lab06

October 24, 2024

[17]:
```
# Initialize Otter
import otter
grader = otter.Notebook("lab06.ipynb")
```

0.1 Lab 6: Examining the Therapeutic Touch

Welcome to Lab 6!

After such an extensive introduction to programming for data science, we are finally moving into the section of the course where we can apply our new skils to answer real questions.

In this lab, we'll use testing techniques that were introduced in lecture to test the idea of the therapeutic touch, the idea that some practitioner can feel and massage your human energy field.

[18]:
```
# Run this cell, but please don't change it.

# These lines import the Numpy and Datascience modules.
import numpy as np
from datascience import *

# These lines do some fancy plotting magic
import matplotlib
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
import warnings
warnings.simplefilter('ignore', FutureWarning)
from matplotlib import patches
from ipywidgets import interact, interactive, fixed
import ipywidgets as widgets
```

1 Lab Warm Up!

We will work together as a class in the following coding cells to prepare you for all sections of this lab.

Make sure to come to lab on time so you don't miss points for this warm-up!

Let's review hypothesis testing teminology, click in this markdown cell and add what we discuss as a class:

A null hypotheses (model we can simulate from) Alternative hypotheses (alternative model)

What can we easily simulate using code? A chance model –> simulation of random chance

What is a statistic? A single number summary from a set of data, (mean, median, proportion)

Simulating randomness, what function shave we learned so far? tbl.sample randomly samples rows from a table np.random.choice randomly samples from an array sample_proportions is a function that samples from a categorical distribution. There are two inputs: 1. sample size 2. a distribution of the categories in the population, as a list of array of proportions that add up to 1. (hint: distribtion of sides of a coin: make_array(0., 0.5))

ouput: 1. an array of th the proportios of the saampled elements in each cateory

[19]:
```
# Proportions
# Consider a high school where the student population identifies as follows:
# 46% male, 49% female & 5% non-binary
student_proportions = make_array(0.46, 0.49, 0.05)
student_proportions
```

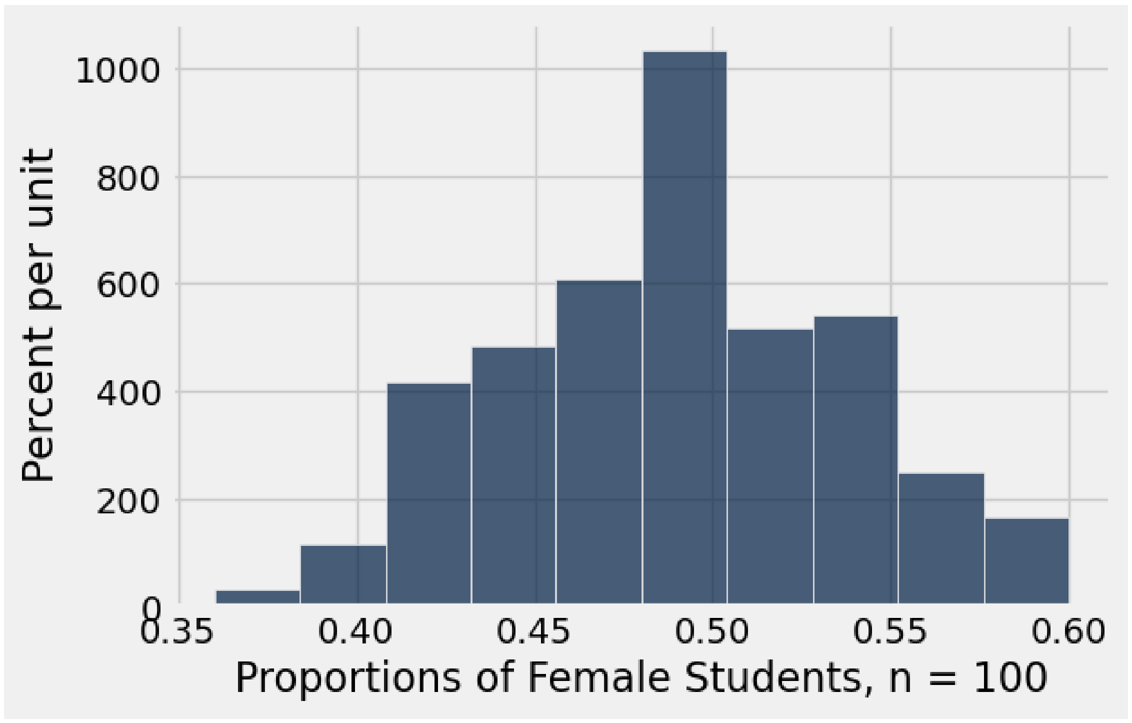
[19]:
```
array([ 0.46,  0.49,  0.05])
```

[20]:
```
# Taking 100 random samples from the above population (run this 3 times in a␣
↪row)
sample_proportions(100, student_proportions)
```

[20]:
```
array([ 0.39,  0.55,  0.06])
```

[21]:
```
# Simulating 500 times, save proportion of female students in an array,␣
↪visualize
num_repetitions = 500
simulated_statistics = make_array()
for i in range(num_repetitions):
    statistic = sample_proportions(100, student_proportions).item(1)
    simulated_statistics = np.append(simulated_statistics, statistic)

t=Table().with_column('Proportions of Female Students, n = 100',␣
↪simulated_statistics)
t.hist()
```



[22]:
```
# Let's look at today's example: Examining Therapeutic Touch
# We'll work on 1.1 - 1.3 as a class
```

1.1 1. What is the Therapeutic Touch

The Therapeutic Touch (TT) is the idea that everyone can feel the Human Energy Field (HEF) around individuals. Those who practice TT have described different people's HEFs as "warm as Jell-O" and "tactile as taffy."

TT was a popular technique used throughout the 20th century that was toted as a great way to bring balance to a person's health. Certain practitioners claim they have the ability to feel the HEF and can massage it in order to promote health and relaxation in individuals.

1.1.1 Emily Rosa

Emily Rosa was a 4th grade student who was very familiar with the world of TT, thanks to her parents, who were both medical practitioners and skeptics of TT.

For her 4th grade science fair project, Emily decided to test whether or not TT practitioners could truly interact with a person's HEF. She later went on to publish her work in TT, becoming the youngest person to have a research paper published in a peer reviewed medical journal.

1.1.2 Emily's Experiment

Emily's experiment was clean, simple, and effective. Due to her parents' occupations in the medical field, she had wide access to people who claimed to be TT practitioners.

Emily took 21 TT practitioners and used them for her science experiment. She would take a TT practitioner and ask them to extend their hands through a screen (which they can't see through). Emily would be on the other side and would flip a fair coin. Depending on how the coin landed, she would put out either her left hand or her right hand. The TT practitioner would then have to answer which hand Emily put out. If a pracitioner could truly interact with a person's HEF, it would be expected that they answered correctly.

Overall, through 210 samples, the practitioner picked the correct hand 44% of the time.

Emily's main goal here was to test whether or not the TT practicioners' guesses were random, like the flip of a coin. In most medical experiments, this is the norm. We want to test whether or not the treatment has an effect, *not* whether or not the treatment actually works.

We will now begin to formulate this experiment in terms of the terminology we learned in this course.

Question 1.1: Describe Emily's model for how likely the TT practitioners are to choose the correct hand. What alternative model is her model meant to discredit? Discuss with students around you to come to a conclusion. Check in with your instructor or tutor if you are stuck.

Null model: that the TT practioners' guesses were random like the flip of a coin Alternative model

Question 1.2: Remember that the practitioner got the correct answer 44% (0.44) of the time. According to Emily's model, on average, what proportion of times do we expect the practitioner to guess the correct hand? Make sure your answer is a number between 0 and 1.

```
[23]: expected_proportion_correct = 0.5
      expected_proportion_correct
```

```
[23]: 0.5
```

```
[24]: grader.check("q1_2")
```

```
[24]: q1_2 results: All test cases passed!
```

The goal now is to see if our **deviation** from this expected proportion of correct answers is due to something other than chance.

Question 1.3: We usually use a statistic to help determine which model the evidence points towards. What is a statistic that we can use to compare outcomes under Emily's model to what was observed? Assign `valid_stat` to the integer that represents the best test statistic that Emily can use:

1. The difference between the expected percent correct and the actual percent correct
2. The absolute difference between the expected percent correct and the actual percent correct
3. The sum of the expected percent correct and the actual percent correct

```
[25]: valid_stat = 2
      valid_stat
```

```
[25]: 2
```

[26]:
```
grader.check("q1_3")
```

[26]:
```
q1_3 results: All test cases passed!
```

Question 1.4: Why is the statistic from Question 1.3 the appropriate choice for comparing outcomes in Emily's experiment? How does it relate to the models you defined in Question 1.1?

Why the stastic from question 1.3 is the appropriate choice for comparing outcomes in Emily's experiment is because there is a 50 chance for the TT practioners expected percent correct and actual percent correct that means using absolute difference of both can tell us how far the practioners were from jsut random guessing. Using the absolute calue tells how big the difference is without worrying if its more or less. Because you'd expect the practioners success rate to be higher than 50% but eventually ended up being 44% which is below ranom chance this supports the null model that they were just guessing and the use of the absolute value can help us tell more clearly.

Question 1.5: Define the function `statistic` which takes in an expected proportion and an actual proportion, and returns the value of the statistic chosen in Question 1.3. Assume that the argument takes in proportions, but return your answer as a percentage.

[27]:
```
#HINT: Remember we are asking for a **percentage**, not a proportion.
def statistic(expected_prop, actual_prop):
    return (abs(expected_prop - actual_prop)) * 100

#statistic(50, 44)
```

[28]:
```
grader.check("q1_5")
```

[28]:
```
q1_5 results: All test cases passed!
```

Question 1.6: Use your newly defined function `statistic` to calculate the observed statistic from Emily's experiment.

[29]:
```
observed_statistic = statistic(0.5, 0.44)
observed_statistic
```

[29]:
```
6.0
```

[30]:
```
grader.check("q1_6")
```

[30]:
```
q1_6 results: All test cases passed!
```

Is this observed statistic consistent with what we expect to see under Emily's model?

In order to answer this question, we must simulate the experiment as though Emily's model was correct, and calculate our statistic for every simulation.

1.1.3 `sample_proportions`

`sample_proportions` can be used to randomly sample from multiple categories when you know the proportion of data points that are expected to fall in each category. `sample_proportions` takes two arguments: the sample size and an array that contains the distribution of categories in the population (should sum to 1).

Consider flipping a fair coin, where the two outcomes (coin lands heads and coin lands tails) occur with an equal chance. We expect that half of all coin flips will land heads, and half of all coin flips will land tails.

Run the following cell to see the simulation of 10 flips of a fair coin. Let the first item of `coin_proportions` be the proportion of heads and the second item of `coin_proportions` be the proportion of tails.

Observe what happens when you run this cell multiple times—the proportion of coin flips that land heads and tails appears to change, as you are simulating flipping 10 coins each time!

```
[31]: coin_proportions = make_array(0.5, 0.5)
      ten_flips = sample_proportions(10, coin_proportions)
      ten_flips
```

```
[31]: array([ 0.4,  0.6])
```

`sample_proportions` returns an array that is the same length as the proportion array that is passed through. It contains the proportion of each category that appears in the sample.

In our example, the first item of `ten_flips` is the simulated proportion of heads and the second item of `ten_flips` is the simulated proportion of tails.

```
[32]: simulated_proportion_heads = ten_flips.item(0)
      simulated_proportion_tails = ten_flips.item(1)

      print("In our simulation, " + str(simulated_proportion_heads) + " of flips were␣
       ↪heads and " \
            + str(simulated_proportion_tails) + " of flips were tails.")
```

```
In our simulation, 0.4 of flips were heads and 0.6 of flips were tails.
```

Question 1.7: To begin simulating, we should start by creating a representation of Emily's model to use for our simulation. This will be an array with two items in it. The first item should be the proportion of times a TT practictioner picks the correct hand, assuming that Emily's model was correct. The second item should be the proportion of times, under the same assumption, that the TT practitioner picks the incorrect hand. Assign `model_proportions` to this array.

After this, we can simulate 210 hand choices, as Emily evaluated in real life, and find a single statistic to summarize this instance of the simulation. Use the `sample_proportions` function and assign the **proportion of correct hand choices** (out of 210) to `simulation_proportion_correct`. Lastly, use your statistic function to assign `one_statistic` to the value of the statistic for this one simulation.

Hint: `sample_proportions` usage can be found on the Python Reference.

```
[33]: # This saves the random state of our code so that we can
      # generate the same numbers each time we run the code.
      # Please do not change this line.
      np.random.seed(16)

      # STEP 1: Make an array of 2 elements representing the proportions of Emily's␣
      ↪model (representing correct/incorrect hand picks)
      model_proportions = make_array(0.5, 0.5)
      # STEP 2: Use the sample_proportions functions to simulate 210 hand choices.␣
      ↪The first item in the array will represent proportion of correct hand picks
      simulation_proportion_correct = sample_proportions(210, model_proportions).
      ↪item(0)
      # STEP 3: Use the statistic function that you defined in question 1.5 to find␣
      ↪the value of the statistic for this simulation, assign it to one_statistic
      one_statistic = statistic(0.5, simulation_proportion_correct)
      one_statistic
```

[33]:
```
0.9523809523809545
```

[34]:
```
grader.check("q1_7")
```

[34]:
```
q1_7 results: All test cases passed!
```

Question 1.8: Let's now see what the distribution of statistics is actually like under Emily's model.

Part 1 Define the function `simulation_and_statistic` to take in the `model_proportions` array and the expected proportion of times a TT practitioner would guess a hand correctly under Emily's model. The function should simulate Emily running through the experiment 210 times and return the statistic of this one simulation.

Part 2 Using this function, assign `simulated_statistics` to an array of 1000 statistics that you calculated under the assumption that Emily's model was true.

[35]:
```
# PART 1: Define a function
# HINT: This code should follow the same pattern as the code you wrote in the␣
↪previous problem.

def simulation_and_statistic(model_proportions, expected_proportion_correct):
    '''Simulates 210 TT hand choices under Emily's model.
    Returns one statistic from the simulation.'''
    # STEP 1: Use the sample_proportions functions to simulate 210 proportions␣
↪from model_proportions. The first item in the array will represent the␣
↪proportion of correct hand picks
    simulated_proportion =  sample_proportions(210, model_proportions).item(0)
    # STEP 2: Use the statistic function with the correct function arguments to␣
↪find the value of the statistic for this simulation
    simulated_statistic = statistic(expected_proportion_correct,␣
↪simulated_proportion)
    # STEP 3: return the obtained statistic from the function
    return simulated_statistic
```

```
[36]: # PART 2: Use the simulation_and_statistics function to simulate 1000 times
num_repetitions = 1000

# STEP 1: Start from an empty array
simulated_statistics = make_array()

# STEP 2: Create a loop that runs for num_repetitions iterations
for i in range(num_repetitions):
# STEP 3: get a statistic for each iteration of the loop
    onestatistic = simulation_and_statistic(model_proportions,␣
 ↪expected_proportion_correct)
# STEP 4: append the statistic to the array of simulated statistics
    simulated_statistics = np.append(simulated_statistics, onestatistic)

simulated_statistics # This should output an array of 1000 statistics
```

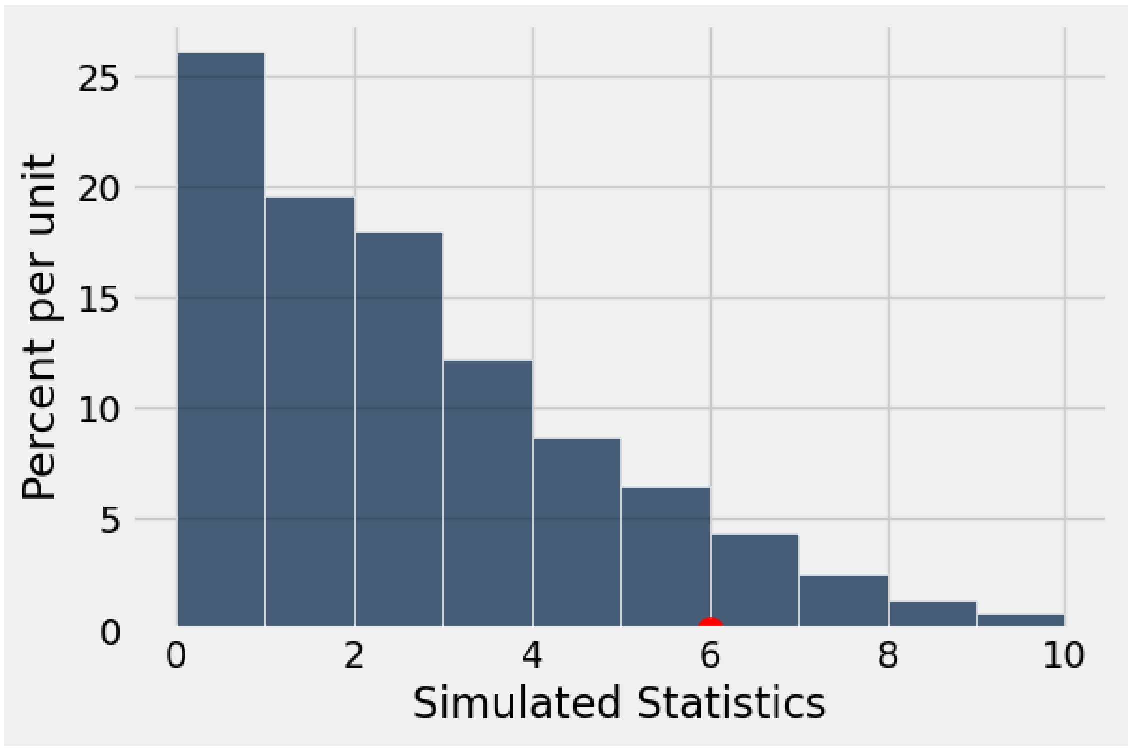
```
[36]: array([ 6.19047619,  2.85714286,  1.42857143,  5.23809524,
          6.19047619,  0.95238095,  0.47619048,  4.28571429,
          6.19047619,  3.33333333,  3.33333333,  3.80952381,
          1.9047619 ,  2.85714286,  1.9047619 ,  2.38095238,
          0.47619048,  3.80952381,  4.28571429,  0.47619048,
          6.66666667,  1.42857143,  4.28571429,  1.9047619 ,
          0.        ,  2.85714286,  1.42857143,  2.85714286,
          1.9047619 ,  7.14285714,  3.33333333,  2.85714286,
          7.14285714,  0.95238095,  0.95238095,  2.38095238,
          3.80952381,  0.47619048,  0.47619048,  5.71428571,
          0.47619048,  1.9047619 ,  4.76190476,  0.47619048,
          2.85714286,  1.42857143,  2.38095238,  1.42857143,
          0.95238095,  2.38095238,  3.80952381,  5.71428571,
          6.19047619,  2.38095238,  9.04761905,  3.33333333,
          2.38095238,  7.14285714,  4.28571429,  0.95238095,
          1.9047619 ,  0.47619048,  4.76190476,  3.80952381,
          2.85714286,  2.85714286,  0.47619048,  3.80952381,
          1.42857143,  1.9047619 ,  1.42857143,  0.47619048,
          4.28571429,  0.95238095,  0.47619048,  2.85714286,
          1.42857143,  0.        ,  0.47619048,  0.95238095,
          0.        ,  1.9047619 ,  3.80952381,  6.19047619,
          0.        ,  0.47619048,  0.        ,  2.38095238,
          1.42857143,  7.61904762,  2.85714286,  1.42857143,
          0.95238095,  0.95238095,  7.61904762,  6.19047619,
          2.38095238,  4.76190476,  0.47619048,  0.        ,
          3.33333333,  0.        ,  5.71428571,  3.80952381,
          2.85714286,  4.28571429,  4.28571429,  4.28571429,
          4.76190476,  1.9047619 ,  2.38095238,  2.38095238,
          4.28571429,   0.47619048,   2.85714286,   1.9047619 ,
          5.71428571,   3.33333333,   1.9047619 ,   1.42857143,
          2.85714286,   2.38095238,   1.42857143,   2.85714286,
          2.85714286,   4.28571429,   0.47619048,   7.14285714,
          4.76190476,   0.        ,   4.28571429,   3.80952381,
          3.80952381,   3.33333333,   6.66666667,   0.        ,
          6.19047619,   2.38095238,   2.85714286,   0.95238095,
          0.47619048,   1.42857143,   2.85714286,   1.9047619 ,
          0.        ,   1.42857143,   2.85714286,   0.        ,
          0.95238095,   2.38095238,   5.23809524,   2.38095238,
          0.47619048,   3.33333333,   0.95238095,   0.95238095,
          0.95238095,   1.9047619 ,   1.42857143,   0.47619048,
          1.42857143,   0.47619048,   0.        ,   1.9047619 ,
          1.42857143,   2.38095238,   0.        ,   2.85714286,
          1.9047619 ,   0.        ,   0.47619048,   1.9047619 ,
          3.33333333,   0.47619048,   3.80952381,   1.42857143,
          0.        ,   2.38095238,   3.80952381,   4.28571429,
          0.95238095,   2.38095238,   2.85714286,   4.28571429,
          3.80952381,   1.42857143,   6.66666667,   5.71428571,
          1.9047619 ,   2.38095238,   0.47619048,   0.95238095,
          0.47619048,   0.        ,   2.38095238,   2.38095238,
          5.71428571,   5.23809524,   5.23809524,   1.9047619 ,
          6.19047619,   3.33333333,   0.95238095,   3.33333333,
          2.85714286,   2.38095238,   2.38095238,   3.33333333,
          0.95238095,   1.42857143,   2.38095238,   0.        ,
          2.85714286,   1.42857143,   1.42857143,   1.42857143,
          6.19047619,   0.47619048,   4.76190476,   5.23809524,
          0.47619048,   5.23809524,   2.38095238,   0.95238095,
          5.71428571,   4.76190476,   3.33333333,   2.85714286,
          1.42857143,   0.        ,   3.80952381,   0.        ,
          0.47619048,   3.33333333,   0.47619048,   0.47619048,
          0.95238095,   3.80952381,   2.85714286,   0.95238095,
          2.38095238,   1.42857143,   3.80952381,   1.9047619 ,
          1.9047619 ,   2.85714286,   2.85714286,   4.76190476,
          3.80952381,   0.95238095,   4.28571429,   1.42857143,
          1.42857143,   1.9047619 ,   1.9047619 ,   0.47619048,
          2.38095238,   0.95238095,   0.95238095,   2.38095238,
          2.38095238,   2.38095238,   0.95238095,   0.47619048,
          0.47619048,   2.85714286,   1.9047619 ,   7.14285714,
          5.71428571,   6.66666667,   0.47619048,   0.95238095,
          2.38095238,   6.66666667,   3.33333333,   2.38095238,
          2.85714286,   4.28571429,   2.38095238,   3.33333333,
          1.42857143,   1.42857143,   2.85714286,   1.42857143,
          0.        ,   7.14285714,   0.95238095,   2.38095238,
          4.76190476,   1.9047619 ,   3.33333333,   0.        ,
          3.33333333,   4.28571429,   5.71428571,   3.33333333,
          8.0952381 ,   2.85714286,   5.71428571,   2.85714286,
          1.9047619 ,   2.85714286,   3.33333333,   0.95238095,
          1.9047619 ,   0.47619048,   0.95238095,   5.23809524,
          2.38095238,   1.9047619 ,   3.33333333,   2.38095238,
          3.33333333,   1.42857143,   2.38095238,   1.42857143,
          1.9047619 ,   5.71428571,   6.66666667,   3.80952381,
          0.47619048,   2.38095238,   5.71428571,   1.42857143,
          4.28571429,   0.47619048,   0.95238095,   6.66666667,
          0.        ,   2.85714286,   2.38095238,   1.42857143,
          3.80952381,   1.9047619 ,   3.33333333,   3.33333333,
          3.80952381,   2.85714286,   3.80952381,   1.9047619 ,
          0.47619048,   6.66666667,   1.42857143,   0.        ,
          0.47619048,   7.14285714,   2.85714286,   0.47619048,
          0.47619048,   1.42857143,   3.80952381,   1.9047619 ,
          1.42857143,   0.47619048,   7.61904762,   1.42857143,
          0.47619048,   2.38095238,   6.19047619,   1.9047619 ,
          5.23809524,   0.47619048,   0.        ,   2.38095238,
          1.42857143,   0.47619048,   0.        ,   0.        ,
          1.9047619 ,   3.80952381,   1.42857143,   0.95238095,
          1.9047619 ,   3.33333333,   0.95238095,   3.80952381,
          2.38095238,   1.9047619 ,   2.85714286,   4.76190476,
          3.33333333,   5.71428571,   1.42857143,   8.0952381 ,
          2.38095238,   1.9047619 ,   0.95238095,   4.76190476,
          0.95238095,   5.71428571,   0.95238095,   2.38095238,
          2.85714286,   3.80952381,   0.47619048,   1.42857143,
          4.28571429,   0.        ,   5.71428571,   1.9047619 ,
          0.        ,   3.80952381,   4.28571429,   5.71428571,
          5.23809524,   0.47619048,   5.71428571,   1.42857143,
          0.47619048,   3.33333333,   4.28571429,   0.95238095,
          3.33333333,   1.42857143,   2.85714286,   1.42857143,
          1.42857143,   1.9047619 ,   0.95238095,   5.23809524,
          3.33333333,   0.        ,   3.80952381,   0.95238095,
          1.9047619 ,   2.85714286,   2.85714286,   2.85714286,
          2.85714286,   0.95238095,   0.47619048,   6.19047619,
          2.38095238,   2.38095238,   0.95238095,   6.19047619,
          2.38095238,   0.47619048,   5.23809524,   2.38095238,
          1.9047619 ,   0.95238095,   3.33333333,   0.        ,
          4.28571429,   0.95238095,   2.38095238,   5.23809524,
          2.85714286,   1.42857143,   1.9047619 ,   0.47619048,
          0.95238095,   0.47619048,   0.47619048,   0.95238095,
          4.76190476,   1.42857143,   0.95238095,   0.47619048,
          5.23809524,   0.        ,   0.        ,   5.71428571,
          5.23809524,   2.38095238,   0.47619048,   0.47619048,
          0.95238095,   4.76190476,   0.47619048,   2.38095238,
          0.47619048,   6.66666667,   1.9047619 ,   7.14285714,
          3.33333333,   1.9047619 ,   4.76190476,   0.47619048,
          2.85714286,   1.9047619 ,   0.47619048,   2.85714286,
          1.42857143,   2.38095238,   0.95238095,   1.42857143,
          0.47619048,   0.95238095,   2.85714286,   0.95238095,
          0.47619048,   1.42857143,   2.85714286,   7.61904762,
          2.38095238,   8.0952381 ,   0.47619048,   1.9047619 ,
          1.9047619 ,   2.85714286,   2.85714286,   2.38095238,
          0.47619048,   0.        ,   2.38095238,   5.23809524,
          5.23809524,   0.95238095,   3.33333333,   4.76190476,
          2.85714286,   3.33333333,   2.85714286,   1.42857143,
          7.61904762,   2.38095238,   6.19047619,   4.76190476,
          0.95238095,   1.42857143,   0.95238095,   0.        ,
          2.38095238,   5.23809524,   8.57142857,   2.38095238,
          2.38095238,   1.42857143,   4.28571429,   0.95238095,
          1.42857143,   2.38095238,   3.80952381,   4.28571429,
          8.0952381 ,   0.47619048,   1.9047619 ,   4.76190476,
          1.9047619 ,   2.38095238,   0.95238095,   2.85714286,
          0.        ,   7.61904762,   5.23809524,   1.42857143,
          4.76190476,   1.9047619 ,   0.47619048,   6.19047619,
          6.19047619,   3.33333333,   5.71428571,   5.71428571,
          0.95238095,   3.33333333,   1.9047619 ,   0.95238095,
          2.85714286,   5.23809524,   6.66666667,   1.9047619 ,
          5.23809524,   1.9047619 ,   4.76190476,   8.0952381 ,
          0.95238095,   5.23809524,   1.42857143,   2.38095238,
          1.42857143,   2.38095238,   3.80952381,   0.95238095,
          1.42857143,   0.95238095,   3.80952381,   0.47619048,
          2.85714286,   0.47619048,   1.42857143,   0.47619048,
          2.38095238,   4.76190476,   0.47619048,   4.76190476,
          7.61904762,   1.9047619 ,   0.47619048,   7.61904762,
          5.23809524,   0.        ,   2.85714286,   4.76190476,
          4.28571429,   3.33333333,   1.9047619 ,   2.38095238,
          3.33333333,   2.38095238,   2.85714286,   1.42857143,
          0.95238095,   2.38095238,   1.42857143,   4.76190476,
          0.95238095,   1.42857143,   4.76190476,   2.85714286,
          0.95238095,   1.9047619 ,   1.9047619 ,   0.        ,
          0.95238095,   0.        ,   6.19047619,   6.19047619,
          2.38095238,   1.9047619 ,   8.0952381 ,   2.85714286,
          0.95238095,   0.        ,   2.85714286,   5.71428571,
          2.85714286,   2.38095238,   2.38095238,   3.33333333,
          1.42857143,   2.38095238,   1.42857143,   0.47619048,
          1.9047619 ,   3.80952381,   5.71428571,   0.        ,
          1.9047619 ,   5.23809524,   1.42857143,   4.76190476,
          0.47619048,   2.85714286,   0.47619048,   2.85714286,
          6.19047619,   2.85714286,   0.47619048,   3.80952381,
          4.28571429,   3.80952381,   0.95238095,   3.33333333,
          1.42857143,   0.95238095,   0.        ,   1.9047619 ,
          4.28571429,   1.9047619 ,   2.38095238,   0.95238095,
          1.42857143,   4.76190476,   0.95238095,   0.95238095,
          0.95238095,   2.85714286,   2.85714286,   3.80952381,
          0.95238095,   0.        ,   0.        ,   5.71428571,
          2.38095238,   0.95238095,   0.47619048,   4.28571429,
          1.9047619 ,   1.42857143,   3.80952381,   1.9047619 ,
          7.61904762,   3.33333333,   1.42857143,   1.9047619 ,
          3.80952381,   3.33333333,   3.80952381,   4.28571429,
          3.80952381,   9.04761905,   5.23809524,   2.85714286,
          5.23809524,   6.19047619,   3.80952381,   3.33333333,
          3.80952381,   1.9047619 ,   8.0952381 ,   2.85714286,
          6.66666667,   4.28571429,   0.95238095,   3.80952381,
          0.47619048,   1.42857143,   0.95238095,   3.33333333,
          0.95238095,   0.47619048,   0.95238095,   2.38095238,
          3.80952381,   2.38095238,   4.28571429,   6.66666667,
          0.95238095,   0.95238095,   0.95238095,   1.42857143,
          4.76190476,   2.38095238,   5.71428571,   8.57142857,
          2.85714286,   6.19047619,   0.47619048,   3.33333333,
          0.95238095,   0.95238095,   4.28571429,   2.38095238,
          4.28571429,   2.85714286,   4.28571429,   0.        ,
          7.14285714,   2.38095238,   2.85714286,   1.42857143,
          1.9047619 ,   3.33333333,   4.28571429,   1.9047619 ,
          0.        ,   1.9047619 ,   2.38095238,   4.28571429,
          1.9047619 ,   2.85714286,   2.85714286,   5.23809524,
          1.42857143,   2.38095238,   4.28571429,   0.        ,
          8.0952381 ,   0.47619048,   6.66666667,   1.9047619 ,
          0.47619048,   7.14285714,   2.38095238,   3.33333333,
          4.28571429,   2.38095238,   0.        ,   1.42857143,
          0.95238095,   3.33333333,   1.9047619 ,   1.9047619 ,
          7.61904762,   5.23809524,   0.95238095,   2.85714286,
          0.47619048,   7.14285714,   2.38095238,   5.71428571,
          3.33333333,   0.95238095,   3.33333333,   1.9047619 ,
          3.80952381,   4.76190476,   1.42857143,   1.42857143,
          1.42857143,   4.76190476,   0.95238095,   1.9047619 ,
          6.66666667,   1.9047619 ,   1.9047619 ,   3.33333333,
          1.9047619 ,   0.        ,   2.38095238,   2.38095238,
          1.42857143,   1.42857143,   1.9047619 ,   0.47619048,
          4.76190476,   6.19047619,   1.9047619 ,   0.47619048,
          3.33333333,   0.47619048,   4.76190476,   6.19047619,
          7.14285714,   0.        ,   0.95238095,   4.76190476,
          4.28571429,   7.61904762,   4.28571429,   3.80952381,
          2.85714286,   1.42857143,   5.23809524,   3.33333333,
          0.47619048,   0.47619048,   4.76190476,   3.80952381,
          1.42857143,   7.14285714,   0.47619048,   0.47619048,
          2.38095238,   3.80952381,   1.9047619 ,   0.        ,
          0.95238095,   5.23809524,   4.76190476,   2.85714286,
          1.42857143,   1.42857143,   4.28571429,   6.19047619,
          2.38095238,   0.95238095,   9.04761905,   1.42857143,
          2.38095238,   0.        ,   5.71428571,   5.23809524,
          2.38095238,   3.80952381,   9.04761905,   0.95238095,
          1.42857143,   1.9047619 ,   8.0952381 ,   0.47619048,
          2.85714286,  3.33333333,  4.76190476,  0.47619048,
          1.9047619 ,  2.38095238,  2.38095238,  0.47619048,
          3.80952381,  0.        ,  5.23809524,  2.38095238,
          2.38095238,  5.23809524,  4.28571429,  3.33333333,
          2.85714286,  0.95238095,  1.42857143,  3.33333333,
          1.42857143,  2.38095238,  3.33333333,  1.9047619 ,
          0.47619048,  6.19047619,  3.33333333, 10.        ,
          3.33333333,  0.47619048,  4.28571429,  8.57142857,
          1.9047619 ,  2.38095238,  2.85714286,  3.80952381,
          2.38095238,  1.42857143,  0.        ,  5.23809524,
          2.38095238,  0.95238095,  4.28571429,  4.28571429,
          3.33333333,  3.33333333,  0.        ,  5.23809524,
          6.19047619,  1.9047619 ,  1.42857143,  1.9047619 ,
          0.47619048,  1.42857143,  6.19047619,  2.38095238,
          5.23809524,  3.80952381,  4.76190476,  3.33333333,
          2.38095238,  3.80952381,  8.57142857,  0.47619048,
          5.23809524,  0.95238095,  1.9047619 ,  3.33333333,
          3.33333333,  4.76190476,  1.42857143,  4.28571429,
          3.80952381,  0.47619048,  3.80952381,  1.42857143,
          3.80952381,  1.9047619 ,  1.42857143,  2.85714286,
          1.9047619 ,  0.95238095,  3.80952381,  0.47619048,
          5.71428571,  3.33333333,  0.95238095,  1.9047619 ,
          0.47619048,  0.        ,  4.28571429,  0.95238095,
          1.9047619 ,  1.9047619 ,  1.42857143,  9.04761905,
          4.28571429,  4.76190476,  2.38095238,  0.95238095,
          3.80952381,  7.61904762,  0.95238095,  9.04761905,
          3.80952381,  1.9047619 ,  1.9047619 ,  5.71428571,
          1.9047619 ,  1.42857143,  0.95238095,  0.47619048,
          2.85714286,  1.9047619 ,  4.28571429,  4.28571429,
          0.95238095,  1.9047619 ,  6.19047619,  0.95238095,
          0.95238095,  0.        ,  2.38095238,  0.47619048,
          6.66666667,  6.66666667,  6.19047619,  0.95238095,
          0.95238095,  1.42857143,  5.71428571,  0.        ,
          1.42857143,  0.47619048,  1.9047619 ,  3.33333333])
```

[41]:
```
grader.check("q1_8")
```

[41]:
```
q1_8 results: All test cases passed!
```

Let's view the distribution of the simulated statistics under Emily's model, and visually compare where the observed statistic lies relative to the simulated statistics.

[42]:
```
t = Table().with_column('Simulated Statistics', c)
t.hist()
plt.scatter(observed_statistic, 0, color='red', s=100, zorder=2);
```



We can make a visual argument as to whether we believe the observed statistic is consistent with Emily's model. Here, since larger values of the test statistic suggest the alternative model (where the chance of guessing the correct hand is something other than 50%), we can formalize our analysis by finding what proportion of simulated statistics were as large or larger than our observed test statistic (the area at or to the right of the observed test statistic). If this area is small enough, we'll declare that the observed data are inconsistent with our simulated model. Here is the link to the section in the textbook.

Question 1.9: Calculate the proportion of simulated statistics greater than or equal to the observed statistic.

Hint: `np.count_nonzero` usage can be found here.

[74]:
```
# HINT: Use np.count_nonzero together with a boolean conditional statement to␣
↪find the number of simulated statistics
# that are greater or equal to the observed statistic
proportion_greater_or_equal = (np.count_nonzero(simulated_statistics >=␣
↪observed_statistic))/1000
proportion_greater_or_equal
```

[74]:
```
0.089
```

[75]:
```
grader.check("q1_9")
```

[75]:
```
q1_9 results: All test cases passed!
```

By convention, we often compare the proportion we just calculated to 0.05. If the proportion of simulated statistics greater than or equal to the observed statistic is sufficiently small (less than or equal to 0.05), then this is evidence against Emily's model. Conceptually, you may think of this as the case where less than 5% of simulated values are as far or farther away from what we had expected. If this is not the case, we don't have any reason to doubt Emily's model.

This should help you make your own conclusions about Emily Rosa's experiment.

Therapeutic touch fell out of use after this experiment, which was eventually accepted into one of the premier medical journals. TT practitioners hit back and accused Emily and her family of tampering with the results, while some claimed that Emily's bad spiritual mood towards therapeutic touch made it difficult to read her HEF. Whatever it may be, Emily's experiment is a classic example about how anyone, with the right resources, can test anything they want!

Question 1.10: Now, take some time to think to yourself and discuss with your peers:

1. Is the data more consistent with Emily' model (practioners were randomly guessing)? The data is more consistent with Emily's model(practioners were randomly guessing).
2. What does this mean in terms of Emily's experiment? Do the TT practitioners' answers follow an even chance model or is there something else at play? What this means in term's of Emily's experiment is this supports her random guessing hypotheses. The TT practioners' answers follow a random chance model.

Check in with your instructor or tutor if you are unsure about the answers.

1.2 2. Submission

Appa and Momo want to congratulate you on completing the lab!! Now you can relax like them!

Be sure to…

- run the tests and verify that they all pass,
- choose **Download as PDF** from the **File** menu
- submit the .pdf file on **canvas**.