hw06

April 25, 2022

Name: Allan Gongora

Section: 0131

1 Homework 6: Probability, Simulation, Estimation, and Assessing Models

Please complete this notebook by filling in the cells provided. Before you begin, execute the following cell to load the provided tests.

[1]:
```
pip install gofer-grader
```

```
Collecting gofer-grader
  Using cached gofer_grader-1.1.0-py3-none-any.whl (9.9 kB)
Requirement already satisfied: tornado in /opt/conda/lib/python3.7/site-packages
(from gofer-grader) (6.1)
Requirement already satisfied: pygments in /opt/conda/lib/python3.7/site-
packages (from gofer-grader) (2.11.2)
Requirement already satisfied: jinja2 in /opt/conda/lib/python3.7/site-packages
(from gofer-grader) (3.0.3)
Requirement already satisfied: MarkupSafe>=2.0 in /opt/conda/lib/python3.7/site-
packages (from jinja2->gofer-grader) (2.0.1)
Installing collected packages: gofer-grader
Successfully installed gofer-grader-1.1.0
Note: you may need to restart the kernel to use updated packages.
```

[2]:
```
# Don't change this cell; just run it.

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 gofer.ok import check
```

Recommended Reading: * Randomness * Sampling and Empirical Distributions * Testing Hypotheses 1) For all problems that you must write explanations and sentences for, you **must** provide your answer in the designated space. This can include: A) Sentence reponses to questions that ask for an explanation B) Numeric responses to multiple choice questions C) Programming code

2) Moreover, throughout this homework and all future ones, please be sure to not re-assign variables throughout the notebook! For example, if you use `max_temperature` in your answer to one question, do not reassign it later on. Otherwise, you will fail tests that you thought you were passing previously!

Once you're finished, select "Save and Checkpoint" in the File menu. Your name and course section number should be in the first and last cell of the assignment. Be sure you have run all cells with code and that the output from that is showing. Then click "Print Preview" in the File menu. Print a copy from there in pdf format. (This means you right click and choose print and choose "save as pdf" from your printer options.) You will need to submit the pdf in Canvas by the deadline.

The gopher grader output and/or output from your coding are essential to helping your instructor grade your work correctly and in a timely manner.

Files submitted that are missing the required output will lose some to all points so double check your pdf before submitting.

1.1 1. Probability

We will be testing some probability concepts that were introduced in lecture. For all of the following problems, we will introduce a problem statement and give you a proposed answer. Next, for each of the following questions, you must assign the provided variable to one of three integers. You are more than welcome to create more cells across this notebook to use for arithmetic operations, but be sure to assign the provided variable to 1, 2, or 3 in the end.

1. Assign the variable to 1 if you believe our proposed answer is too low.
2. Assign the variable to 2 if you believe our proposed answer is correct.
3. Assign the variable to 3 if you believe our proposed answer is too high.

Question 1.1 You roll a 6-sided die 10 times. What is the chance of getting 10 sixes?

Our proposed answer:

$$(\frac{1}{6})^{10}$$

Assign `ten_sixes` to either 1, 2, or 3 depending on if you think our answer is too low, correct, or too high.

```
[3]: ten_sixes = 2
     ten_sixes
```

```
[3]: 2
```

```
[4]: check('tests/q1_1.py')
```

```
[4]: <gofer.ok.OKTestsResult at 0x7fd1f01dfdd0>
```

Question 1.2 Take the same problem set-up as before, rolling a fair die 10 times. What is the chance that every roll is less than or equal to 5?

Our proposed answer:

$$1 - (\frac{1}{6})^{10}$$

Assign `five_or_less` to either 1, 2, or 3.

```
[5]: five_or_less = 3
     five_or_less
```

```
[5]: 3
```

```
[6]: check('tests/q1_2.py')
```

```
[6]: <gofer.ok.OKTestsResult at 0x7fd1f0182610>
```

Question 1.3 Assume we are picking a lottery ticket. We must choose three distinct numbers from 1 to 100 and write them on a ticket. Next, someone picks three numbers one-by-one, each time without putting the previous number back in. We win if our numbers are all called.

If we decide to play the game and pick our numbers as 12, 14, and 89, what is the chance that we win?

Our proposed answer:

$$(\frac{3}{100})^3$$

Assign `lottery` to either 1, 2, or 3.

```
[7]: ((1 - 97/100) * (1 - 97/99) * (1 - 97/98)) < (3/100)**3
```

```
[7]: True
```

```
[8]: lottery = 3
```

```
[9]: check('tests/q1_3.py')
```

```
[9]: <gofer.ok.OKTestsResult at 0x7fd1f0182350>
```

Question 1.4 Assume we have two lists, List A and List B. List A contains the numbers [10,20,30], while List B contains the numbers [10,20,30,40]. We choose one number from List A randomly and one number from List B randomly. What is the chance that the number we drew from List A is larger than the number we drew from List B?

Our proposed solution:

1/4

Assign `list_chances` to either 1, 2, or 3.

```
[10]: # a = [10, 20, 30]
      # b = [10, 20, 30, 40]
      # chance from_a > from_b
      # 0 + 1/4 + 2/4 = 3/4
```

```
[11]: list_chances = 1
```

```
[12]: check('tests/q1_4.py')
```

```
[12]: <gofer.ok.OKTestsResult at 0x7fd1ab75dd90>
```

1.2 2. Monkeys Typing Shakespeare

(…or at least the string "datascience") A monkey is banging repeatedly on the keys of a typewriter. Each time, the monkey is equally likely to hit any of the 26 lowercase letters of the English alphabet, regardless of what it has hit before. There are no other keys on the keyboard.

Question 2.1 Suppose the monkey hits the keyboard 11 times. Compute the chance that the monkey types the sequence `datascience`. (Call this `datascience_chance`.) Use Algebra and type in an arithmetic equation that Python can evaluate.

```
[13]: datascience_chance = (1/26)**11
      datascience_chance
```

```
[13]: 2.7245398995795435e-16
```

```
[14]: check('tests/q2_1.py')
```

```
[14]: <gofer.ok.OKTestsResult at 0x7fd1ab785ed0>
```

Question 2.2 Write a function called `simulate_key_strike`. It should take **no arguments**, and it should return a random one-character string that is equally likely to be any of the 26 lower-case English letters.

```
[15]: # We have provided the code below to compute a list called letters,
      # containing all the lower-case English letters.  Print it if you
      # want to verify what it contains.
      import string
      letters = list(string.ascii_lowercase)

      def simulate_key_strike():
          """Simulates one random key strike."""
          return np.random.choice(letters)
```

```
# An example call to your function:
simulate_key_strike()
```

```
[15]: 'r'
```

```
[16]: check('tests/q2_2.py')
```

```
[16]: <gofer.ok.OKTestsResult at 0x7fd1f01ca9d0>
```

Question 2.3 Write a function called `simulate_several_key_strikes`. It should take one argument: an integer specifying the number of key strikes to simulate. It should return a string containing that many characters (each one obtained from simulating a key strike by the monkey).

Hint: If you make a list or array of the simulated key strikes, you can convert that to a string by calling `"".join(key_strikes_array)` (if your array is called `key_strikes_array`).

```
[17]: def simulate_several_key_strikes(num_strikes):
          # Fill in this function.  Our solution used several lines
          # of code.
          to_return = ""
          for _ in range(num_strikes):
              to_return += np.random.choice(letters)
          return to_return

      # An example call to your function:
      simulate_several_key_strikes(11)
```

```
[17]: 'qdyqshkxsnl'
```

```
[18]: check('tests/q2_3.py')
```

```
[18]: <gofer.ok.OKTestsResult at 0x7fd1c155aad0>
```

Question 2.4 Use `simulate_several_key_strikes` 1000 times, each time simulating the monkey striking 11 keys. Compute the proportion of times the monkey types `"datascience"`, calling that proportion `datascience_proportion`.

```
[19]: # Your solution may take more than one line.
      trials = 1000
      datascience_proportion = np.array([simulate_several_key_strikes(11) for _ in␣
       ↪range(trials)])
      datascience_proportion = len(datascience_proportion[datascience_proportion ==␣
       ↪"datascience"]) / trials
      datascience_proportion
```

```
[19]: 0.0
```

2 THIS EXPIREMENT DID NOT CONVERGE IN 10 MINUTES

```
[20]: # iters = 0
      # while datascience_proportion == 0:
      #     datascience_proportion = np.array([simulate_several_key_strikes(11) for _␣
       ↪in range(trials)])
      #     datascience_proportion =␣
       ↪len(datascience_proportion[datascience_proportion == "datascience"]) / trials
      #     iters += 1
      # print(f"Took {times * trials} calls of the function to output atleast one␣
       ↪`datascience`")
```

```
[21]: check('tests/q2_4.py')
```

```
[21]: <gofer.ok.OKTestsResult at 0x7fd1f01d3dd0>
```

Question 2.5 Check the value your simulation computed for `datascience_proportion`. Is your simulation a good way to estimate the chance that the monkey types `"datascience"` in 11 strikes (the answer to Question 2.1)? Why or why not?

Probably? In the real world i doubt a monkey would hit a single key on a keyboard. They would probably smash the keyboard and therefore hit surrounding keys. This thenn changes the probability of a key beinng pressed. However if we imagine we can force a fair chance between any key being selected, then yes this is a good simulation.

Question 2.6 Compute the chance that the monkey types the letter `"e"` at least once in the 11 strikes. Call it `e_chance`. Use Algebra and type in an arithmetic equation that Python can evaluate.

```
[22]: e_chance =  1 - ((25/26)**11) # 1 - minus the chance of never choosing e
      e_chance
```

```
[22]: 0.35041906843673165
```

```
[23]: check('tests/q2_6.py')
```

```
[23]: <gofer.ok.OKTestsResult at 0x7fd1ab732510>
```

Question 2.7 Do you think that a computer simulation is more or less effective to estimate `e_chance` compared to when we tried to estimate `datascience_chance` this way? Why or why not? (You don't need to write a simulation, but it is an interesting exercise.)

Equally as effective. We are just testing randomness and in the same way just for different scenarios i dont see how the effectiveness would have changed.

2.1 3. Sampling Basketball Players

This exercise uses salary data and game statistics for basketball players from the 2014-2015 NBA season. The data was collected from Basketball-Reference and Spotrac.

Run the next cell to load the two datasets.

```
[24]: player_data = Table.read_table('player_data.csv')
      salary_data = Table.read_table('salary_data.csv')
      player_data.show(3)
      salary_data.show(3)
```

```
<IPython.core.display.HTML object>

<IPython.core.display.HTML object>
```

Question 3.1 We would like to relate players' game statistics to their salaries. Compute a table called `full_data` that includes one row for each player who is listed in both `player_data` and `salary_data`. It should include all the columns from `player_data` and `salary_data`, except the `"PlayerName"` column.

```
[25]: full_data = player_data.join("Name", salary_data, "PlayerName")
      full_data
```

```
[25]: Name             | Age  | Team | Games | Rebounds | Assists | Steals | Blocks |
      Turnovers | Points | Salary
      A.J. Price       | 28   | TOT  | 26    | 32       | 46      | 7      | 0      |
      14        | 133    | 62552
      Aaron Brooks     | 30   | CHI  | 82    | 166      | 261     | 54     | 15     |
      157       | 954    | 1145685
      Aaron Gordon     | 19   | ORL  | 47    | 169      | 33      | 21     | 22     |
      38        | 243    | 3992040
      Adreian Payne    | 23   | TOT  | 32    | 162      | 30      | 19     | 9      |
      44        | 213    | 1855320
      Al Horford       | 28   | ATL  | 76    | 544      | 244     | 68     | 98     |
      100       | 1156   | 12000000
      Al Jefferson     | 30   | CHO  | 65    | 548      | 113     | 47     | 84     |
      68        | 1082   | 13666667
      Al-Farouq Aminu  | 24   | DAL  | 74    | 342      | 59      | 70     | 62     |
      55        | 412    | 1100602
      Alan Anderson    | 32   | BRK  | 74    | 204      | 83      | 56     | 5      |
      60        | 545    | 1276061
      Alec Burks       | 23   | UTA  | 27    | 114      | 82      | 17     | 5      |
      52        | 374    | 3034356
      Alex Kirk        | 23   | CLE  | 5     | 1        | 1       | 0      | 0      | 0
      | 4       | 507336
      … (482 rows omitted)
```

```
[26]: check('tests/q3_1.py')
```

```
[26]: <gofer.ok.OKTestsResult at 0x7fd1ab75df10>
```

Basketball team managers would like to hire players who perform well but don't command high salaries. From this perspective, a very crude measure of a player's *value* to his/her team is the number of points the player scored in a season for every **$1000 of salary** (*Note*: the `Salary` column is in dollars, not thousands of dollars). For example, Al Horford scored 1156 points and has a salary of **$12 million.** This is equivalent to 12,000 thousands of dollars, so his value is $\frac{1156}{12000}$.

Question 3.2 Create a table called `full_data_with_value` that's a copy of `full_data`, with an extra column called `"Value"` containing each player's value (according to our crude measure). Then, make a histogram of players' values. **Specify bins that make the histogram informative, and don't forget your units!** Remember that `hist()` takes in an optional third argument that allows you to specify the units!

Hint: Informative histograms contain a majority of the data and **exclude outliers**.

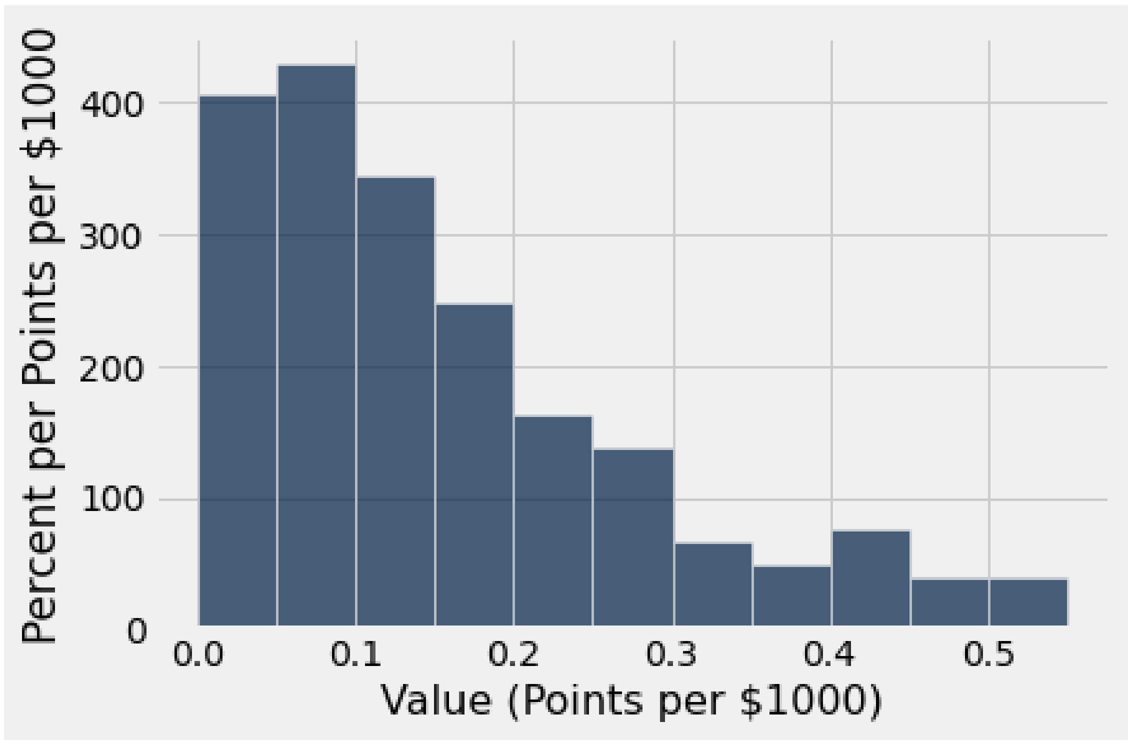
[27]:
```
full_data_with_value = full_data.with_column("Value", full_data["Points"] /␣
  ↪(full_data["Salary"] / 1000))
```

[37]:
```
val = full_data_with_value["Value"]
q1 = np.quantile(val, .25)
q3 = np.quantile(val, .75)
iqr = q3 - q1
```

[40]:
```
x = full_data_with_value.where("Value", are.above_or_equal_to(q1 - (1.5 * iqr)))
x = x.where("Value", are.below_or_equal_to(q3 + (1.5 * iqr)))
```

[52]:
```
meaningful_bins = np.arange(x["Value"].min(), round(x["Value"].max() + .05, 2),␣
  ↪.05)
```

[57]:
```
full_data_with_value.hist("Value", unit="Points per $1000",␣
  ↪bins=meaningful_bins)
```



Now suppose we weren't able to find out every player's salary (perhaps it was too costly to interview each player). Instead, we have gathered a *simple random sample* of 100 players' salaries. The cell below loads those data.

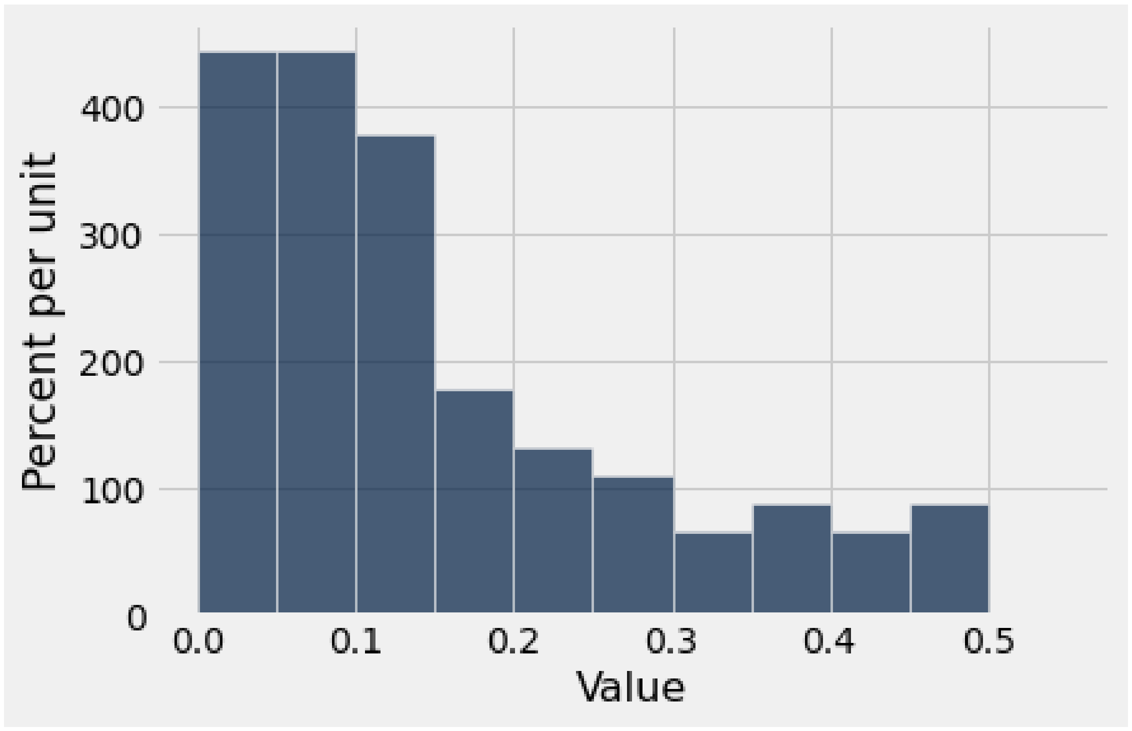
[55]:
```
sample_salary_data = Table.read_table("sample_salary_data.csv")
sample_salary_data.show(3)
```

```
<IPython.core.display.HTML object>
```

Question 3.3 Make a histogram of the values of the players in `sample_salary_data`, using the same method for measuring value we used in Question 3.2. **Use the same bins, too.**

Hint: This will take several steps.

[56]:
```
complete_sample = sample_salary_data.join("PlayerName", full_data_with_value,␣
  ↪"Name")
complete_sample.hist("Value", bins=meaningful_bins)
```



Now let us summarize what we have seen. To guide you, we have written most of the summary already.

Question 3.4 Complete the statements below by filling in the [SQUARE BRACKETS].

Hint 1: For a refresher on distribution types, check out Section 10.1

Hint 2: The `hist()` table method ignores data points outside the range of its bins, but you may ignore this fact and calculate the areas of the bars using what you know about histograms from lecture.

The plot in Question 3.2 displayed a(n) Empirical distribution of the population of 492 players. The areas of the bars in the plot sum to 1.

The plot in Question 3.3 displayed a(n) sample distribution of the sample of 100 players. The areas of the bars in the plot sum to 1.

Question 3.5 For which range of values does the plot in Question 3.3 better depict the distribution of the **population's player values**: 0 to 0.5, or above 0.5? Explain your answer.

0 - .5 because outliers

2.2 4. Earthquakes

The next cell loads a table containing information about **every earthquake with a magnitude above 4.5** in 2017, compiled by the US Geological Survey.

[60]:
```
earthquakes = Table().read_table('earthquakes_2017.csv').select(['time', 'mag',␣
 ↪'place'])
earthquakes
```

[60]:
```
time                     | mag  | place
2017-12-31T23:48:50.980Z | 4.8  | 30km SSE of Pagan, Northern Mariana Islands
2017-12-31T20:59:02.500Z | 5.1  | Southern East Pacific Rise
2017-12-31T20:27:49.450Z | 5.2  | Chagos Archipelago region
2017-12-31T19:42:41.250Z | 4.6  | 18km NE of Hasaki, Japan
2017-12-31T16:02:59.920Z | 4.5  | Western Xizang
2017-12-31T15:50:22.510Z | 4.5  | 156km SSE of Longyearbyen, Svalbard and Jan
Mayen
2017-12-31T14:53:32.590Z | 5.1  | 41km S of Daliao, Philippines
2017-12-31T14:51:58.200Z | 5.1  | 132km SSW of Lata, Solomon Islands
2017-12-31T12:24:13.150Z | 4.6  | 79km SSW of Hirara, Japan
2017-12-31T04:02:18.500Z | 4.8  | 10km W of Korini, Greece
… (6350 rows omitted)
```

There are several earthquakes that occurred in 2017 that we're interested in, and generally, we won't have access to this large population. Instead, if we sample correctly, we can take a small subsample of earthquakes in this year to get an idea about the distribution of magnitudes throughout the year!

Question 4.1 In the following lines of code, we take two different samples from the earthquake table, and calculate the mean of the magnitudes of these earthquakes. Are these samples representative of the population of earthquakes in the original table (that is, should we expect the mean to be close to the population mean)?

Hint: Consider the ordering of the `earthquakes` table.

[65]:
```
sample1 = earthquakes.sort('mag', descending = True).take(np.arange(100))
sample1_magnitude_mean = np.mean(sample1.column('mag'))
sample2 = earthquakes.take(np.arange(100))
sample2_magnitude_mean = np.mean(sample2.column('mag'))
[sample1_magnitude_mean, sample2_magnitude_mean]
```

[65]:
```
[6.422999999999999, 4.7749999999999995]
```

No. sample 1 is a the top 100. sample 2 is a convinience sample.

Question 4.2 Write code producing a sample that represents the population of size 500. Then, take the mean of the magnitudes of the earthquakes in this sample. Assign these to `representative_sample` and `representative_mean`, respectively.

Hint: In class, what sort of samples can properly represent the population?

[70]:
```
representative_sample = earthquakes.sample(500)
representative_mean = representative_sample["mag"].mean()
representative_mean
```

[70]:
```
4.7736
```

```
[71]: check('tests/q4_2.py')
```

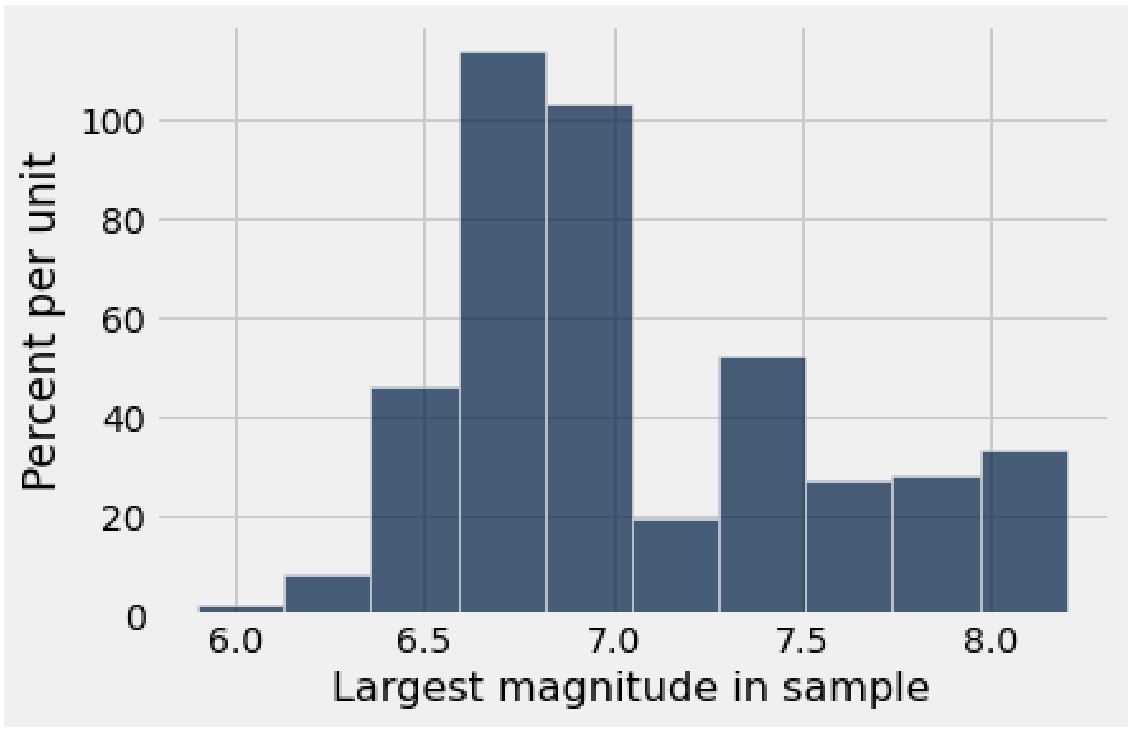
```
[71]: <gofer.ok.OKTestsResult at 0x7fd1a64c7d10>
```

Question 4.3 Suppose we want to figure out what the biggest magnitude earthquake was in 2017, but we are tasked with doing this only with a sample of 500 from the earthquakes table.

To determine whether trying to find the biggest magnitude from a sample is a plausible idea, write code that simulates the maximum of a random sample of size 500 from the `earthquakes` table 5000 times. Assign your array of maximums to `maximums`.

```
[81]: maximums = []
      for i in np.arange(5000):
          maximums.append(earthquakes.sample(500)["mag"].max())
```

```
[82]: #Histogram of your maximums
      Table().with_column('Largest magnitude in sample', maximums).hist('Largest␣
       ↪magnitude in sample')
```



```
[74]: check('tests/q4_3.py')
```

```
[74]: <gofer.ok.OKTestsResult at 0x7fd1a9e2d890>
```

Question 4.4 We want to see if a random sample of size 500 is likely to help determine the largest magnitude earthquake in the population. To help determine this, find the magnitude of the (actual) strongest earthquake in 2017.

```
[79]: strongest_earthquake_magnitude = earthquakes["mag"].max()
      strongest_earthquake_magnitude
```

```
[79]: 8.2
```

```
[80]: check('tests/q4_4.py')
```

```
[80]: <gofer.ok.OKTestsResult at 0x7fd1a6628e10>
```

Question 4.5 Explain whether you believe you can accurately use a sample size of 500 to determine the maximum. What is a specific con of using the maximum as your estimator? Use the histogram above to help answer.

No. 8.2 is not common in the histogram

2.3 5. Assessing Gary's Models

Games with Gary Our friend Gary comes over and asks us to play a game with him. The game works like this:

> We will flip a fair coin 10 times, and if the number of heads is greater than or equal to 5, we win!
>
> Otherwise, Gary wins.

We play the game once and we lose, observing 1 head. We are angry and accuse Gary of cheating! Gary is adamant, however, that the coin is fair.

Gary's model claims that there is an equal chance of getting heads or tails, but we do not believe him. We believe that the coin is clearly rigged, with heads being less likely than tails.

Question 5.1 Assign `coin_model_probabilities` to a two-item array containing the chance of heads as the first element and the chance of tails as the second element under Gary's model. Make sure your values are between 0 and 1.

```
[85]: coin_model_probabilities = np.array([.5, .5])
      coin_model_probabilities
```

```
[85]: array([0.5, 0.5])
```

```
[86]: check('tests/q5_1.py')
```

```
[86]: <gofer.ok.OKTestsResult at 0x7fd1a9188c50>
```

Question 5.2 We believe Gary's model is incorrect. In particular, we believe there is a smaller chance of heads. Which of the following statistics can we use during our simulation to test between the model and our alternative? Assign `statistic_choice` to the correct answer.

1. The distance (absolute value) between the actual number of heads in 10 flips and the expected number of heads in 10 flips (5).

2. The expected number of heads in 10 flips.
3. The actual number of heads we get in 10 flips.

[105]:
```
statistic_choice = 2
statistic_choice
```

[105]:
```
2
```

Question 5.3 Define the function `coin_simulation_and_statistic`, which, given a sample size and an array of model proportions (like the one you created in Question 5.1), returns the number of heads in one simulation of flipping the coin under the model specified in `model_proportions`.

Hint: Think about how you can use the function `sample_proportions`.

[103]:
```
def coin_simulation_and_statistic(sample_size, model_proportions):
    return sample_proportions(sample_size, model_proportions)[0] * sample_size

coin_simulation_and_statistic(10, coin_model_probabilities)
```

[103]:
```
3.0
```

[104]:
```
check('tests/q5_3.py')
```

[104]:
```
<gofer.ok.OKTestsResult at 0x7fd1a6d85810>
```

Question 5.4 Use your function from above to simulate the flipping of 10 coins 5000 times under the proportions that you specified in Question 5.1. Keep track of all of your statistics in `coin_statistics`.

[106]:
```
coin_statistics = []
repetitions = 5000

for _ in range(repetitions):
    coin_statistics.append(coin_simulation_and_statistic(10,␣
 ↪coin_model_probabilities))

coin_statistics
```

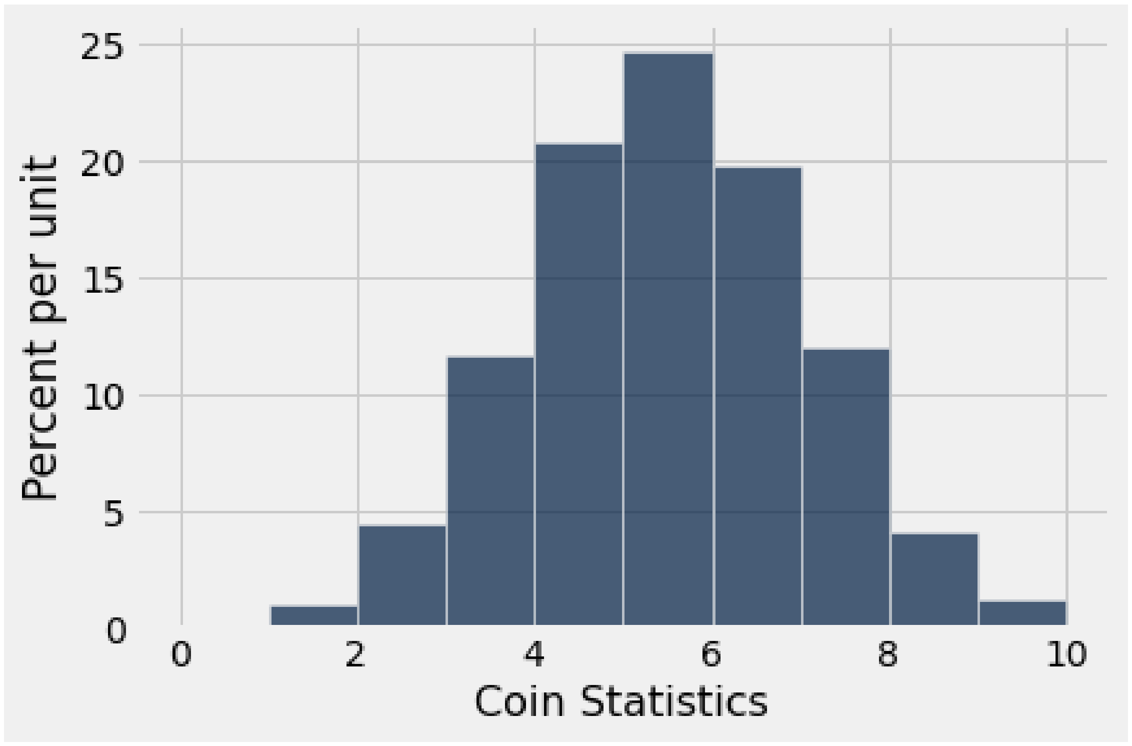
[106]:
```
[7.0,
 5.0,
 4.0,
 7.0,
 3.0,
 4.0,
 5.0,
 5.0,
 6.0,
 4.0,
 4.0,
4.0,
6.0,
4.0,
7.0,
6.0,
8.0,
5.0,
6.0,
5.0,
4.0,
8.0,
3.0,
3.0,
4.0,
4.0,
5.0,
6.0,
7.0,
5.0,
5.0,
6.0,
2.0,
5.0,
6.0,
5.0,
7.0,
8.0,
3.0,
3.0,
7.0,
6.0,
5.0,
4.0,
5.0,
4.0,
5.0,
7.0,
6.0,
3.0,
4.0,
3.0,
4.0,
5.0,
5.0,
6.0,
3.0,
5.0,
6.0,
3.0,
6.0,
6.0,
4.0,
7.0,
5.0,
2.0,
6.0,
5.0,
5.0,
5.0,
5.0,
5.0,
8.0,
7.0,
7.0,
6.0,
6.0,
3.0,
8.0,
5.0,
6.0,
7.0,
7.0,
4.0,
6.0,
6.0,
4.0,
5.0,
3.0,
4.0,
7.0,
3.0,
5.0,
4.0,
4.0,
7.0,
7.0,
5.0,
6.0,
1.0,
5.0,
6.0,
6.0,
5.0,
5.0,
6.0,
7.0,
5.0,
7.0,
8.0,
2.0,
4.0,
5.0,
5.0,
6.0,
8.0,
6.0,
4.0,
6.0,
6.0,
9.0,
6.0,
5.0,
6.0,
5.0,
7.0,
9.0,
4.0,
4.0,
6.0,
7.0,
5.0,
4.0,
6.0,
4.0,
6.0,
4.0,
5.0,
5.0,
6.0,
7.0,
5.0,
7.0,
5.0,
5.0,
5.0,
7.0,
7.0,
4.0,
5.0,
5.0,
4.0,
9.0,
7.0,
4.0,
5.0,
6.0,
6.0,
5.0,
3.0,
5.0,
3.0,
4.0,
4.0,
7.0,
5.0,
4.0,
6.0,
4.0,
5.0,
6.0,
6.0,
3.0,
4.0,
4.0,
5.0,
7.0,
4.0,
6.0,
6.0,
5.0,
8.0,
7.0,
6.0,
5.0,
4.0,
4.0,
5.0,
6.0,
9.0,
4.0,
6.0,
5.0,
6.0,
5.0,
6.0,
7.0,
5.0,
5.0,
7.0,
2.0,
5.0,
3.0,
6.0,
6.0,
4.0,
2.0,
5.0,
4.0,
5.0,
8.0,
6.0,
8.0,
9.0,
3.0,
6.0,
6.0,
5.0,
6.0,
4.0,
4.0,
7.0,
3.0,
6.0,
0.0,
3.0,
6.0,
4.0,
6.0,
4.0,
6.0,
5.0,
6.0,
5.0,
6.0,
7.0,
8.0,
4.0,
4.0,
10.0,
4.0,
7.0,
8.0,
6.0,
3.0,
4.0,
7.0,
6.0,
6.0,
3.0,
3.0,
6.0,
6.0,
6.0,
8.0,
3.0,
5.0,
5.0,
3.0,
5.0,
3.0,
5.0,
6.0,
6.0,
2.0,
5.0,
4.0,
4.0,
5.0,
5.0,
5.0,
6.0,
5.0,
5.0,
3.0,
6.0,
5.0,
5.0,
6.0,
3.0,
6.0,
5.0,
7.0,
4.0,
7.0,
2.0,
6.0,
5.0,
5.0,
6.0,
6.0,
4.0,
8.0,
3.0,
3.0,
5.0,
5.0,
4.0,
4.0,
6.0,
4.0,
5.0,
6.0,
5.0,
6.0,
3.0,
4.0,
3.0,
4.0,
2.0,
5.0,
7.0,
5.0,
3.0,
5.0,
1.0,
6.0,
5.0,
8.0,
6.0,
4.0,
4.0,
4.0,
6.0,
4.0,
7.0,
4.0,
4.0,
1.0,
6.0,
5.0,
6.0,
4.0,
3.0,
5.0,
7.0,
6.0,
6.0,
8.0,
4.0,
8.0,
5.0,
4.0,
8.0,
2.0,
4.0,
7.0,
4.0,
6.0,
4.0,
5.0,
3.0,
6.0,
6.0,
7.0,
5.0,
4.0,
5.0,
4.0,
3.0,
5.0,
5.0,
7.0,
7.0,
2.0,
7.0,
2.0,
6.0,
6.0,
4.0,
7.0,
8.0,
7.0,
4.0,
4.0,
9.0,
2.0,
7.0,
4.0,
5.0,
6.0,
5.0,
4.0,
5.0,
5.0,
5.0,
6.0,
6.0,
6.0,
2.0,
5.0,
5.0,
7.0,
6.0,
5.0,
7.0,
5.0,
6.0,
3.0,
8.0,
7.0,
4.0,
5.0,
6.0,
4.0,
7.0,
2.0,
2.0,
6.0,
4.0,
5.0,
6.0,
6.0,
8.0,
7.0,
6.0,
2.0,
7.0,
5.0,
5.0,
4.0,
3.0,
5.0,
7.0,
4.0,
6.0,
4.0,
6.0,
6.0,
4.0,
7.0,
4.0,
6.0,
5.0,
8.0,
4.0,
7.0,
5.0,
5.0,
3.0,
4.0,
5.0,
5.0,
4.0,
5.0,
3.0,
4.0,
4.0,
3.0,
6.0,
5.0,
5.0,
6.0,
6.0,
4.0,
2.0,
4.0,
8.0,
5.0,
4.0,
5.0,
4.0,
6.0,
3.0,
5.0,
6.0,
3.0,
7.0,
4.0,
6.0,
6.0,
4.0,
9.0,
3.0,
3.0,
7.0,
3.0,
4.0,
3.0,
3.0,
3.0,
1.0,
7.0,
4.0,
3.0,
3.0,
3.0,
7.0,
3.0,
6.0,
5.0,
5.0,
3.0,
4.0,
4.0,
8.0,
3.0,
7.0,
6.0,
3.0,
3.0,
6.0,
5.0,
4.0,
5.0,
7.0,
9.0,
6.0,
5.0,
6.0,
5.0,
7.0,
4.0,
9.0,
7.0,
2.0,
7.0,
3.0,
3.0,
3.0,
5.0,
4.0,
1.0,
3.0,
4.0,
7.0,
4.0,
5.0,
6.0,
6.0,
5.0,
3.0,
6.0,
7.0,
7.0,
5.0,
5.0,
8.0,
4.0,
6.0,
5.0,
2.0,
8.0,
4.0,
3.0,
8.0,
6.0,
7.0,
3.0,
4.0,
7.0,
4.0,
4.0,
4.0,
4.0,
7.0,
3.0,
5.0,
6.0,
6.0,
4.0,
5.0,
4.0,
6.0,
1.0,
4.0,
4.0,
7.0,
5.0,
6.0,
8.0,
4.0,
2.0,
2.0,
6.0,
5.0,
4.0,
5.0,
4.0,
5.0,
5.0,
4.0,
4.0,
4.0,
6.0,
5.0,
8.0,
5.0,
4.0,
5.0,
2.0,
2.0,
6.0,
4.0,
4.0,
2.0,
2.0,
4.0,
0.0,
6.0,
5.0,
5.0,
6.0,
6.0,
5.0,
3.0,
6.0,
3.0,
7.0,
5.0,
7.0,
3.0,
5.0,
3.0,
5.0,
5.0,
6.0,
4.0,
2.0,
4.0,
3.0,
6.0,
7.0,
5.0,
4.0,
4.0,
4.0,
4.0,
3.0,
4.0,
5.0,
4.0,
2.0,
5.0,
7.0,
5.0,
3.0,
6.0,
3.0,
7.0,
4.0,
5.0,
4.0,
3.0,
4.0,
7.0,
4.0,
4.0,
4.0,
4.0,
4.0,
5.0,
5.0,
5.0,
5.0,
2.0,
4.0,
5.0,
5.0,
5.0,
6.0,
6.0,
4.0,
5.0,
4.0,
6.0,
5.0,
3.0,
6.0,
5.0,
6.0,
6.0,
8.0,
5.0,
5.0,
5.0,
5.0,
8.0,
6.0,
7.0,
3.0,
1.0,
5.0,
3.0,
8.0,
4.0,
5.0,
5.0,
7.0,
6.0,
6.0,
1.0,
4.0,
7.0,
3.0,
4.0,
4.0,
3.0,
5.0,
6.0,
4.0,
4.0,
5.0,
4.0,
3.0,
5.0,
4.0,
3.0,
5.0,
5.0,
4.0,
5.0,
7.0,
5.0,
5.0,
7.0,
4.0,
8.0,
6.0,
7.0,
4.0,
4.0,
4.0,
4.0,
6.0,
5.0,
7.0,
6.0,
5.0,
4.0,
2.0,
5.0,
3.0,
5.0,
5.0,
9.0,
5.0,
9.0,
4.0,
5.0,
3.0,
5.0,
4.0,
4.0,
4.0,
7.0,
7.0,
4.0,
6.0,
6.0,
8.0,
5.0,
6.0,
7.0,
7.0,
6.0,
8.0,
6.0,
4.0,
4.0,
3.0,
4.0,
5.0,
5.0,
5.0,
5.0,
3.0,
7.0,
3.0,
5.0,
6.0,
5.0,
5.0,
4.0,
7.0,
5.0,
6.0,
5.0,
6.0,
4.0,
5.0,
6.0,
5.0,
6.0,
5.0,
6.0,
5.0,
4.0,
8.0,
5.0,
8.0,
8.0,
8.0,
7.0,
6.0,
6.0,
7.0,
6.0,
4.0,
1.0,
3.0,
4.0,
6.0,
5.0,
7.0,
6.0,
4.0,
3.0,
4.0,
5.0,
7.0,
5.0,
4.0,
7.0,
7.0,
2.0,
7.0,
8.0,
5.0,
2.0,
4.0,
5.0,
3.0,
5.0,
6.0,
9.0,
6.0,
3.0,
8.0,
5.0,
9.0,
7.0,
2.0,
4.0,
6.0,
4.0,
7.0,
5.0,
5.0,
5.0,
6.0,
7.0,
8.0,
8.0,
6.0,
4.0,
7.0,
7.0,
7.0,
7.0,
6.0,
3.0,
5.0,
8.0,
3.0,
6.0,
4.0,
4.0,
6.0,
5.0,
5.0,
4.0,
3.0,
7.0,
6.0,
5.0,
4.0,
2.0,
6.0,
5.0,
4.0,
4.0,
7.0,
4.0,
5.0,
6.0,
5.0,
5.0,
8.0,
5.0,
4.0,
2.0,
4.0,
5.0,
4.0,
4.0,
5.0,
6.0,
4.0,
7.0,
3.0,
8.0,
4.0,
4.0,
4.0,
2.0,
7.0,
4.0,
6.0,
5.0,
3.0,
5.0,
7.0,
4.0,
6.0,
4.0,
8.0,
5.0,
6.0,
8.0,
5.0,
7.0,
4.0,
8.0,
6.0,
6.0,
5.0,
6.0,
7.0,
6.0,
3.0,
8.0,
5.0,
6.0,
8.0,
3.0,
5.0,
6.0,
5.0,
6.0,
3.0,
5.0,
5.0,
7.0,
3.0,
7.0,
5.0,
6.0,
7.0,
5.0,
4.0,
3.0,
4.0,
4.0,
3.0,
2.0,
4.0,
5.0,
7.0,
5.0,
5.0,
4.0,
4.0,
8.0,
5.0,
5.0,
4.0,
7.0,
3.0,
7.0,
4.0,
6.0,
4.0,
7.0,
8.0,
3.0,
7.0,
4.0,
3.0,
4.0,
5.0,
2.0,
5.0,
5.0,
6.0,
5.0,
4.0,
5.0,
2.0,
3.0,
6.0,
3.0,
3.0,
4.0,
7.0,
7.0,
5.0,
7.0,
4.0,
5.0,
4.0,
4.0,
2.0,
3.0,
4.0,
8.0,
3.0,
6.0,
 6.0,
 2.0,
 …]
```

```
[107]: check('tests/q5_4.py')
```

```
[107]: <gofer.ok.OKTestsResult at 0x7fd1a923c4d0>
```

Let's take a look at the distribution of statistics, using a histogram.

```
[108]: #Draw a distribution of statistics
       Table().with_column('Coin Statistics', coin_statistics).hist()
```



Question 5.5 Given your observed value, do you believe that Gary's model is reasonable, or is our alternative more likely? Explain your answer using the distribution drawn in the previous problem.

Alternative more likely as 1 flip is not likely in the hist.

2.4 6. Submission

Once you're finished, select "Save and Checkpoint" in the File menu. Your name and course section number should be in the first and last cell of the assignment. Be sure you have run all cells with code and that the output from that is showing.

Double check that you have completed all of the free response questions as the auto-grader does NOT check that and YOU are responsible for knowing those questions are there and completing them as part of the grade for this homework. When ready, click "Print Preview" in the File menu. Print a copy from there in pdf format. (This means you right click and choose print and choose "save as pdf" from your printer options.) You will need to submit the pdf in Canvas by the deadline.

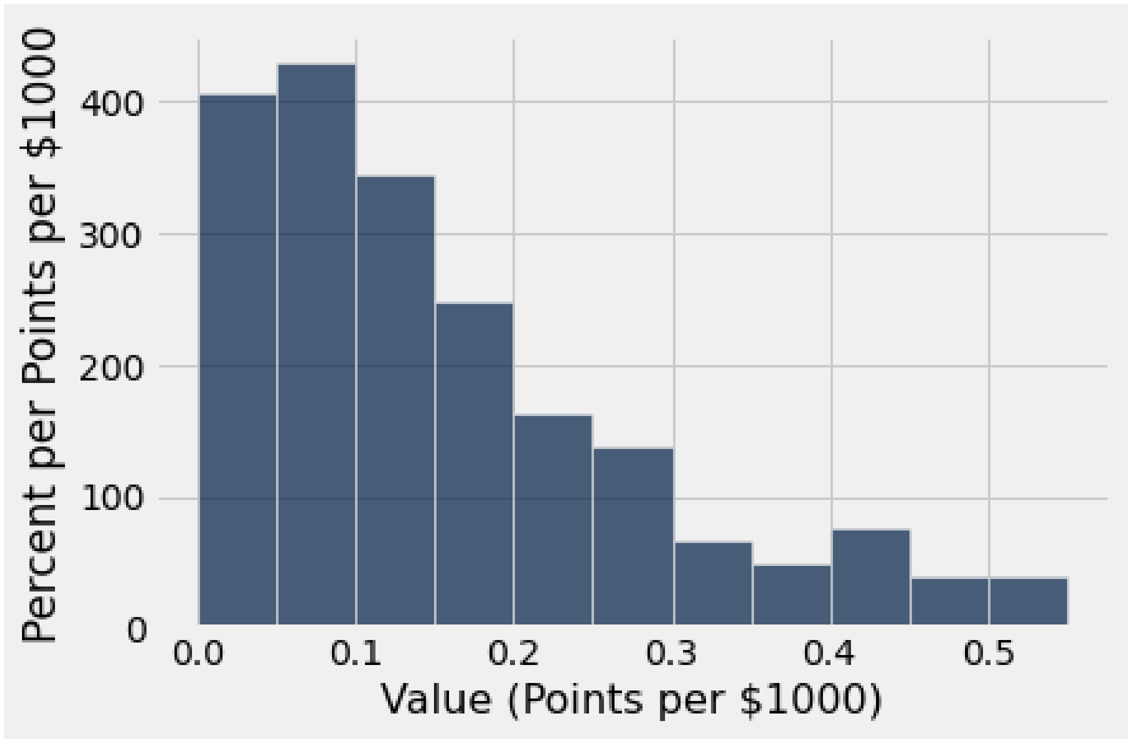
The gopher grader output and/or output from your coding are essential to helping your instructor grade your work correctly and in a timely manner.

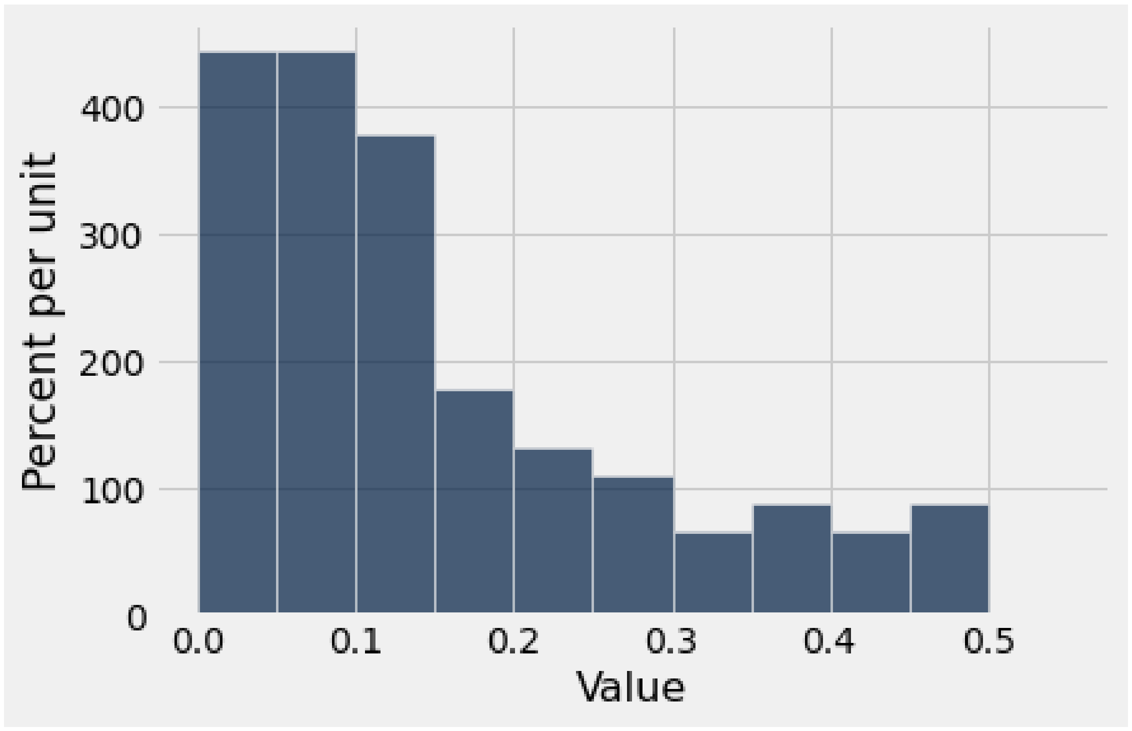
Files submitted that are missing the required output will lose some to all points so double check your pdf before submitting.

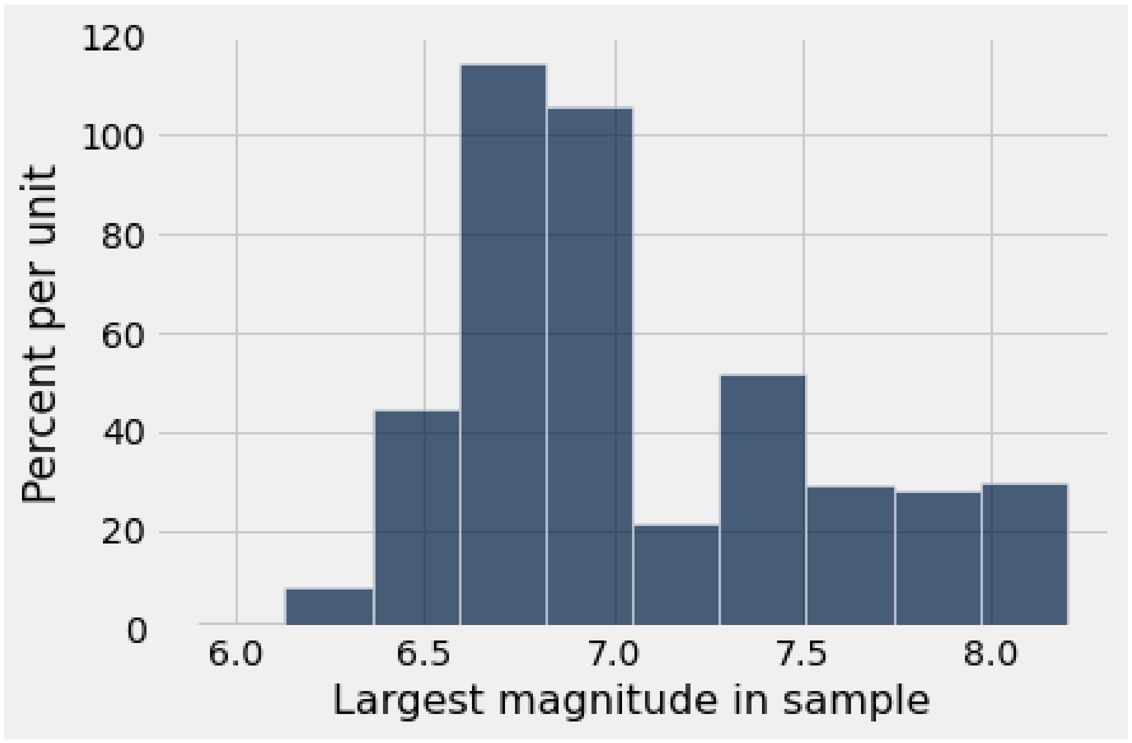
[109]:
```
# For your convenience, you can run this cell to run all the tests at once!
import glob
from gofer.ok import grade_notebook
if not globals().get('__GOFER_GRADER__', False):
    display(grade_notebook('hw06.ipynb', sorted(glob.glob('tests/q*.py'))))
```

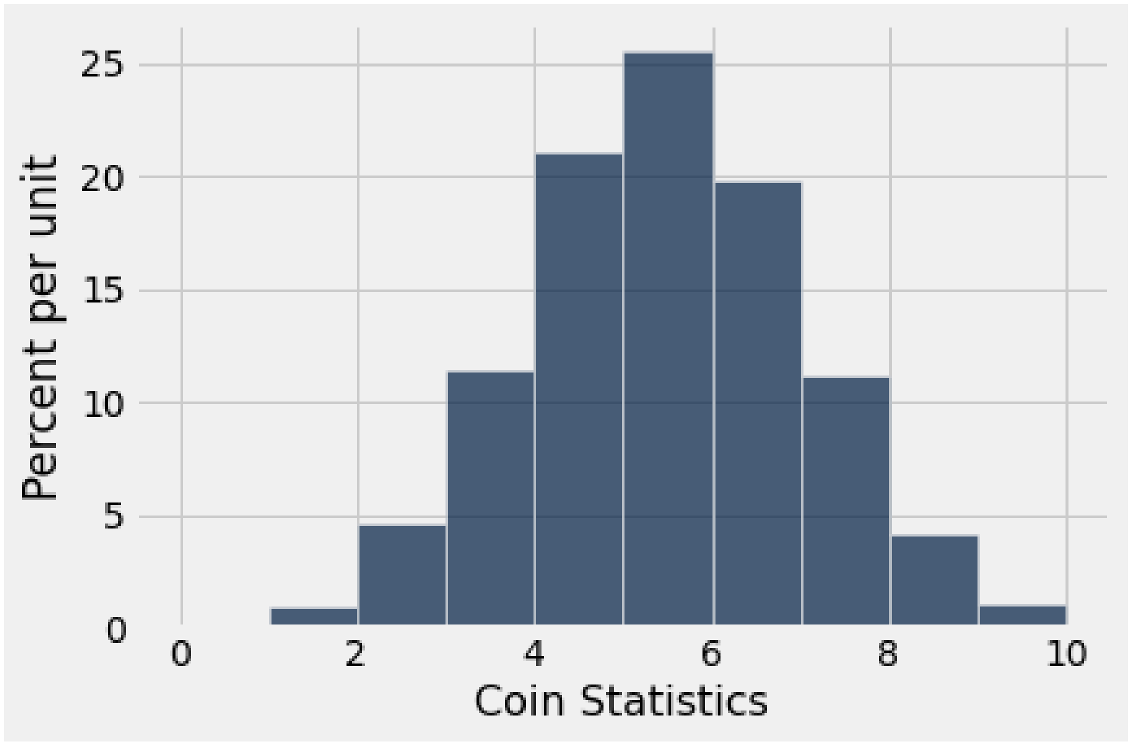
```
['tests/q1_1.py', 'tests/q1_2.py', 'tests/q1_3.py', 'tests/q1_4.py',
'tests/q2_1.py', 'tests/q2_2.py', 'tests/q2_3.py', 'tests/q2_4.py',
'tests/q2_6.py', 'tests/q3_1.py', 'tests/q4_2.py', 'tests/q4_3.py',
'tests/q4_4.py', 'tests/q5_1.py', 'tests/q5_3.py', 'tests/q5_4.py']
Question 1:

<gofer.ok.OKTestsResult at 0x7fd1a6d9bb50>

Question 2:

<gofer.ok.OKTestsResult at 0x7fd1a6e32710>

Question 3:

<gofer.ok.OKTestsResult at 0x7fd1a706b190>

Question 4:

<gofer.ok.OKTestsResult at 0x7fd1ab7328d0>

Question 5:

<gofer.ok.OKTestsResult at 0x7fd1ab732cd0>

Question 6:

<gofer.ok.OKTestsResult at 0x7fd1a6699090>

Question 7:

<gofer.ok.OKTestsResult at 0x7fd1a6699fd0>

Question 8:

<gofer.ok.OKTestsResult at 0x7fd1a6dcec50>

Question 9:
```

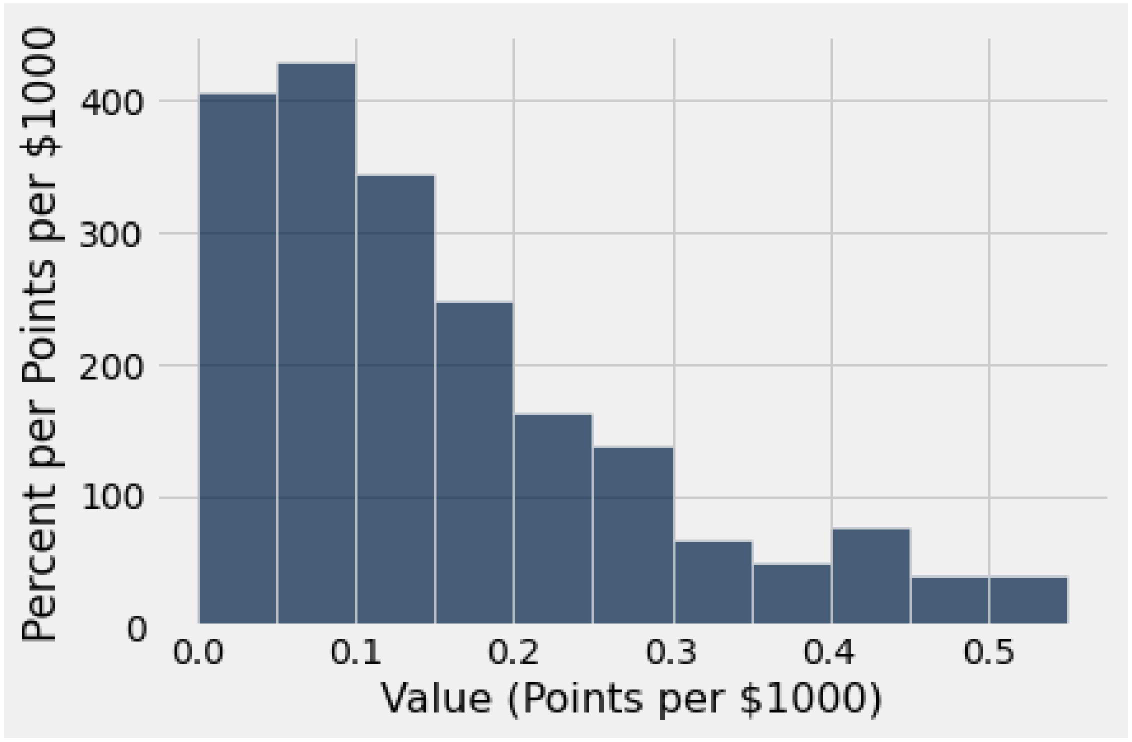
```
<gofer.ok.OKTestsResult at 0x7fd1a6e32e50>

Question 10:

<gofer.ok.OKTestsResult at 0x7fd1a6f32f90>

Question 11:

<gofer.ok.OKTestsResult at 0x7fd1a6fdf610>

Question 12:

<gofer.ok.OKTestsResult at 0x7fd1a6ff3190>

Question 13:

<gofer.ok.OKTestsResult at 0x7fd1a9250250>

Question 14:

<gofer.ok.OKTestsResult at 0x7fd1a936d2d0>

Question 15:

<gofer.ok.OKTestsResult at 0x7fd1aa432bd0>

Question 16:

<gofer.ok.OKTestsResult at 0x7fd1a9194e50>

1.0
```

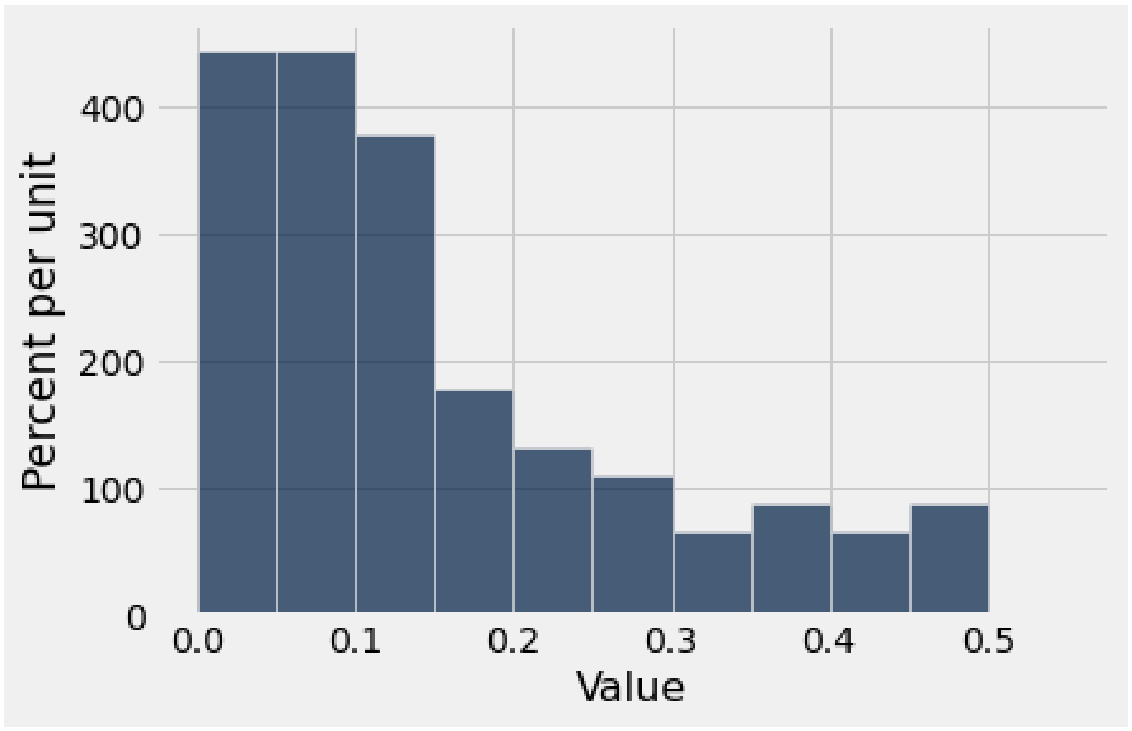


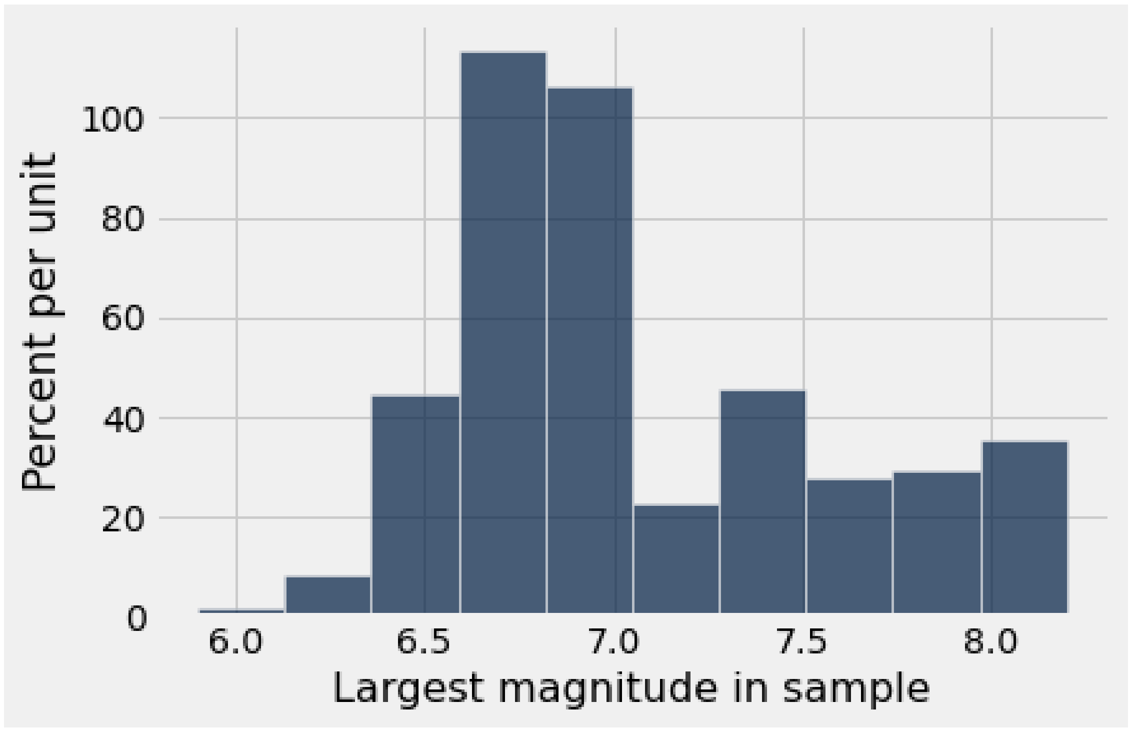

Percent per unit
400
300
200
100
0
0.0
0.1
0.2
0.3
0.4
0.5
Value

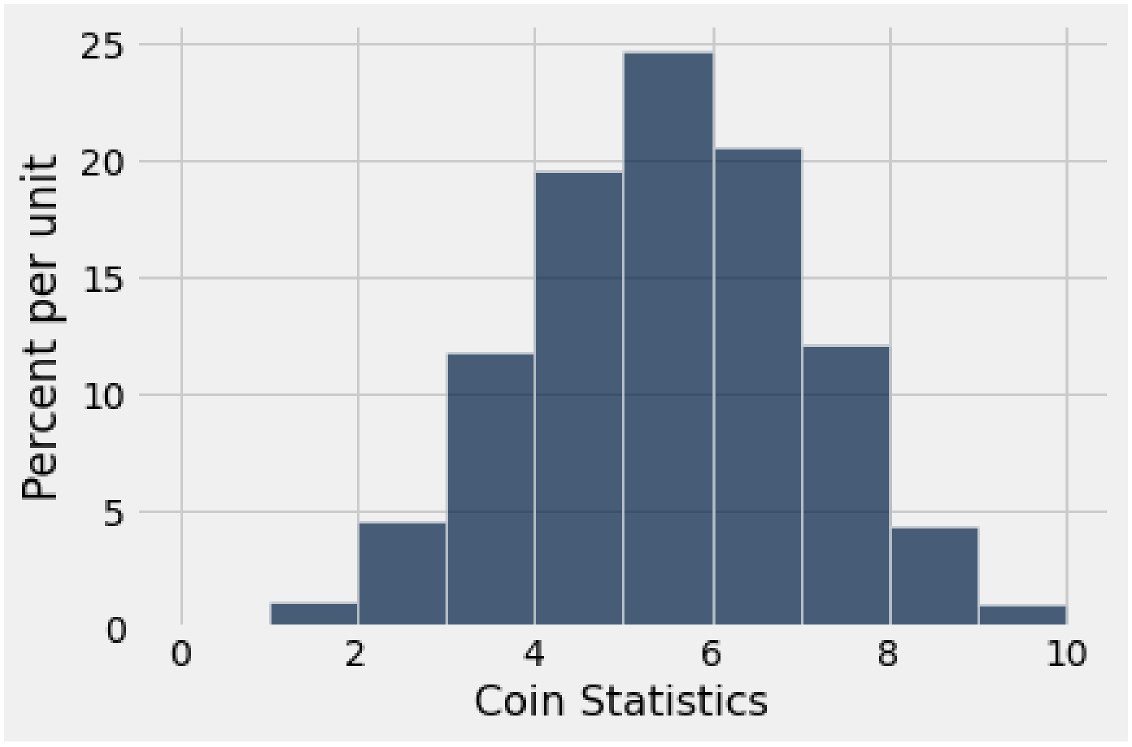

Percent per unit
120
100
80
60
40
20
0
6.0
6.5
7.0
7.5
8.0
Largest magnitude in sample


Percent per unit
0
5
10
15
20
25
0
2
4
6
8
10
Coin Statistics


Percent per Points per $1000
0
100
200
300
400
0.0
0.1
0.2
0.3
0.4
0.5
Value (Points per $1000)


Percent per unit
0
100
200
300
400
0.0
0.1
0.2
0.3
0.4
0.5
Value


Percent per unit
0
20
40
60
80
100
6.0
6.5
7.0
7.5
8.0
Largest magnitude in sample



Name: Allan Gongora

Section: 0131