the sample could turn out, there is a corresponding way the statistic would turn out. So the distribution of the data induces a distribution of the statistic.

Let's take a look at the sampling distribution of our estimator $b(X, Y)$. We already saw the value of $b(X, Y)$ for the one particular sample we drew. But that was just one draw from the sampling distribution of $b(X, Y)$. To approximate the full distribution, we're going to use the *Monte Carlo method*. We'll draw a large number of samples of the data (X, Y) from their probability distribution. Then we'll compute and save the estimate $b(X, Y)$ for each sample we drew.

The easiest way to run a Monte Carlo simulation like this is with the `replicate()` function. I'll first illustrate the function by showing how we can take 5 samples of $X \overset{i.i.d.}{\sim} U[0, 1]$, each with $N = 3$.

```
replicate(5, runif(3, min = 0, max = 1))
```

```
##        [,1]   [,2]  [,3]  [,4]  [,5]
## [1,] 0.0434 0.9419 0.703 0.125 0.550
## [2,] 0.4292 0.0791 0.792 0.388 0.239
## [3,] 0.2121 0.7077 0.345 0.530 0.782
```

`replicate()` takes the expression in its second argument, runs it the number of times given to its first argument, and stores the results from each run. It's much nicer than writing a `for` loop. Now let's use it to draw from the sampling distribution of our estimator.

```
n_mc <- 1000
n_obs <- 100
sampling_distribution <- replicate(n_mc, {
    x <- runif(n_obs, min = 0, max = 1)
    x <- sort(x)
    y <- 1 + 2 * x + runif(n_obs, min = -1, max = 1)
    b <- (y[n_obs] - y[1]) / (x[n_obs] - x[1])
    b
})
```

Is our estimator unbiased—on average, does it get the right result? The only way to show for sure that an estimator is unbiased (or not) is through a mathematical proof, but a Monte Carlo simulation usually gets us pretty close.

```
mean(sampling_distribution)
```

```
## [1] 2
```

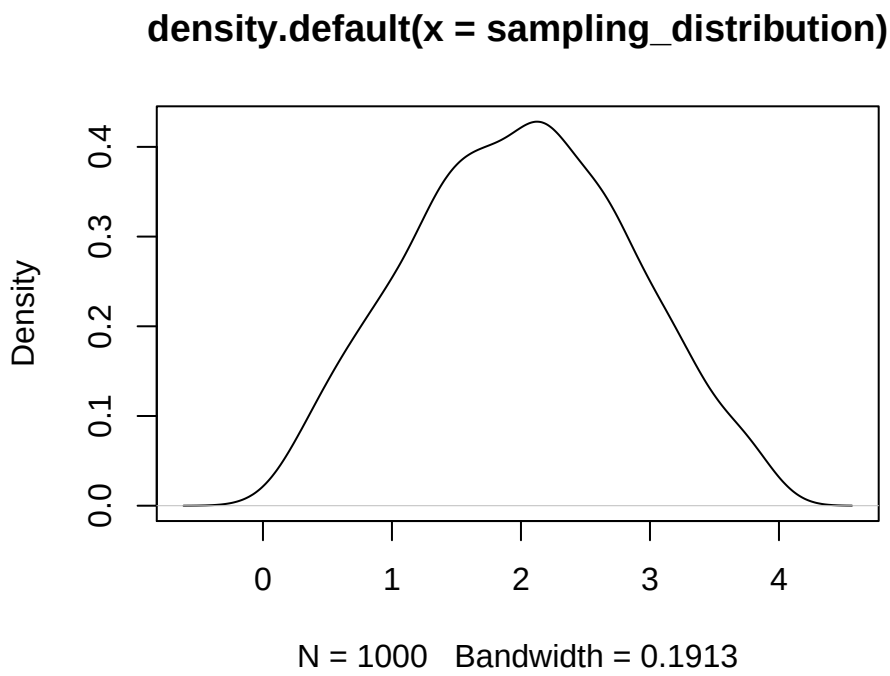
Well that's pretty suggestive. Let's see some more digits.

```
format(mean(sampling_distribution), digits = 10)
```

```
## [1] "1.998875886"
```

If I had to guess, I'd guess our estimator is unbiased. Now let's take a look at the shape of the full sampling distribution.

```
plot(density(sampling_distribution))
```



Looks normal-ish, and centered around the true parameter value of $\beta = 2$. Hey, maybe this estimator isn't so bad after all! (It is. I swear.)

Standard Error

A **standard error** is the standard deviation of a sampling distribution. The bigger the standard error, the more variable the statistic is across samples. It's easier to make inferences from a statistic with a lower standard error, since it is less liable to vary wildly across samples.

Since the standard error is a property of the population distribution of a statistic, which in turn is a function of the population distribution of data, this means standard errors are parameters. Like any parameter, their value is fixed.

Wait a minute. I just said standard errors are fixed parameters. But my statistical software spits out standard errors. And I know those are a function of the data.

```
ols_xy <- lm(y ~ x)
summary(ols_xy)
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.9567 -0.5005 -0.0759  0.5032  1.0148
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)    1.047      0.112    9.32  3.7e-15
## x              1.893      0.204    9.26  4.9e-15
##
## Residual standard error: 0.581 on 98 degrees of freedom
## Multiple R-squared:  0.467,  Adjusted R-squared:  0.461
## F-statistic: 85.8 on 1 and 98 DF,  p-value: 4.86e-15
```

See? Standard errors! It says it right there! The standard error of the coefficient on X_i is 0.204.

Wrong. Those are *estimated* standard errors. As researchers, we usually don't know the sampling distribution of the estimator we're using. The sampling distribution of any decent estimator depends on the true parameter value, and

if we knew that, we wouldn't have to estimate it in the first place. So to quantify our uncertainty about our estimate, we also need to estimate its standard error. Whenever an applied paper talks about "the standard errors," it means "the estimated standard errors" or, equivalently, "the standard error estimates".

Returning to the running example, let's use our Monte Carlo simulation results to approximate the standard error of our pet estimator.

```
sd(sampling_distribution)
```

```
## [1] 0.846
```

That's more than four times the estimated standard error of OLS. Maybe the OLS estimate of the standard error is way, way off. Just to check, let's simulate its standard error.

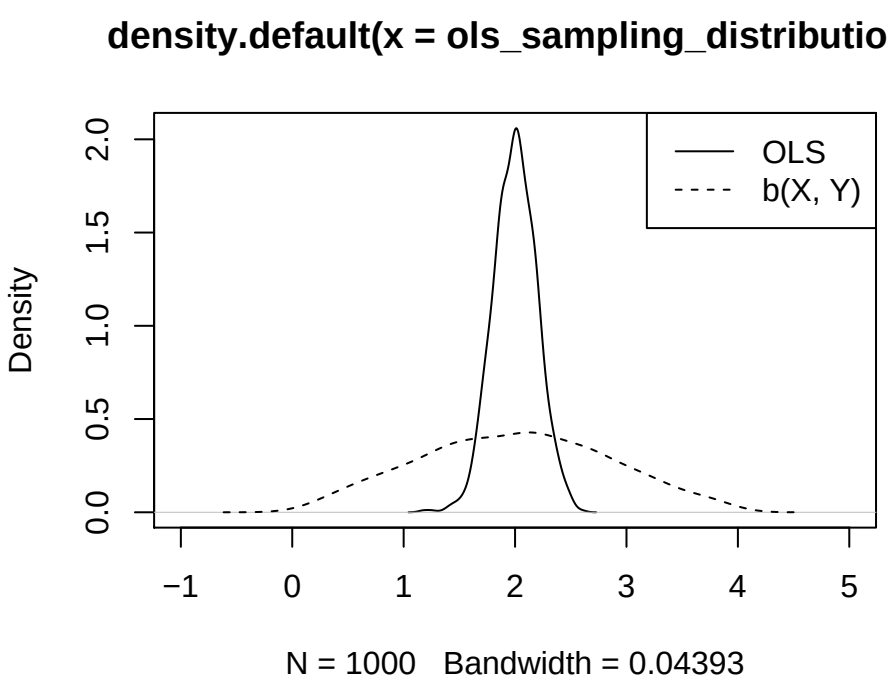
```
ols_sampling_distribution <- replicate(n_mc, {
    x <- runif(n_obs, min = 0, max = 1)
    x <- sort(x)
    y <- 1 + 2 * x + runif(n_obs, min = -1, max = 1)
    fit_ols <- lm(y ~ x)
    coef(fit_ols)["x"]
})

sd(ols_sampling_distribution)
```

```
## [1] 0.195
```

Nope—it's less than one-fourth that of the estimator we invented. Let's compare their densities.

```
plot(density(ols_sampling_distribution), xlim = c(-1, 5))
lines(density(sampling_distribution), lty = 2)
legend("topright", c("OLS", "b(X, Y)"), lty = c(1, 2))
```



Both are centered around the true value of $\beta = 2$, but the OLS estimate is much more likely to be close to the true parameter since its standard error is so much lower. This is why our invented estimator is bad: it's so inefficient compared to OLS.