Models and Simulation

Simulation is a powerful tool in data science. It allows us to set two different viewpoints of the world, and then find out which view is more reliable. We can also measure how reliable this is using something called the **p-value**. This whole process is called **statistical inference** (the process of making conclusions based on random samples).

Let's get some terminology out of the way first:

Parameter: Some number that comes from the population
Statistic: Some number that's calculated from a sample of the population

Simulating a Statistic

Simulating a statistic basically means take a random sample of the population, calculate the statistic you want to calculate, then create a histogram and observe the data you just created. Why is this important? Well we end up simulating a statistic it estimates what the population parameter actually is. If we knew the population parameter, then we wouldn't have to estimate it! So here are the steps to simulating a statistic:

1. Figure out the code to create one statistic from some sample of the data
2. In a separate cell, create an empty array (to store the statistics you'll be calculating)
3. Run a for loop many times (like 1000)
    a. Each iteration, simulate one statistic using the code from step 1 (making sure you have a different sample each time)
    b. Append the new statistic to your stored statistics
4. Draw out a histogram of your simulated statistics

Let's go through an example on. Let's say you are given a coin and you are told that it is supposed to land on heads 70 times out of 100 flips. *How would you simulate the statistic of the probability of heads in 100 coin flips?*

Remember, the first step is to write code to simulate that data:

```
In  [1]: coin_probs = make_array(0.7, 0.3) #item 0: prob of heads
         sample_proportions(100, coin_probs).item(0)

Out [1]: 0.73
```

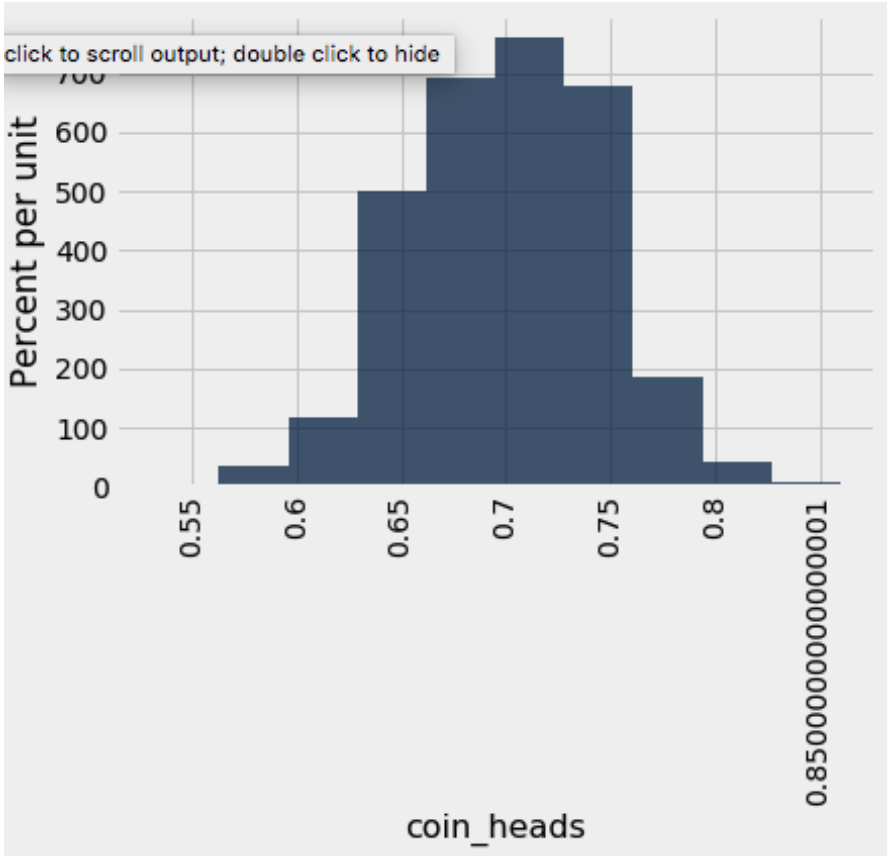
Sample proportions does two steps in one, it does the random sampling and finding the test statistic for us! Look more in the python reference sheet if you are still confused.

Now this simulates a set of 100 coin flips, where we found that 73 of them were heads and 27 were tails.

Now for the simulating part. Since we know how to create one sample, we do these many times and add those simulated statistics to an array:

```
In [2]: coin_probs = make_array(0.7, 0.3) #item 0: prob of heads
        coin_heads = make_array()
        for i in np.arange(10000):
            head_prop = sample_proportions(100, coin_probs).item(0)
            coin_heads = np.append(head_prop, coin_heads)
        Table().with_column('coin_heads', coin_heads).hist()
```

Out [2]:



Here's your histogram of simulated proportions. So that's how you simulate a statistic. Now simulating this doesn't have much value unless we it has some purpose, like to compare our simulated statistic to reality. We do this through hypothesis tests.

Hypothesis Testing and Statistical Inference

Hypothesis tests will hypothesize how some data came to be. The results are done by comparing the simulated test statistics with an observed statistic seen in reality. Before we go through an example, let's look at some terminology:

Null Hypothesis: We're going to have some question we want to answer based on the data we're given, specifically how that data was generated. The **null hypothesis** describes a well-defined chance model about how that data was generated. The important part about the null hypothesis is that we can simulate data under its assumptions.

Alternative Hypothesis: This is a different take on how the data was generated.

Let's go through some examples:

Robert flips a coin 10 times, and 7 of the 10 times it showed heads. You are skeptical and think that he's using an unfair coin. If you were to simulate this, what would your null and alternative hypotheses be?

Null: Remember that we have to be able to simulate under this. We can say something like "*The coin is fair and any deviations are due to random chance"*. This way, we can simulate this because we know that the probability flipping heads for a fair coin is 50%.

Alternative: *"The coin is not fair"*. This is a different take than the null hypothesis.

- Note that the alternative doesn't specify whether the coin is biased toward heads or tails
- Also note that when doing hypothesis tests, you are deciding the more accurate of the two hypotheses. That does not mean it is the most accurate hypothesis, it's just the most accurate of the two.

Now we're ready for an example:

You go to some local coffee shop and buy a skinny vanilla latte everyday. You do this because for some reason, the skinny vanilla latte is cheaper than the vanilla latte, even though it tastes worse. You noticed though, that the syrup company doesn't label the vanilla latte and skinny vanilla latte's bottles. You've noticed that out of the past 100 days you've gone there, 90 of them have actually been vanilla lattes and not skinny vanilla lattes :gasp:! (You've been going there a very long time). You've also gathered data before and found out that the barista that always makes 85% of the skinny vanilla lattes for you as regular vanilla lattes because the bottle is closer and easier to reach. You want to see whether she's giving more attention to you than other customers, or whether this is all just random. Maybe it's a sign, or maybe it's nothing, only data science has the answer!

First, let's set up a null and alternative hypothesis. Try to do it yourself before you look on the next page for the answer, and if you're up to it, try and do the whole simulation before looking on the next page too.

Null: The barista uses regular vanilla over skinny vanilla 85% of the time and any deviation is due to chance.
Alt: The barista is giving you vanilla lattes way more than than normal (this deviation is not due to chance)

Simulate one statistic:

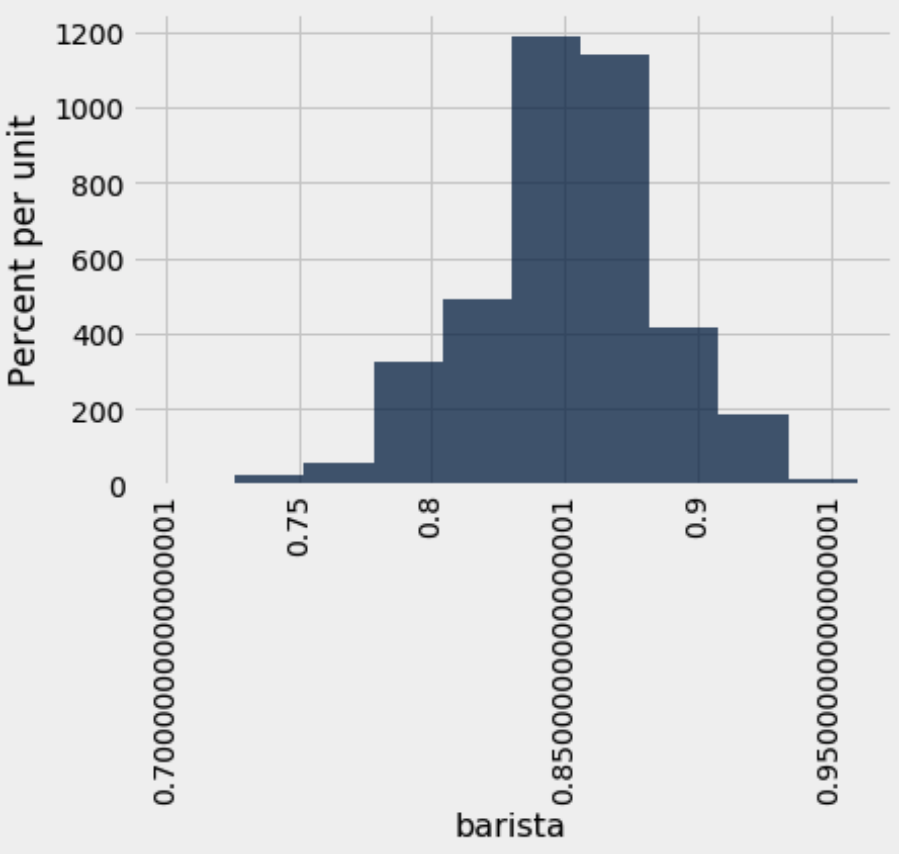
```
In  [3]: probs = make_array(0.85, 0.15) #item 0: prob of vanilla
         sample_proportions(100, probs).item(0)

Out [3]: 0.88
```

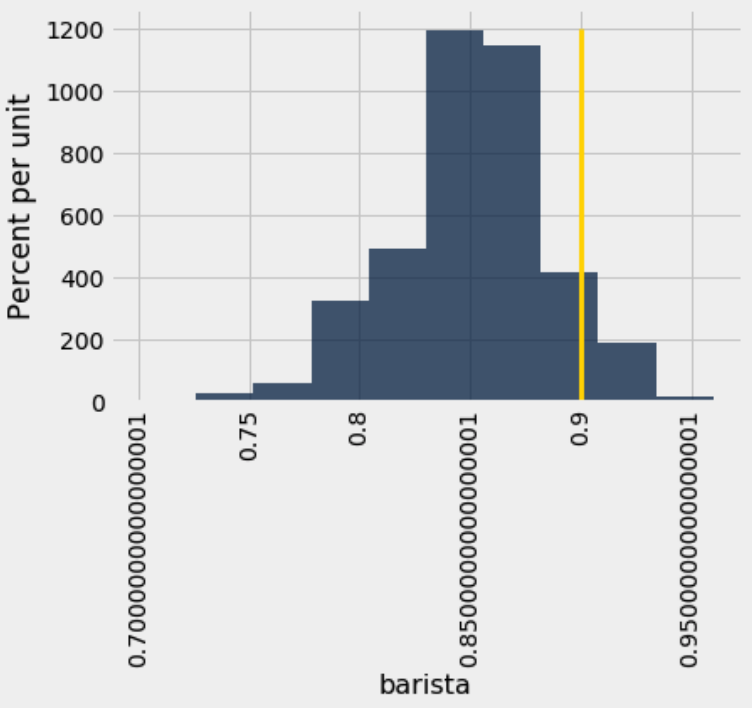
Simulate Many and Visualize Results

```
In  [4]: probs = make_array(0.85, 0.15) #item 0: prob of heads
         barista_stats = make_array()
         for i in np.arange(10000):
             drinks = sample_proportions(100, coin_probs).item(0)
             barista_stats = np.append(drinks, barista_stats)
         Table().with_column('barista', barista_stats).hist()
```

Out [4]:



Great, we have our simulated statistics. Now we need to find where our observed statistic lies. Remember, our observed statistic was 90 out of 100 drinks, so 0.9. That lays here:



So now we can see where our observed test statistic lied. Let me describe how I like to see this picture.

So the way I see it, every simulated statistic is the barista creating another 100 drinks for you in some alternate universe under the exact same circumstances. We computed 10000 alternate universes and we found that most of the universes would have given you around 85 regular vanilla lattes instead of 90. However, there were some alternate universes that gave you more, and many that gave you less. The gold line is our universe, what actually happened. Remember, that this is assuming the null hypothesis is true.

Now, given that the null hypothesis is true, what's the likelihood we ended up getting the gold universe instead of the other simulated blue universes? If that probability is really low, then it's likely that the null hypothesis is actually false since it's very unlikely. However, if the probability is high, then the alternative hypothesis is probably false. This probability is called a **p-value**. **More formally defined, it's the chance under the null hypothesis that the test statistic is equal to the value that was observed in the data or further in the direction of the alternative** (taken from slides created by John DeNero and Adi Adhikari: http://data8.org/materials-su18/lec/lec16.pdf). Typically if the p-value is below five percent, we reject the null hypothesis and if it's above then we fail to reject the null.

Let's calculate the p-value on the next page.

```
In  [3]: np.count_nonzero(barista_stats >= 0.9) / len(barista_stats)

Out [3]: 0.0952
```

Remember that `barista_stats >= 0.9` will give an array of a bunch of True's and False's, and `np.count_nonzero` will count the number of True's (because False evaluates to 0 in python).

Since the p-value is about 9.52 percent, we will fail to reject the null hypothesis. So we can't say that the barista was giving us more vanilla lattes than normal.