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

Reading:

- Randomness
- Sampling and Empirical Distributions
- Testing Hypotheses

Please complete this notebook by filling in the cells provided.

For all problems that you must write our explanations and sentences for, you **must** provide your answer in the designated space. 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.

```
# 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)
import random
```

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. You must assign the provided variable to one of the following three integers, depending on whether the proposed answer is too low, too high, or correct.

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

You are more than welcome to create more cells across this notebook to use for arithmetic operations

Question 1. You roll a 6-sided die 10 times. What is the chance of getting 10 sixes?

Our proposed answer:

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

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

```
ten_sixes = 3
ten_sixes
```

```
3
```

Question 2. Take the same problem set-up as before, rolling a fair dice 10 times. What is the chance that every roll is less than or equal to 5?

Our proposed answer:

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

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

```
five_or_less = 2
five_or_less
```

```
2
```

Question 3. Assume we are picking a lottery ticket. We must choose three distinct numbers from 1 to 1000 and write them on a ticket. Next, someone picks three numbers one by one from a bowl with numbers from 1 to 1000 each time without putting the previous number back in. We win if our numbers are all called in order.

If we decide to play the game and pick our numbers as 12, 140, and 890, what is the chance that we win?

Our proposed answer:

$$\left(\frac{3}{1000}\right)^3$$

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

```
lottery = 1
```

Question 4. Assume we have two lists, list A and list B. List A contains the numbers [20,10,30], while list B contains the numbers [10,30,20,40,30]. 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 or equal to the number we drew from list B?

Our proposed solution:

$$1/5$$

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

Hint: Consider the different possible ways that the items in List A can be greater than or equal to items in List B. Try working out your thoughts with a pencil and paper, what do you think the correct solutions will be close to?

```
list_chances = 1
```

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, 26 uppercase letters of the English alphabet, and any number between 0-9 (inclusive), regardless of what it has hit before. There are no other keys on the keyboard.

This question is inspired by a mathematical theorem called the Infinite monkey theorem (https://en.wikipedia.org/wiki/Infinite_monkey_theorem), which postulates that if you put a monkey in the situation described above for an infinite time, they will eventually type out all of Shakespeare's works.

Question 1. Suppose the monkey hits the keyboard 5 times. Compute the chance that the monkey types the sequence `Data8`. (Call this `data_chance`.) Use algebra and type in an arithmetic equation that Python can evalute.

```
data_chance = pow(1/62,5)
data_chance
```

```
1.0915447684774164e-09
```

Question 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, 26 upper-case English letters, or any number between 0-9 (inclusive).

```
# We have provided the code below to compute a list called keys,
# containing all the lower-case English letters, upper-case English letters, and the digits
# want to verify what it contains.
import string
keys = list(string.ascii_lowercase + string.ascii_uppercase + string.digits)

def simulate_key_strike():
    """Simulates one random key strike."""
    return random.sample(keys,1)[0]

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

```
'F'
```

Question 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 called `key_strikes_array`, you can convert that to a string by calling `"".join(key_strikes_array)`

```
def simulate_several_key_strikes(num_strikes):
  ret = "";
  for x in range(num_strikes):
    ret = ret + simulate_key_strike()

  return ret

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

```
'10XVBXCkOxY'
```

Question 4. Call `simulate_several_key_strikes` 5000 times, each time simulating the monkey striking 5 keys. Compute the proportion of times the monkey types `"Data8"`, calling that proportion `data_proportion`.

```
count = 0;
for x in range(5000):
  if(simulate_several_key_strikes(5)=='Data8'):
    count += 1

data_proportion = count/5000
data_proportion
```

```
0.0
```

Question 5. Check the value your simulation computed for `data_proportion`. Is your simulation a good way to estimate the chance that the monkey types `"Data8"` in 5 strikes (the answer to question 1)? Why or why not?

yes and no, if there was a keyboard that had separate keys for uppercase letters then this simulates it pretty well- 5000 samples is more than enough data to rely on a conclusion

Question 6. Compute the chance that the monkey types the letter `"t"` at least once in the 5 strikes. Call it `t_chance`. Use algebra and type in an arithmetic equation that Python can evalute.

```
t_chance = 5/62
t_chance
```

```
0.08064516129032258
```

Question 7. Do you think that a computer simulation is more or less effective to estimate `t_chance` compared to when we tried to estimate `data_chance` this way? Why or why not? (You don't need to write a simulation, but it is an interesting exercise.)

no- i think theyre about the same- but a simulation can help explain the outcome a bit better

3. Sampling Basketball Players

This exercise uses salary data and game statistics for basketball players from the 2019-2020 NBA season. The data was collected from [Basketball-Reference](https://www.basketball-reference.com).

Run the next cell to load the two datasets.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
player_data = Table.read_table('/content/drive/MyDrive/player_data.csv')
salary_data = Table.read_table('/content/drive/MyDrive/salary_data.csv')
player_data.show(3)
salary_data.show(3)
```

Player	3P	2P	PTS
Steven Adams	0	4.4	10.7
Bam Adebayo	0	6.2	15.8
LaMarcus Aldridge	1.2	6.3	19.1

... (585 rows omitted)

Name	Salary
Stephen Curry	40231758
Chris Paul	38506482
Russell Westbrook	38178000

... (522 rows omitted)

Question 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 `"Name"` column.

```
full_data = player_data.join("Player", salary_data,"Name")
full_data
```

Player	3P	2P	PTS	Salary
Aaron Gordon	1.2	4.1	14.2	19863636
Aaron Holiday	1.5	2.2	9.9	2239200
Abdel Nader	0.7	1.3	5.7	1618520
Admiral Schofield	0.5	0.6	3.2	898310
Al Horford	1.4	3.4	12	28000000
Al-Farouq Aminu	0.5	0.9	4.3	9258000
Alec Burks	1.7	3.3	15.8	2320044
Alec Burks	1.8	3.3	16.1	2320044
Alec Burks	0	1	2	2320044
Alen Smailagić	0.3	1.3	4.7	898310

... (552 rows omitted)

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 their team is the number of 3 pointers and free throws the player scored in a season for every **$100000 of salary** (*Note*: the `Salary` column is in dollars, not hundreds of thousands of dollars). For example, Al Horford scored an average of 5.2 points for 3 pointers and free throws combined, and has a salary of **$28 million.** This is equivalent to 280 thousands of dollars, so his value is $\frac{5.2}{280}$. The formula is:

$$\frac{\text{"PTS"} - 2 * \text{"2P"}}{\text{"Salary"}\ /\ 100000}$$

Question 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! Refer to the python reference to look at `tbl.hist(...)` if necessary.

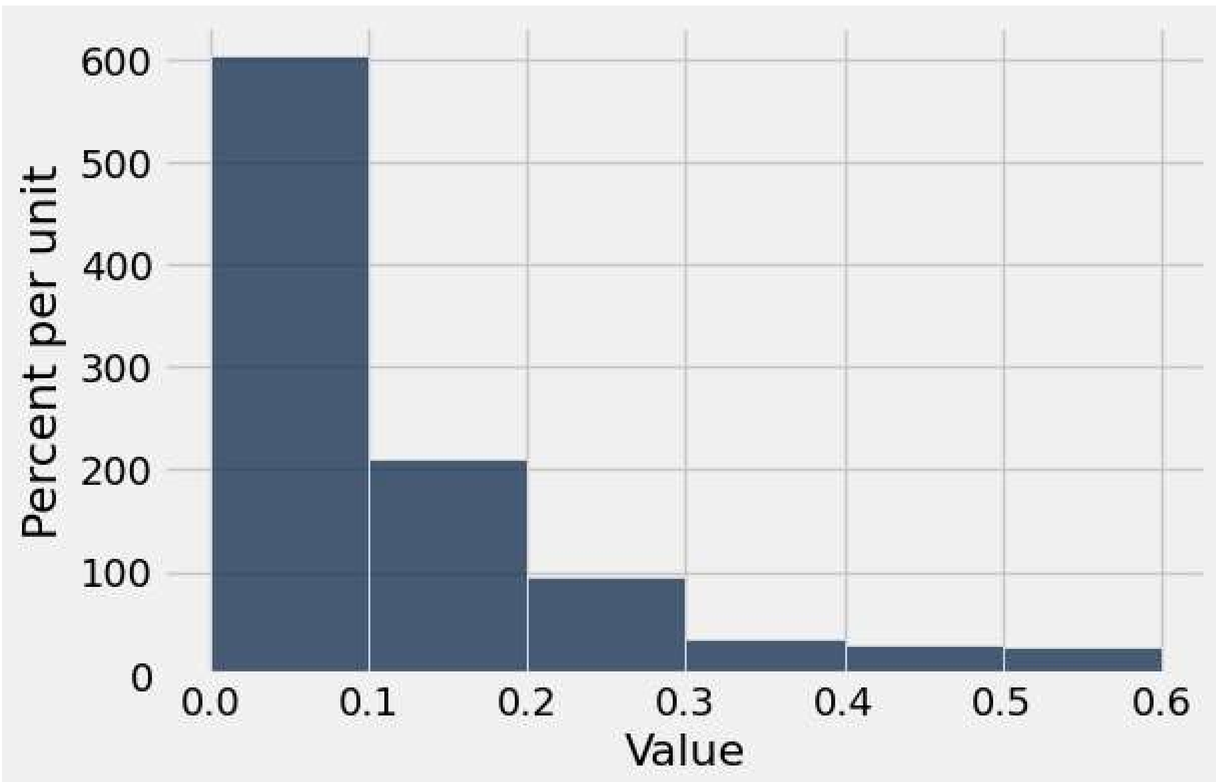
Just so you know: Informative histograms contain a majority of the data and **exclude outliers**

```
bins = np.arange(0, 0.7, .1) # Use this provided bins when you make your histogram

def find_value(PTS, TP, Salary):
    '''Returns the total score calculated by adding up the score of each quarter'''
    return ((PTS-2 * TP)/(Salary/100000));

#sum_scores(14, 7, 3, 0) #DO NOT CHANGE THIS LINE

value_table = full_data.apply(find_value, "PTS", "2P", "Salary")
full_data_with_value = full_data.with_column("Value", value_table)
full_data_with_value.select(5).hist(bins=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 50 players' salaries. The cell below loads those data.

```
sample_salary_data = Table.read_table("/content/drive/MyDrive/sample_salary_data.csv")
sample_salary_data.show(3)
```

Name	Salary
D.J. Wilson	2961120
Yante Maten	100000
Abdel Nader	1618520

... (47 rows omitted)

Question 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 2. Make sure to specify the units again in the histogram as stated in the previous problem. **Use the same bins, too.**

Hint: This will take several steps.

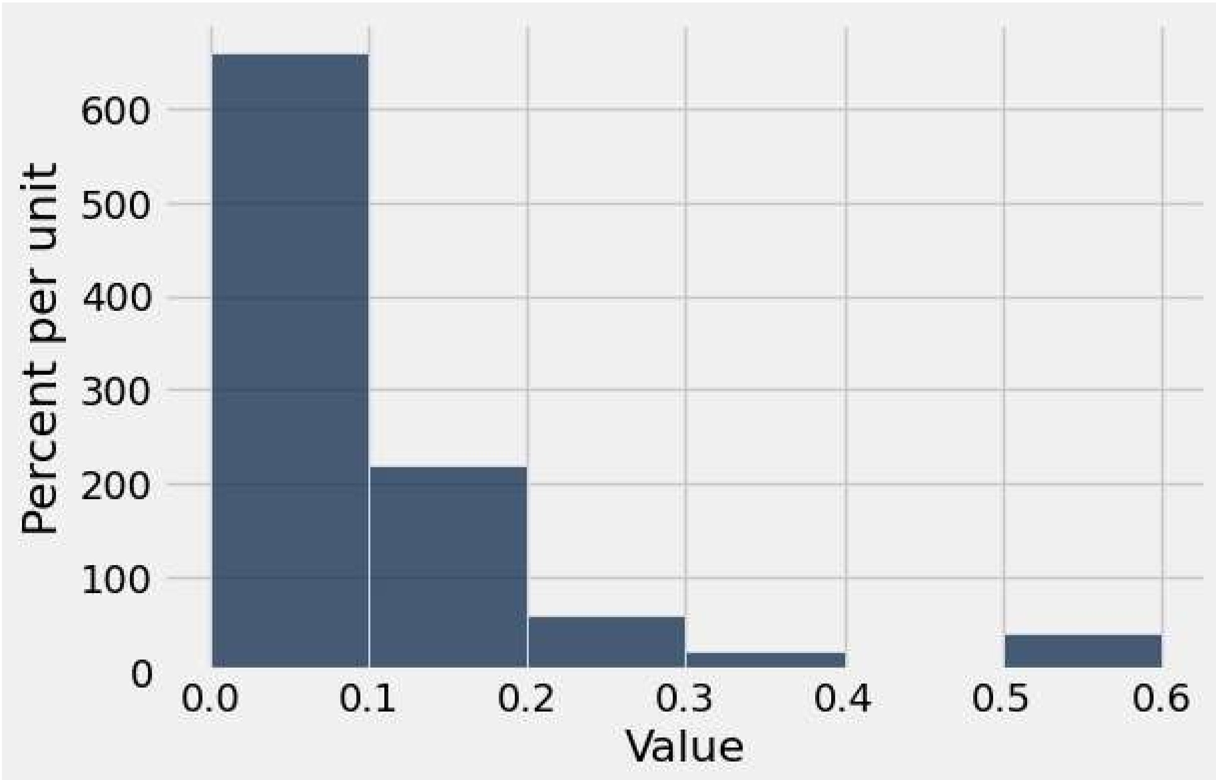
```
sample_data = player_data.join('Player', sample_salary_data, 'Name')


def find_value(PTS, TP, Salary):
    '''Returns the total score calculated by adding up the score of each quarter'''
    return ((PTS-2 * TP)/(Salary/100000));


sample_value_table = sample_data.apply(find_value, "PTS", "2P", "Salary")
sample_data_with_value = sample_data.with_column("Value", sample_value_table)
sample_data_with_value.show(2)
sample_data_with_value.select(5).hist(bins=bins)
```

Player	3P	2P	PTS	Salary	Value
Abdel Nader	0.7	1.3	5.7	1618520	0.191533
Alize Johnson	0	0.6	1.6	1416852	0.0282316

... (49 rows omitted)



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

Question 4. Complete the statements below by setting each relevant variable name to the value that correctly fills the blank.

- The plot in question 2 displayed a(n) [`distribution_1`] distribution of the population of [`player_count_1`] players. The areas of the bars in the plot sum to [`area_total_1`].
- The plot in question 3 displayed a(n) [`distribution_2`] distribution of the sample of [`player_count_2`] players. The areas of the bars in the plot sum to [`area_total_2`].

`distribution_1` and `distribution_2` should be set to one of the following strings: `"empirical"` or `"probability"`.

`player_count_1`, `area_total_1`, `player_count_2`, and `area_total_2` should be set to integers.

Remember that areas are represented in terms of percentages.

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.

```
distribution_1 = empirical
player_count_1 = 552
area_total_1 = %97.5

distribution_2 = empirical
player_count_2 = 50
area_total_2 = %95
```

```
  File "<ipython-input-18-3c413d1f31a5>", line 3
    area_total_1 = %95
                   ^
SyntaxError: invalid syntax
```

Question 5. For which range of values does the plot in question 3 better depict the distribution of the **population's player values**: 0 to 0.3, or above 0.3? Explain your answer.

above .3- anything less is an outlier and not a good depiction of the data

4. Earthquakes

The next cell loads a table containing information about **every earthquake with a magnitude above 5** in 2019 (smaller earthquakes are generally not felt, only recorded by very sensitive equipment), compiled by the US Geological Survey. (source: https://earthquake.usgs.gov/earthquakes/search/)

```
earthquakes = Table().read_table('/content/drive/MyDrive/earthquakes_2019.csv').select(['tim
earthquakes
```

time	mag	place
2019-12-31T11:22:49.734Z	5	245km S of L'Esperance Rock, New Zealand
2019-12-30T17:49:59.468Z	5	37km NNW of Idgah, Pakistan
2019-12-30T17:18:57.350Z	5.5	34km NW of Idgah, Pakistan
2019-12-30T13:49:45.227Z	5.4	33km NE of Bandar 'Abbas, Iran
2019-12-30T04:11:09.987Z	5.2	103km NE of Chichi-shima, Japan
2019-12-29T18:24:41.656Z	5.2	Southwest of Africa
2019-12-29T13:59:02.410Z	5.1	138km SSW of Kokopo, Papua New Guinea
2019-12-29T09:12:15.010Z	5.2	79km S of Sarangani, Philippines
2019-12-29T01:06:00.130Z	5	9km S of Indios, Puerto Rico
2019-12-28T22:49:15.959Z	5.2	128km SSE of Raoul Island, New Zealand

... (1626 rows omitted)

If we were studying all human-detectable 2019 earthquakes and had access to the above data, we'd be in good shape - however, if the USGS didn't publish the full data, we could still learn something about earthquakes from just a smaller subsample. If we gathered our sample correctly, we could use that subsample to get an idea about the distribution of magnitudes (above 5, of course) throughout the year!

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.

```
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]
```

```
[6.458999999999987, 5.279000000000008]
```

Question 1. Are these samples representative of the population of earthquakes in the original table (that is, the should we expect the mean to be close to the population mean)?

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

the first sample is not accurate because we sort it- the second sample could be more accurate because its randomized

Question 2. Write code to produce a sample of size 200 that is representative of the population. 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, we learned what kind of samples should be used to properly represent the population.

```
representative_sample = earthquakes.take(np.arange(200))
representative_mean = np.mean(representative_sample.column('mag'))
representative_mean
```

```
5.3054999999999994
```

Question 3. Suppose we want to figure out what the biggest magnitude earthquake was in 2019, but we only have our representative sample of 200. Let's see if trying to find the biggest magnitude in the population from a random sample of 200 is a reasonable idea!

Write code that takes many random samples from the `earthquakes` table and finds the maximum of each sample. You should take a random sample of size 200 and do this 5000 times. Assign the array of maximum magnitudes you find to `maximums`.

```
maximums = []
for i in np.arange(5000):
    maximums.append(max(earthquakes.take(np.arange(200)).column(1)))
```

```
#Histogram of your maximums
Table().with_column('Largest magnitude in sample', maximums).hist('Largest magnitude in samp
```

Question 4. Now find the magnitude of the actual strongest earthquake in 2019 (not the maximum of a sample). This will help us determine whether a random sample of size 200 is likely to help you determine the largest magnitude earthquake in the population.

```
strongest_earthquake_magnitude = max(earthquakes.column(1))
strongest_earthquake_magnitude
```

```
8.0
```

Question 5. Explain whether you believe you can accurately use a sample size of 200 to determine the maximum. What is one problem with using the maximum as your estimator? Use the histogram above to help answer.

well- finding the mean from a sample is doable but things like max and min rely on outliers that only appear in the data a few times- so it's less likely to be accurate

5. Assessing Jade's Models

Games with Jade

Our friend Jade comes over and asks us to play a game with her. The game works like this:

> We will draw randomly with replacement from a simplified 13 card deck with 4 face cards (A, J, Q, K), and 9 numbered cards (2, 3, 4, 5, 6, 7, 8, 9, 10). If we draw cards with replacement 13 times, and if the number of face cards is greater than or equal to 4, we lose.
>
> Otherwise, Jade wins.

We play the game once and we lose, observing 8 total face cards. We are angry and accuse Jade of cheating! Jade is adamant, however, that the deck is fair.

Jade's model claims that there is an equal chance of getting any of the cards (A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K), but we do not believe her. We believe that the deck is clearly rigged, with face cards (A, J, Q, K) being more likely than the numbered cards (2, 3, 4, 5, 6, 7, 8, 9, 10).

Question 1

Assign `deck_model_probabilities` to a two-item array containing the chance of drawing a face card as the first element, and the chance of drawing a numbered card as the second element under Jade's model. Since we're working with probabilities, make sure your values are between 0 and 1.

```
deck_model_probabilities = np.array([4/13, 9/13])
deck_model_probabilities
```

```
array([ 0.30769231,  0.69230769])
```

Question 2

We believe Jade's model is incorrect. In particular, we believe there to be a larger chance of getting a face card. 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 actual number of face cards we get in 13 draws

2. The distance (absolute value) between the actual number of face cards in 13 draws and the expected number of face cards in 13 draws (4)
3. The expected number of face cards in 13 draws (4)

```
statistic_choice = 2
statistic_choice
```

```
2
```

Question 3

Define the function `deck_simulation_and_statistic`, which, given a sample size and an array of model proportions (like the one you created in Question 1), returns the number of face cards in one simulation of drawing a card under the model specified in `model_proportions`.

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

```
def deck_simulation_and_statistic(sample_size, model_proportions):
    sample = sample_proportions(sample_size, model_proportions)
    num_face_cards = sample.item(0) * sample_size

    return num_face_cards
deck_simulation_and_statistic(13, deck_model_probabilities)
```

```
2.0
```

Question 4

Use your function from above to simulate the drawing of 13 cards 5000 times under the proportions that you specified in Question 1. Keep track of all of your statistics in `deck_statistics`.

```
deck_statistics = make_array()
for _ in range(5000):
    num_face_cards = deck_simulation_and_statistic(13, deck_model_probabilities)
    deck_statistics = np.append(deck_statistics, num_face_cards)
```

Let’s take a look at the distribution of simulated statistics.

```
#Draw a distribution of statistics
Table().with_column('Deck Statistics', deck_statistics).hist()
```