hw06

March 25, 2025

[1]:
```
# Initialize Otter
import otter
grader = otter.Notebook("hw06.ipynb")
```

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

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

Helpful Resource: - Python Reference: Cheat sheet of helpful array & table methods used in Data 8! - Sampling Methods Guide: Guide for the randomization methods.

Recommended Readings: * Randomness * Sampling and Empirical Distributions * Testing Hypotheses

Please complete this notebook by filling in the cells provided. **Before you begin, execute the cell below to setup the notebook by importing some helpful libraries.** Each time you start your server, you will need to execute this cell again.

For all problems that you must write 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. Otherwise, you will fail tests that you thought you were passing previously!

Deadline:

This assignment is **due Wednesday, 10/9 at 5:00pm PT**. Submissions after this time will be accepted for 24 hours and will incur a 20% penalty. Any submissions later than this 24 hour period will not be accepted unless an extension has been granted as per the policies page. Turn it in by Tuesday, 10/8 at 5:00pm PT for 5 extra credit points.

Note: This homework has hidden tests on it. That means even though tests may say 100% passed, it doesn't mean your final grade will be 100%. We will be running more tests for correctness once everyone turns in the homework.

Directly sharing answers is not okay, but discussing problems with the course staff or with other students is encouraged. Refer to the policies page to learn more about how to learn cooperatively.

You should start early so that you have time to get help if you're stuck. Office hours are held Monday through Friday in Warren Hall 101B. The office hours schedule appears here.

The point breakdown for this assignment is given in the table below: | Category | Points | | — | — | | Autograder (Coding questions) | 90 | | Written | 10 | | **Total** | 100 |

1.1 1. Roulette

[2]:
```
# Run this cell to set up the notebook, 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)
```

A Nevada roulette wheel has 38 pockets and a small ball that rests on the wheel. When the wheel is spun, the ball comes to rest in one of the 38 pockets. That pocket is declared the winner.

The pockets are labeled 0, 00, 1, 2, 3, 4, … , 36. Pockets 0 and 00 are green, and the other pockets are alternately red and black. The table `wheel` is a representation of a Nevada roulette wheel. **Note that *both* columns consist of strings.** Below is an example of a roulette wheel!

Run the cell below to load the `wheel` table.

[3]:
```
wheel = Table.read_table('roulette_wheel.csv', dtype=str)
wheel
```

[3]:
```
Pocket | Color
00     | green
0      | green
1      | red
2      | black
3      | red
4      | black
5      | red
6      | black
7      | red
8      | black
… (28 rows omitted)
```

1.1.1 Betting on Red

If you bet on *red*, you are betting that the winning pocket will be red. This bet *pays 1 to 1*. That means if you place a one-dollar bet on red, then:

- If the winning pocket is red, you gain 1 dollar. That is, you get your original dollar back, plus one more dollar.
- If the winning pocket is not red, you lose your dollar. In other words, you gain -1 dollars.

Let's see if you can make money by betting on red at roulette.

Question 1. Define a function `dollar_bet_on_red` that takes the name of a color and returns your gain in dollars if that color had won and you had placed a one-dollar bet on red. Remember that the gain can be negative. Make sure your function returns an integer. **(4 points)**

Note: You can assume that the only colors that will be passed as arguments are red, black, and green. Your function doesn't have to check that.

[4]:
```
def dollar_bet_on_red(color):
    if color == 'red':
        return 1
    else:
        return -1
```

[5]:
```
grader.check("q1_1")
```

[5]:
```
q1_1 results: All test cases passed!
```

Run the cell below to make sure your function is working.

[6]:
```
print(dollar_bet_on_red('green'))
print(dollar_bet_on_red('black'))
print(dollar_bet_on_red('red'))
```

```
-1
-1
1
```

Question 2. Add a column labeled `Winnings: Red` to the table `wheel`. For each pocket, the column should contain your gain in dollars if that pocket won and you had bet one dollar on red. Your code should use the function `dollar_bet_on_red`. **(4 points)**

Hint: You should not need a `for` loop for this question, instead try using a table method!

[7]:
```
red_winnings = wheel.apply(dollar_bet_on_red, 'Color')
wheel = wheel.with_column("Winnings: Red", red_winnings)
wheel
```

[7]:
```
Pocket | Color | Winnings: Red
00     | green | -1
0      | green | -1
1      | red   | 1
2      | black | -1
```

```
3      | red   | 1
4      | black | -1
5      | red   | 1
6      | black | -1
7      | red   | 1
8      | black | -1
… (28 rows omitted)
```

[8]:
```
grader.check("q1_2")
```

[8]:
```
q1_2 results: All test cases passed!
```

1.1.2 Simulating 10 Bets on Red

Roulette wheels are set up so that each time they are spun, the winning pocket is equally likely to be any of the 38 pockets regardless of the results of all other spins. Let's see what would happen if we decided to bet one dollar on red each round.

Question 3. Create a table `ten_bets` by sampling the table `wheel` to simulate 10 spins of the roulette wheel. Your table should have the same three column labels as in `wheel`. Once you've created that table, set `sum_bets` to your net gain in all 10 bets, assuming that you bet one dollar on red each time. **(4 points)**

Note: The Sampling Methods Guide may be helpful!

Hint: It may be helpful to print out `ten_bets` after you create it!

[9]:
```
ten_bets = wheel.sample(10).select(0, 1, 2)
sum_bets = sum(ten_bets.column("Winnings: Red"))
sum_bets
```

[9]:
```
0
```

[10]:
```
grader.check("q1_3")
```

[10]:
```
q1_3 results: All test cases passed!
```

Run the cells above a few times to see how much money you would make if you made 10 one-dollar bets on red. Making a negative amount of money doesn't feel good, but it is a reality in gambling. Casinos are a business, and they make money when gamblers lose.

Question 4. Let's see what would happen if you made more bets. Define a function `net_gain_red` that takes the number of bets and returns the net gain in that number of one-dollar bets on red. **(4 points)**

Hint: You should use your `wheel` table within your function.

[11]:
```
def net_gain_red(num_bets):
    bets = wheel.sample(num_bets)
    return sum(bets.column("Winnings: Red"))
```

[12]:
```
grader.check("q1_4")
```

[12]:
```
q1_4 results: All test cases passed!
```

Run the cell below a few times to make sure that the results are similar to those you observed in the previous exercise.

[13]:
```
net_gain_red(10)
```

[13]:
```
0
```

Question 5. Complete the cell below to simulate the net gain in 200 one-dollar bets on red, repeating the process 10,000 times. After the cell is run, `simulated_gains_red` should be an array with 10,000 entries, each of which is the net gain in 200 one-dollar bets on red. **(4 points)**

Hint: Think about which computational tool might be helpful for simulating a process multiple times. Lab 5 might be a good resource to look at!

Note: This cell might take a few seconds to run.

[14]:
```
num_bets = 200
repetitions = 10000

simulated_gains_red = make_array()
for i in range(repetitions):
    simulated_gains_red = np.append(simulated_gains_red, net_gain_red(num_bets))

len(simulated_gains_red) # Do not change this line! Check that␣
 ↪simulated_gains_red is length 10000.
```

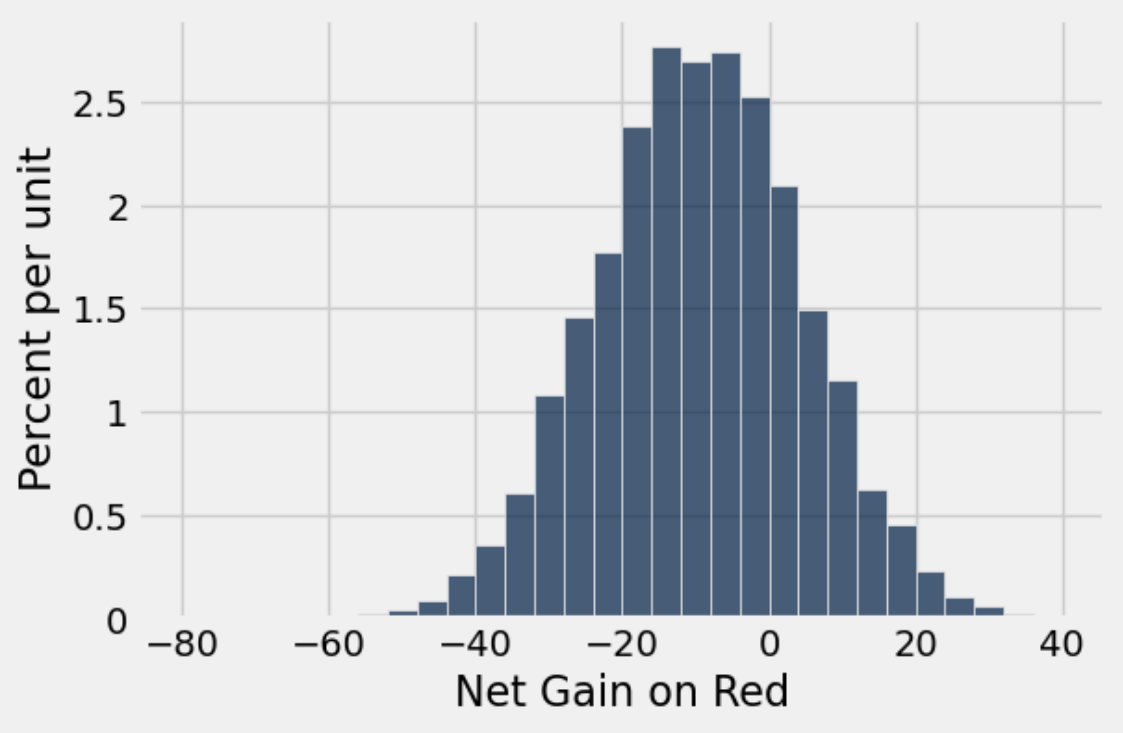
[14]:
```
10000
```

[15]:
```
grader.check("q1_5")
```

[15]:
```
q1_5 results: All test cases passed!
```

Run the cell below to visualize the results of your simulation.

[16]:
```
gains = Table().with_columns('Net Gain on Red', simulated_gains_red)
gains.hist(bins = np.arange(-80, 41, 4))
```



Question 6: Using the histogram above, decide whether the following statement is true or false:

If you make 200 one-dollar bets on red, your chance of losing money is more than 50%.

Assign `loss_more_than_50` to either `True` or `False` depending on your answer to the question. **(2 points)**

[17]:
```
loss_more_than_50 = True
```

[18]:
```
grader.check("q1_6")
```

[18]:
```
q1_6 results: All test cases passed!
```

1.1.3 Betting on a Split

If betting on red doesn't seem like a good idea, maybe a gambler might want to try a different bet. A bet on a *split* is a bet on two consecutive numbers such as 5 and 6. This bets pays 17 to 1. That means if you place a one-dollar bet on the split 5 and 6, then:

- If the winning pocket is either 5 or 6, your gain is 17 dollars.
- If any other pocket wins, you lose your dollar, so your gain is -1 dollars.

Question 7. Define a function `dollar_bet_on_split` that takes a pocket number and returns your gain in dollars if that pocket won and you had bet one dollar on the 5-6 split. **(4 points)**

Hint: Remember that the pockets are represented as strings.

[19]:
```
def dollar_bet_on_split(number):
    if number in ['5', '6']:
        return 17
    else:
        return -1
```

[20]:
```
grader.check("q1_7")
```

[20]:
```
q1_7 results: All test cases passed!
```

Run the cell below to check that your function is doing what it should.

[21]:
```
print(dollar_bet_on_split('5'))
print(dollar_bet_on_split('6'))
print(dollar_bet_on_split('00'))
print(dollar_bet_on_split('23'))
```

```
17
17
-1
-1
```

Question 8. Add a column `Winnings: Split` to the `wheel` table. For each pocket, the column should contain your gain in dollars if that pocket won and you had bet one dollar on the 5-6 split. **(4 points)**

[22]:
```
split_winnings = wheel.apply(dollar_bet_on_split, "Pocket")
wheel = wheel.with_column("Winnings: Split", split_winnings)
wheel.show(5)  # Do not change this line.
```

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

[23]:
```
grader.check("q1_8")
```

[23]:
```
q1_8 results: All test cases passed!
```

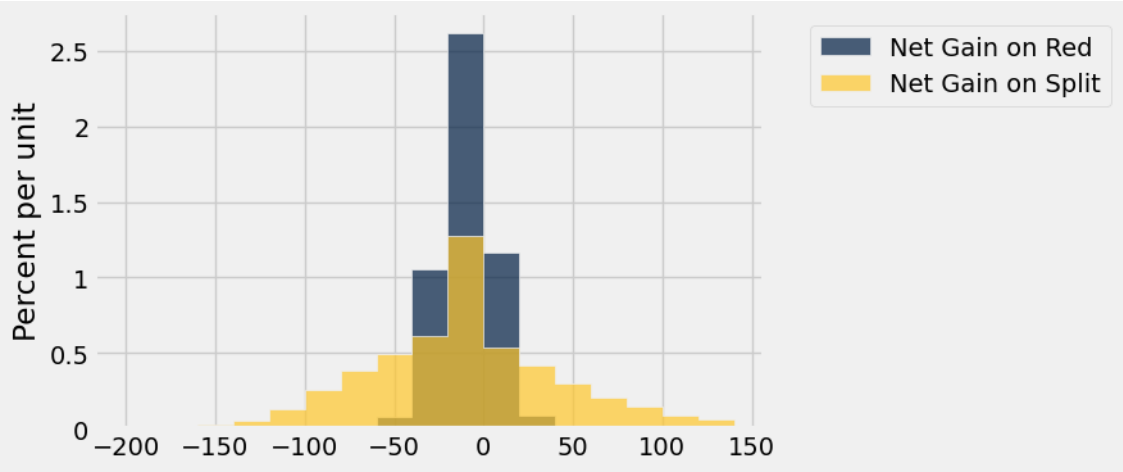
Question 9. Simulate the net gain in 200 one-dollar bets on the 5-6 split, repeating the process 10,000 times and saving your gains in the array `simulated_gains_split`. **(5 points)**

Hint: Your code in Questions 4 and 5 may be helpful here!

[24]:
```
simulated_gains_split = make_array()
def net_gain_split(num_bets):
    bets = wheel.sample(num_bets)
    return sum(bets.column("Winnings: Split"))
```

```
for i in range(10000):
    simulated_gains_split = np.append(simulated_gains_split,␣
 ↪net_gain_split(200))

# Do not change the two lines below
gains = gains.with_columns('Net Gain on Split', simulated_gains_split)
gains.hist(bins = np.arange(-200, 150, 20))
```



[25]:
```
grader.check("q1_9")
```

[25]:
```
q1_9 results: All test cases passed!
```

Question 10. Look carefully at the visualization above, and assign `histogram_statements` to an array of the numbers of each statement below that can be correctly inferred from the overlaid histogram. **(2 points)**

1. If you bet one dollar 200 times on a split, your chance of losing money is more than 50%.
2. If you bet one dollar 200 times in roulette, your chance of making more than 50 dollars is greater if you bet on a split each time than if you bet on red each time.
3. If you bet one dollar 200 times in roulette, your chance of losing more than 50 dollars is greater if you bet on a split each time than if you bet on red each time.

Hint: We've already seen one of these statements in a prior question.

[26]:
```
histogram_statements = make_array(1)
```

[27]:
```
grader.check("q1_10")
```

[27]:
```
q1_10 results: All test cases passed!
```

If this exercise has put you off playing roulette, it has done its job. If you are still curious about other bets, here they all are, and here is the bad news. The house – that is, the casino – always

has an edge over the gambler.

1.2 2. Chances

Before you do this exercise, make sure you understand the logic behind all the examples in Section 9.5.

Good ways to approach probability calculations include:

- Thinking one trial at a time: What does the first one have to be? Then what does the next one have to be?
- Breaking up the event into distinct ways in which it can happen.
- Seeing if it is easier to find the chance that the event does not happen.

1.2.1 Finding Chances

On each spin of a roulette wheel, all 38 pockets are equally likely to be the winner regardless of the results of other spins. Among the 38 pockets, 18 are red, 18 black, and 2 green. In each part below, write an expression that evaluates to the chance of the event described.

Question 1. The winning pocket is black on all of the first three spins. **(3 points)**

[28]:
```
first_three_black = (18/38) ** 3
```

[29]:
```
grader.check("q2_1")
```

[29]:
```
q2_1 results: All test cases passed!
```

Question 2. The color green never wins in the first 10 spins. **(3 points)**

[30]:
```
no_green = (36/38) ** 10
```

[31]:
```
grader.check("q2_2")
```

[31]:
```
q2_2 results: All test cases passed!
```

Question 3. The color green wins **at least once** on the first 10 spins. **(3 points)**

[32]:
```
at_least_one_green = 1 - no_green
```

[33]:
```
grader.check("q2_3")
```

[33]:
```
q2_3 results: All test cases passed!
```

Question 4. Two of the three colors **never** win in the first 10 spins. **(3 points)**

Hint: What situation(s) lead to two of the three colors never winning in the first 10 spins?

[34]:
```
lone_winners = (18 / 38) ** 10 + (18 / 38) ** 10 + (2 / 18) ** 10
```

[35]:
```
grader.check("q2_4")
```

[35]:
```
q2_4 results: All test cases passed!
```

1.2.2 Comparing Chances

In each of Questions 5-7, two events A and B are described. Choose from one of the following three options and set each answer variable to a single integer:

1. Event A is more likely than Event B
2. Event B is more likely than Event A
3. The two events have the same chance.

You should be able to make the choices **without calculation**. Good ways to approach this exercise include imagining carrying out the chance experiments yourself, one trial at a time, and by thinking about the law of averages.

Question 5. A child picks four times at random from a box that has four toy animals: a bear, an elephant, a giraffe, and a kangaroo. **(2 points)**

- Event A: all four different animals are picked (assuming the child picks without replacement)
- Event B: all four different animals are picked (assuming the child picks with replacement)

[36]:
```
toys_option = 1
```

[37]:
```
grader.check("q2_5")
```

[37]:
```
q2_5 results: All test cases passed!
```

Question 6. In a lottery, two numbers are drawn at random with replacement from the integers 1 through 1000. **(2 points)**

- Event A: The number 8 is picked on both draws
- Event B: The same number is picked on both draws

[38]:
```
lottery_option = 2
```

[39]:
```
grader.check("q2_6")
```

[39]:
```
q2_6 results: All test cases passed!
```

Question 7. A fair coin is tossed repeatedly. **(2 points)**

- Event A: There are 60 or more heads in 100 tosses

- Event B: There are 600 or more heads in 1000 tosses

Hint: Think about the law of averages!

[40]:
```
coin_option = 1
```

[41]:
```
grader.check("q2_7")
```

[41]:
```
q2_7 results: All test cases passed!
```

1.3 3. Three Ways Python Draws Random Samples

You have learned three ways to draw random samples using Python:

- `tbl.sample` draws a random sample of rows from the table `tbl`. The output is a table consisting of the sampled rows.
- `np.random.choice` draws a random sample from a population whose elements are in an array. The output is an array consisting of the sampled elements.
- `sample_proportions` draws from a categorical distribution whose proportions are in an array. The output is an array consisting of the sampled proportions in all the categories.

[42]:
```
# Look through this code and run this cell for questions 1 and 2
top_movies = Table.read_table('top_movies_2017.csv').select(0, 1)
top_movies.show(3)
```

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

[43]:
```
# Look through this code and run this cell for questions 1 and 2
studios_with_counts = top_movies.group('Studio').sort('count', descending=True)
studios_with_counts.show(3)
```

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

[44]:
```
# Look through this code and run this cell for questions 1 and 2
studios_of_all_movies = top_movies.column('Studio')
distinct_studios = studios_with_counts.column('Studio')

print("studios_of_all_movies:", studios_of_all_movies[:10], "...")
print("\n distinct_studios:", distinct_studios)
```

```
studios_of_all_movies: ['MGM' 'Fox' 'Fox' 'Universal' 'Paramount' 'Paramount'
'Universal' 'MGM'
 'Warner Brothers' 'Disney'] …

 distinct_studios: ['Buena Vista' 'Warner Brothers' 'Paramount' 'Fox'
'Universal' 'Disney'
 'Columbia' 'MGM' 'United Artists' 'Newline' 'Paramount/Dreamworks' 'Sony'
 'Dreamworks' 'Lionsgate' 'RKO' 'Tristar' 'AVCO' 'IFC' 'Metro' 'NM' 'Orion'
 'Selz.' 'Sum.']
```

[45]:
```
# Look through this code and run this cell for questions 1 and 2
studio_counts_only = studios_with_counts.column('count')
studio_proportions_only = studio_counts_only / sum(studio_counts_only)

print("studio_counts_only:", studio_counts_only)
print("\n studio_proportions_only:", studio_proportions_only)
```

```
studio_counts_only: [35 29 25 24 23 11  9  7  6  5  4  4  3  3  3  2  1  1  1  1
1  1  1]

 studio_proportions_only: [ 0.175  0.145  0.125  0.12   0.115  0.055  0.045
0.035  0.03   0.025
  0.02   0.02   0.015  0.015  0.015  0.01   0.005  0.005  0.005  0.005
  0.005  0.005  0.005]
```

In Questions 1 and 2, we will present a scenario. For each scenario, we will ask whether the desired result can be achieved by using a given function and the following tables/arrays: `top_movies`, `studios_with_counts`, `studios_of_all_movies`, `distinct_studios`, `studio_counts_only` and `studio_proportions_only`. You can assume we know which index of the array corresponds with the studio with that movie count/proportion.

Note: ***Do not*** *explain your answer; please answer yes or no and the name of the array/table.*

Question 1. Simulate a sample of 10 movies drawn at random with replacement from the 200 movies. Using just this sample, do we have enough information to output `True` if Paramount appears more often than Warner Brothers among studios that released the sampled movies, and `False` otherwise?

Example Answer: Yes, with "studio_proportions_only".

Note: Do not explain your answer for any of the options you've chosen; please follow the structure of the example answer provided.

Question 1(a) Can this be done using the `sample` function? If yes, what table would we call `sample` on? **(1 point)**

Yes, on "top_movies".

Question 1(b) Can this be done using the `np.random.choice` function? If yes, what array would we call `np.random.choice` on? **(1 point)**

Yes, on "studios_of_all_movies".

Question 1(c) Can this be done using the `sample_proportions` function? If yes, what array would we call `sample_proportions` on? **(1 point)**

Yes, on "studio_proportions_only".

Question 2. Simulate a sample of 10 movies drawn at random with replacement from the 200 movies. Using just this sample, do we have enough information to output `True` if the first sampled movie was released by the same studio as the last sampled movie? **(3 points)**

Example Answer: Yes, with "studio_proportions_only".

Note: Do not explain your answer for any of the options you've chosen; please follow the structure of the example answer provided.

Question 2(a) Can this be done using the `sample` function? If yes, what table would we call `sample` on? **(1 point)**

Yes, on "top_movies"

Question 2(b) Can this be done using the `np.random.choice` function? If yes, what array would we call `np.random.choice` on? **(1 point)**

Yes, on "studio_proportions_only"

Question 2(c) Can this be done using the `sample_proportions` function? If yes, what array would we call `sample_proportions` on? **(1 point)**

No

1.4 4. Assessing Jade's Models

Before you begin, Section 10.4 of the textbook is a useful reference for this part.

1.4.1 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 loses.

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. **(3 Points)**

[46]:
```
deck_model_probabilities = make_array(4 / 13, 9 / 13)
deck_model_probabilities
```

[46]:
```
array([ 0.30769231,  0.69230769])
```

[47]:
```
grader.check("q4_1")
```

[47]:
```
q4_1 results: All test cases passed!
```

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. **(3 Points)**

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

[48]:
```
statistic_choice = 1
statistic_choice
```

[48]:
```
1
```

[49]:
```
grader.check("q4_2")
```

[49]:
```
q4_2 results: All test cases passed!
```

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 cards under the model specified in `model_proportions`. **(5 Points)**

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

[50]:
```
def deck_simulation_and_statistic(sample_size, model_proportions):
    return sample_proportions(sample_size, model_proportions)[0] * 13


deck_simulation_and_statistic(13, deck_model_probabilities)
```

[50]:
```
4.0
```

[51]:
```
grader.check("q4_3")
```

[51]:
```
q4_3 results: All test cases passed!
```

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`. **(5 Points)**

[52]:
```
repetitions = 5000
deck_statistics = np.array(
```

```
    [
        deck_simulation_and_statistic(13, deck_model_probabilities)
        for _ in range(repetitions)
    ]
)

deck_statistics
```

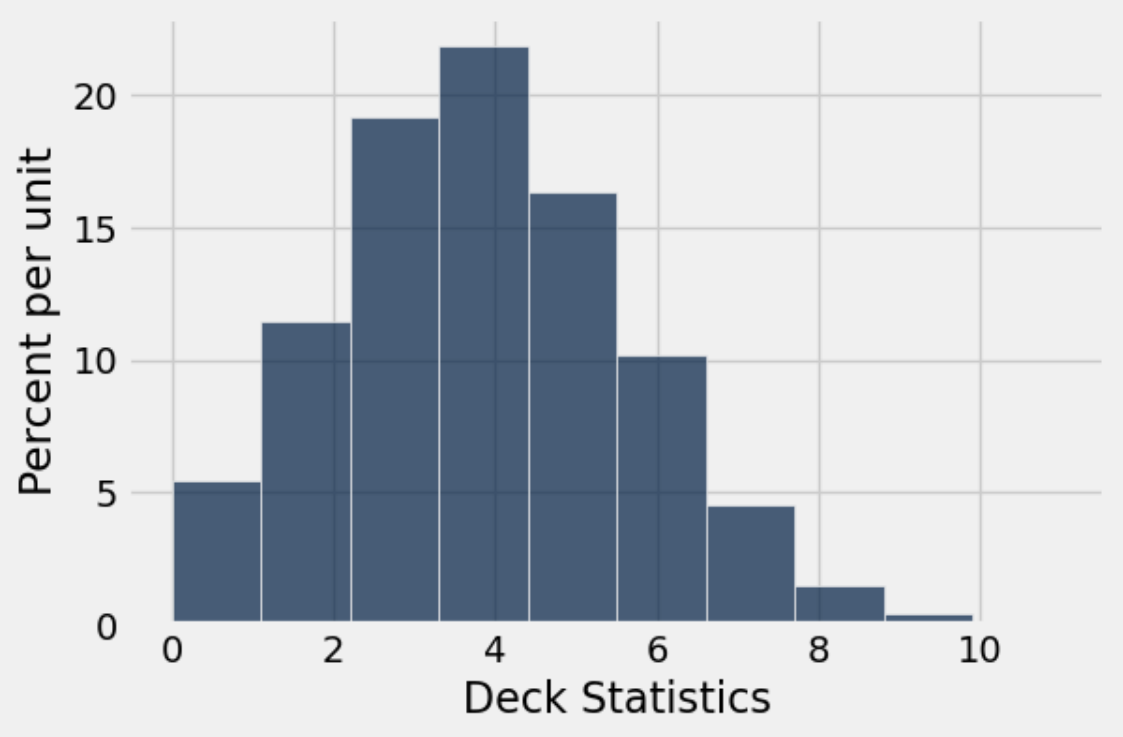
[52]:
```
array([ 4.,  6.,  9., …,  4.,  4.,  4.])
```

[53]:
```
grader.check("q4_4")
```

[53]:
```
q4_4 results: All test cases passed!
```

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

[54]:
```
# Draw a distribution of statistics
Table().with_column('Deck Statistics', deck_statistics).hist()
```



Question 5. Given your observed value, do you believe that Jade's model is reasonable, or is our alternative (that our deck is rigged) more likely? Explain your answer using the histogram of statistics simulated using Jade's model (produced above). **(4 Points)**

I believe it is more likely that our deck is rigged because, according to the statistics simulated using

Jade's model, the expected number of face cards drawn in 13 draws is approximately 4. Not only is the absolute difference between the expected and actual values large (4−8 =4), but the histogram also indicates that the probability of drawing more than 8 face cards in 13 draws is very low.

You're done with Homework 6!

Important submission steps: 1. Run the tests and verify that they all pass. 2. Choose **Save Notebook** from the **File** menu, then **run the final cell**. 3. Click the link to download the zip file. 4. Go to Gradescope and submit the zip file to the corresponding assignment. The name of this assignment is "HW 06 Autograder".

It is your responsibility to make sure your work is saved before running the last cell.

1.5 Pets of Data 8

Cookie says congrats on finishing HW6!

1.6 Submission

Below, you will see two cells. Running the first cell will automatically generate a PDF of all questions that need to be manually graded, and running the second cell will automatically generate a zip with your autograded answers. You are responsible for submitting both the coding portion (the zip) and the written portion (the PDF) to their respective Gradescope portals. **Please save before exporting!**

> **Important: You must correctly assign the pages of your PDF after you submit to the correct gradescope assignment. If your pages are not correctly assigned and/or not in the correct PDF format by the deadline, we reserve the right to award no points for your written work.**

If there are issues with automatically generating the PDF in the first cell, you can try downloading the notebook as a PDF by colicking on `File -> Save and Export Notebook As... -> PDF`. If that doesn't work either, you can manually take screenshots of your answers to the manually graded questions and submit those. Either way, **you are responsible for ensuring your submision follows our requirements, we will NOT be granting regrade requests for submissions that don't follow instructions.**

You must submit the PDF generated via one of these methods, we will not accept screenshots or Word documents.

1.7 Submission

Make sure you have run all cells in your notebook in order before running the cell below, so that all images/graphs appear in the output. The cell below will generate a zip file for you to submit. **Please save before exporting!**

[55]:
```
# Save your notebook first, then run this cell to export your submission.
grader.export(pdf=False, run_tests=True)
```

```
Running your submission against local test cases…
```

```
Your submission received the following results when run against available test
cases:

    q1_1 results: All test cases passed!

    q1_2 results: All test cases passed!

    q1_3 results: All test cases passed!

    q1_4 results: All test cases passed!

    q1_5 results: All test cases passed!

    q1_6 results: All test cases passed!

    q1_7 results: All test cases passed!

    q1_8 results: All test cases passed!

    q1_9 results: All test cases passed!

    q1_10 results: All test cases passed!

    q2_1 results: All test cases passed!

    q2_2 results: All test cases passed!

    q2_3 results: All test cases passed!

    q2_4 results: All test cases passed!

    q2_5 results: All test cases passed!

    q2_6 results: All test cases passed!

    q2_7 results: All test cases passed!

    q4_1 results: All test cases passed!

    q4_2 results: All test cases passed!

    q4_3 results: All test cases passed!

    q4_4 results: All test cases passed!
<IPython.core.display.HTML object>
```