PSTAT 5LS Lab 2

Fall 2025

Section 1

Announcements & Recap

Section 2

Learning Objectives

R Learning Objectives

1. Learn about reproducible randomness by “setting seeds”
2. Understand the basics of simulation in R

Statistical Learning Objectives

1. Understand how to translate a real data scenario into a simulation setup
2. Explore sample-to-sample variability by observing a distribution of simulated p-hat values.

Functions Covered in This Lab

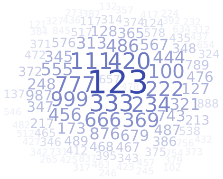
1. `set.seed()`
2. `simulate_chance_model()`
3. `sum()`

Section 3

Lab Tutorial

Pseudo-Random Numbers

It turns out that humans are very bad at generating numbers randomly. Take a look at this word cloud of results from a survey of students in an introductory statistics course, in which students were asked to provide a random three-digit number. The larger a number is, the more often it showed up in the data.



As you can see, people tended to choose 123 a lot, or just typed the same number three times. If these numbers were *truly* random, we'd wouldn't see this – we'd see most three digit numbers in the data, and each would appear only a few times.

What Is a Seed?

Computers are *much* better at randomness than humans are, but they're actually still quite bad. "Random" numbers generated by computers are meant to *look* and *seem* random, but, the sequences of numbers they generate are completely predictable if you know where that sequence starts. That starting point is called a **seed**.

Think of the seed as, well, a seed: if you know what kind of seed you plant, you know exactly what kind of flowers will grow. By setting a seed, you can predict the exact sequence of "random" numbers that will grow from it.

You can choose the seed that your R session will use with a function called `set.seed()`.

Setting a Seed

Run the `setSeed` code chunk in your notes document. No output will occur, but it will set the seed.

```r
set.seed(2573)
```

We chose the number 2573 arbitrarily.

If you execute the code chunk that contains the `set.seed()` code again, it will start your random number generator at the beginning of the sequence.

When you knit your document, it will also start your random number generator at the beginning of the sequence.

Simulating a Chance Model

In lecture, we are learning about how to simulate results using a chance model (that assumes the null hypothesis is true) to evaluate the probability of getting results like we saw in the data or something even more extreme.

Professor Miller's former colleagues Nick Seewald and Elaine Hembree created a special package called `stats250sbi` that is available for install through GitHub. (SBI stands for "simulation-based inference.") Our lab projects will have this package pre-installed; you can find instructions on how to install it for a new RStudio project you start in JupyterHub or on your personal RStudio installation at the bottom of your notes document.

Let's take a look at our first SBI function, `simulate_chance_model()`.

simulate_chance_model()

In order to understand our new function, let's utilize the help feature.

```
?simulate_chance_model
```

You have a spot to run this code in the `aboutSimulate` code chunk.

Notice that the help feature appears in the bottom right pane of the RStudio environment. The help document describes the arguments used in the `simulate_chance_model()` function.

simulate_chance_model()

So the function takes on these arguments:

- `chanceSuccess`: a number between 0 and 1 representing the probability of observing a "success"
- `numDraws`: the number of times to draw a poker chip from the bag needed to complete one repetition of the simulation (this will be equal to the sample size)
- `numRepetitions`: the number of times to repeat the simulation process (this will be an arbitrary large number so that we can see what the chance model looks like)

Think of a "success" as answering yes to a particular question. For example, a "success" in the Doris and Buzz example occurs when Buzz pushes the correct button.

We can now apply this to an example!

Simulation Example

Lee is a teacher at a local high school who wanted to assess whether or not dogs physically resemble their owners enough for people to be able to correctly match a dog to their owner better than if just guessing. Lee, who is also a dog owner, showed pictures of four dogs to her class of 40 students. One photo was of the teacher's dog (Yoda) and the other three photos were of dogs belonging to Lee's friends. The students were asked to guess which dog was actually the teacher's. If dogs do not physically resemble their owners, the students would get a correct match 25% of the time, since the students would be equally likely to choose any of the four dogs. It turned out that 17 of the 40 students correctly picked out the teacher's dog.

Let's go through the entire procedure from start to finish.

Sample Proportion

What is $\hat{p}$, the observed (sample) proportion of correct guesses? Use the `sampProp` code chunk to calculate this point estimate.

Sample Proportion

What is $\hat{p}$, the observed (sample) proportion of correct guesses?

```
phat <- 17/40
phat
```

```
## [1] 0.425
```

Hypotheses

What are the hypotheses to be tested? State the hypotheses using symbols. Be sure to define the parameter. You'll want to define both H_0 and H_A.

Hypotheses

What are the hypotheses to be tested? State the hypotheses using symbols. Be sure to define the parameter. You'll want to define both H_0 and H_A.

$H_0 : p = 0.25,$

$H_A : p > 0.25,$

where p represents the **proportion** of students who **correctly** guess which dog belongs to their teacher.

Setting Up the Simulation

	Assuming the chance model...
One draw	
Blue poker chip	
Yellow poker chip	
Chance of blue	
One repetition	

You have a table just like this in your notes file. Follow along and fill it in!

Small Number of Repetitions

Let's try this out for a small number of repetitions, say, 10, so that we can see what the output is from the function.

```
##  [1] 0.175 0.275 0.325 0.250 0.225 0.250 0.300 0.275 0.225 0.175
```

The output is a vector of the simulated proportions, the simulated $\hat{p}$ values, under the chance model specified.

If we are going to simulate 100 times, 1000 times, 10000 times, we don't want to see the long output! So in the future, we will assign the output to a variable name so that we don't have to read pages and pages of output.

Code to Simulate

In the runSim code chunk in your notes file, add the values for chanceSuccess, numDraws, and numRepetitions to run the simulation 1000 times. Then run the chunk.

```
sim1 <- simulate_chance_model(chanceSuccess = 0.25,
                              numDraws = 40,
                              numRepetitions = 1000)
```

sim1 should now be in your environment in the top right corner.

Notice that the first time you run this code, you get the **exact same values** we got! This is because we both set the same seed! If you run the code again, your values will change and will differ from ours. This is because the number of times that you run your code dictates where your random number generator is in the sequence of random numbers.

Pro Tip: Use your **knitted** document's output to view the first set of random numbers being generated!

Making a Visualization of the Simulation Results

So, if we assigned the output to a variable name, how do we view the results?

Make a histogram!

We will make a histogram of this *numeric* variable `sim1`. In our histogram, we will introduce a new argument called `xlim` that will allow us to set the minimum and maximum values on the x-axis. Since proportions are always between 0 and 1, setting these values as the minimum and maximum is a safe bet. Feel free to change these values at your discretion.

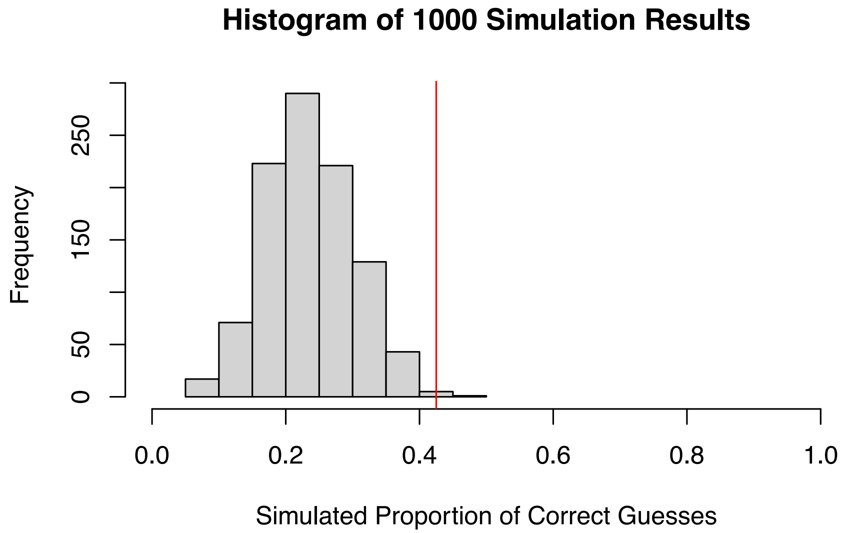
We will also make a vertical red line at the sample proportion so that we can see how the sample proportion compares to the hypothesized value of the population proportion (the 0.25 from our null hypothesis).

Histogram Code

Let's try this code out in the simHist code chunk. The code is ready for you to run in your notes document.

```r
hist(sim1,
     main = "Histogram of 1000 Simulation Results",
     xlab = "Simulated Proportion of Correct Guesses",
     xlim = c(0, 1))
abline(v = 17/40, col = "red")
```

Histogram of Simulation Results



Finding the Probability of Getting This or More Extreme

As we learned in lecture, we want to calculate the *probability that we will get our observed* proportion or a result that is *even more extreme* than what we observed.

Since $H_A : p > 0.25$, the "as or more extreme" is actually **greater than or equal to** $\hat{p}$ of $17/40 = 0.425$.

Let A represent something of interest to us. In this example, an observation in A is a simulated proportion of $17/40$ or greater. The number of observations in A is the sum of the simulated proportions that are $17/40$, $18/40$, $\ldots$, or $40/40$ To find the probability of A in general, we use this formula:

$$P(A) = \frac{\text{number of observations of A}}{\text{total observations}}$$

Finding the Probability of Getting This or More Extreme

So we will be:

1. looking for the number of observations that have a simulated proportion of 17/40 **or greater**
2. divide this number of observations by **1000**, the number of times we ran the simulation

Code to Find the Estimated p-value

Enter the following in the `estpvalue` code chunk in your notes.

```r
sum(sim1 >= 17 / 40) / 1000
```

Let's break this down:

- **`sim1`** is the numeric variable representing a vector of the 1000 simulation proportions
- the **`>=`** operator allows us to find things that are greater than or equal to 17/40 in the `sim1` variable. We do this because our alternative hypothesis has a greater than ($>$) sign
- the `sum()` function will add up the number of observations that meet this criteria. So here, it will find the number of observations of simulation proportions that are greater than or equal to 17/40
- divide the result of `sum()` by the total number of times the simulation was run

So What is the Probability?

The probability is estimated to be

```
## [1] 0.006
```

The probability of getting our observed sample proportion of 17/40 **or greater** is estimated to be 0.006.

Conclusion

What do the results tell us about our research question?

Do we have enough evidence to support the claim that dogs physically resemble their owners enough for people to be able to correctly match a dog to their owner **better** than if just guessing?