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

Reading:

- Randomness (https://www.inferentialthinking.com/chapters/09/randomness.html)
- Sampling and Empirical Distributions (https://www.inferentialthinking.com/chapters/10/sampling-and-empirical-distributions.html)
- Testing Hypotheses (https://www.inferentialthinking.com/chapters/11/testing-hypotheses.html)

Please complete this notebook by filling in the cells provided. Before you begin, execute the following cell to load the provided tests. Each time you start your server, you will need to execute this cell again to load the tests.

Homework 6 is due Thursday, 3/5 at 11:59pm. You will receive an early submission bonus point if you turn in your final submission by Wednesday, 3/4 at 11:59pm. Start early so that you can come to office hours if you're stuck. Check the website for the office hours schedule. Late work will not be accepted as per the policies (http://data8.org/sp20/policies.html) of this course.

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.

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.

In [1]:
```
# 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 client.api.notebook import Notebook
ok = Notebook('hw06.ok')
_ = ok.auth(inline=True)
```

```
=====================================================================
Assignment: Homework 6: Probability, Simulation, Estimation, and Assess
ing Models
OK, version v1.14.19
=====================================================================
```

```
---------------------------------------------------------------------------
LoadingException                          Traceback (most recent call last)
<ipython-input-1-b837597f781c> in <module>
     13 
     14 from client.api.notebook import Notebook
---> 15 ok = Notebook('hw06.ok')
     16 _ = ok.auth(inline=True)

/opt/anaconda3/lib/python3.7/site-packages/client/api/notebook.py in __init__(self, filepath, cmd_args, debug, mode)
     13         ok_logger = logging.getLogger('client')   # Get top-level ok logger
     14         ok_logger.setLevel(logging.DEBUG if debug else logging.ERROR)
---> 15         self.assignment = load_assignment(filepath, cmd_args)
     16         # Attempt a login with enviornment based tokens
     17         login_with_env(self.assignment)

/opt/anaconda3/lib/python3.7/site-packages/client/api/assignment.py in load_assignment(filepath, cmd_args)
     22     if cmd_args is None:
     23         cmd_args = Settings()
---> 24     return Assignment(cmd_args, **config)
     25 
     26 def _get_config(config):

/opt/anaconda3/lib/python3.7/site-packages/client/sources/common/core.py in __call__(cls, *args, **kargs)
    185                 raise ex.SerializeException('__init__() missing expected '
    186                                 'argument {}'.format(attr))
--> 187         obj.post_instantiation()
    188         return obj
    189 

/opt/anaconda3/lib/python3.7/site-packages/client/api/assignment.py in post_instantiation(self)
    151     def post_instantiation(self):
    152         self._print_header()
--> 153         self._load_tests()
    154         self._load_protocols()
    155         self.specified_tests = self._resolve_specified_tests(

/opt/anaconda3/lib/python3.7/site-packages/client/api/assignment.py in _load_tests(self)
    205 
    206         if not self.test_map:
--> 207             raise ex.LoadingException('No tests loaded')
    208 
    209     def dump_tests(self):

LoadingException: No tests loaded
```

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.

```
BEGIN QUESTION
name: q1_1
manual: false
```

In [2]:
```
ten_sixes = 3 # SOLUTION
ten_sixes
```

Out[2]: 3

In [3]:
```
# TEST
ten_sixes in [1,2,3]
```

Out[3]: True

In [4]:
```
# HIDDEN TEST
ten_sixes == 3
```

Out[4]: True

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.

```
BEGIN QUESTION
name: q1_2
manual: false
```

In [5]:
```
five_or_less = 1 # SOLUTION
five_or_less
```

Out[5]: 1

In [6]:
```
# TEST
five_or_less in [1,2,3]
```

Out[6]: True

In [7]:
```
# HIDDEN TEST
five_or_less == 1
```

Out[7]: True

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.

```
BEGIN QUESTION
name: q1_3
manual: false
```

In [8]:
```
lottery = 1 # SOLUTION
```

In [9]:
```
# TEST
lottery in [1,2,3]
```

Out[9]: True

In [10]:
```
# HIDDEN TEST
lottery == 1
```

Out[10]: True

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?

```
BEGIN QUESTION
name: q1_4
manual: false
```

In [11]:
```
list_chances = 2 # SOLUTION
```

In [12]:
```
# TEST
list_chances in [1,2,3]
```

Out[12]: True

In [13]:
```
# HIDDEN TEST
list_chances == 2
```

Out[13]: True

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 (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.

```
BEGIN QUESTION
name: q2_1
manual: false
```

In [14]:
```
data_chance = (1/62)**5 #SOLUTION
data_chance
```

Out[14]: `1.0915447684774164e-09`

In [15]:
```
# TEST
round(data_chance, 11) == 1.09e-09
```

Out[15]: `True`

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).

```
BEGIN QUESTION
name: q2_2
manual: false
```

In [16]:
```
# We have provided the code below to compute a list called keys,
# containing all the lower-case English letters, upper-case English lett
ers, and the digits 0-9 (inclusive).  Print it if you
# want to verify what it contains.
import string
keys = list(string.ascii_lowercase + string.ascii_uppercase + string.dig
its)

def simulate_key_strike():
    """Simulates one random key strike."""
    return np.random.choice(keys) #SOLUTION

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

Out[16]: 'd'

In [17]:
```
# TEST
# It looks like you forgot to have your function return something.
simulate_key_strike() is not None
```

Out[17]: True

In [18]:
```
# TEST
import string
all([simulate_key_strike() in list(string.ascii_lowercase + string.ascii
_uppercase + string.digits) for i in range(100)])
```

Out[18]: True

In [19]:
```
# TEST
# It looks like you didn't use all the letters or numbers of the alphabe
t, or you
# used too many.
import numpy as np
np.random.seed(22)
62 >= len(np.unique([simulate_key_strike() for i in range(500)])) >= 45
```

Out[19]: True

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)`

```
BEGIN QUESTION
name: q2_3
manual: false
```

In [20]:
```
def simulate_several_key_strikes(num_strikes):
    # BEGIN SOLUTION
    """Simulates several random key strikes, returning them as a strin
g."""
    strikes = make_array()
    for i in np.arange(num_strikes):
        one_strike = simulate_key_strike()
        strikes = np.append(strikes, one_strike)
    return "".join(strikes)
    # END SOLUTION

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

Out[20]: 'sPH9Eho20Aa'

In [21]:
```
# TEST
len(simulate_several_key_strikes(15)) == 15
```

Out[21]: True

In [22]:
```
# TEST
# Make sure your function returns a string.
isinstance(simulate_several_key_strikes(15), str)
```

Out[22]: True

In [23]:
```
# TEST
# It looks like your simulation doesn't use all the letters,
# or it uses more than the 26 lower-case letters.
import numpy as np
np.random.seed(22)
62 >= len(np.unique(list(simulate_several_key_strikes(500)))) >= 45
```

Out[23]: True

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`.

```
BEGIN QUESTION
name: q2_4
manual: false
```

In [24]:
```
# BEGIN SOLUTION
num_simulations = 5000
num_dataeight = 0
for i in np.arange(num_simulations):
    if simulate_several_key_strikes(5) == 'Data8':
        num_datascience = num_dataeight + 1

data_proportion = num_dataeight / num_simulations
# END SOLUTION
data_proportion
```

Out[24]: 0.0

In [25]:
```
# TEST
data_proportion == 0
```

Out[25]: True

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?

```
BEGIN QUESTION
name: q2_5
manual: true
```

SOLUTION: No, it is not a good way to estimate it. The monkey types `"Data8"` very rarely - roughly 1 in a billion times. That usually won't happen even once in 5000 simulations, so our estimate will usually be 0. If it happened, our estimate would be at least .0002, which would also be inaccurate! So we would need many more simulations (at least a billion) to have any hope at a reasonable estimate. Algebra is more useful than a computer in this case.

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.

```
BEGIN QUESTION
name: q2_6
manual: false
```

In [26]:
```
t_chance = 1 - (61/62)**5 #SOLUTION
t_chance
```

Out[26]: 0.07808532616807251

In [27]:
```
# TEST
round(t_chance, 4) == .0781
```

Out[27]: True

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.)

```
BEGIN QUESTION
name: q2_7
manual: true
```

SOLUTION: Simulation would work better for estimating `t_chance`. The chance of typing ‘Data8’ was so small that we couldn’t expect the event to happen with only 5000 iterations. But since the probability of `t_chance` is actually around 1/12, it will show up in our simulation as often as it should under its theoretical probability.

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 (http://www.basketball-reference.com)](http://www.basketball-reference.com).

Run the next cell to load the two datasets.

In [28]:
```
player_data = Table.read_table('player_data.csv')
salary_data = Table.read_table('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.

```
BEGIN QUESTION
name: q3_1
manual: false
```

In [29]:
```
full_data = player_data.join('Player', salary_data, 'Name') #SOLUTION
full_data
```

Out[29]:

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)

In [30]:
```
# TEST
full_data.num_rows == 562
```

Out[30]: True

In [31]:
```
# TEST
# Double check the way you're combining the two tables. Are you combinin
g in the correct order
# (in terms of the arguments)? The problem statement saying "except 'Nam
e' column" is a hint
# at the order in which you should combine the tables.
list(full_data.labels)[0] == 'Player'
```

Out[31]: True

In [32]:
```
# TEST
full_data.select(sorted(full_data.labels)).sort(4).take(range(3))
```

Out[32]:

2P	3P	PTS	Player	Salary
0.6	0	1.7	Tyler Cook	50000
0	0	0	William Howard	50000
2	0	6	Eric Mika	50752

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:

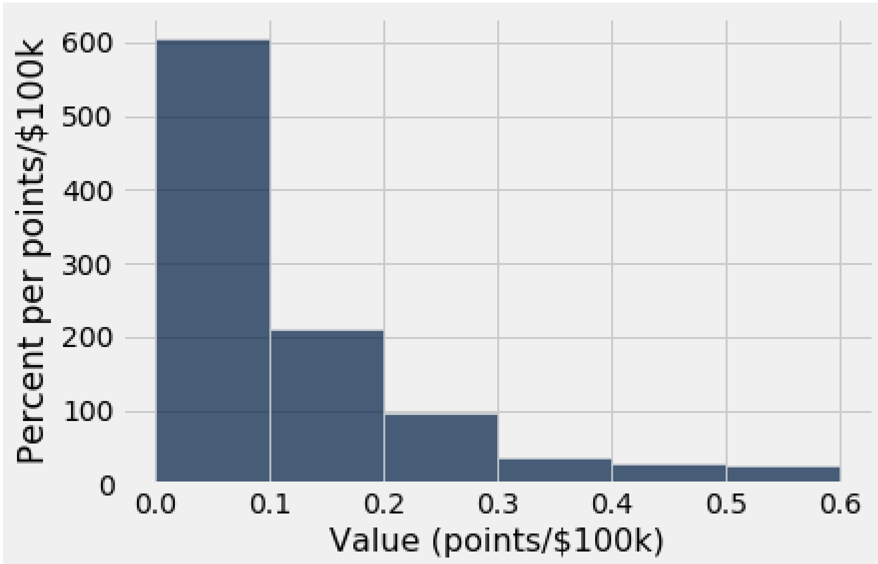
$$\frac{\text{"PTS"} - 2 * \text{"2P"}}{\text{"Salary"} \ / \ 100000}$$

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**

```
BEGIN QUESTION
name: q3_2
manual: true
```

In [33]:
```
bins = np.arange(0, 0.7, .1) # Use this provided bins when you make your histogram
full_data_with_value = full_data.with_column("Value", (full_data.column("PTS") - 2*full_data.column("2P")) / (full_data.column("Salary") / 100000)) #SOLUTION
full_data_with_value.hist("Value", bins=bins, unit="points/$100k") #SOLUTION
```



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.

In [34]:
```
sample_salary_data = Table.read_table("sample_salary_data.csv")
sample_salary_data.show(3)
```

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

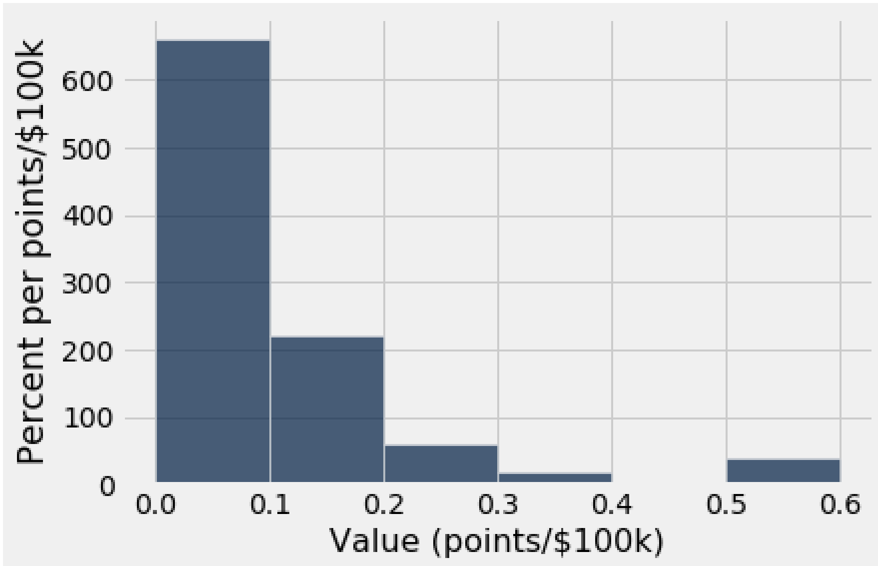
... (47 rows omitted)

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.

```
BEGIN QUESTION
name: q3_3
manual: true
```

In [35]:
```
sample_data = player_data.join('Player', sample_salary_data, 'Name')
sample_data_with_value = sample_data.with_column("Value", (sample_data.column("PTS") - 2*sample_data.column("2P")) / (sample_data.column("Salary") / 100000)) #SOLUTION
sample_data_with_value.hist("Value", bins=bins, unit="points/$100k") #SOLUTION
```



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 (https://www.inferentialthinking.com/chapters/10/1/empirical-distributions.html)](https://www.inferentialthinking.com/chapters/10/1/empirical-distributions.html)

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.

```
BEGIN QUESTION
name: q3_4
```

In [36]:
```
distribution_1 = "empirical" # SOLUTION
player_count_1 = 562 # SOLUTION
area_total_1 = 100 # SOLUTION

distribution_2 = "empirical" # SOLUTION
player_count_2 = 50 # SOLUTION
area_total_2 = 100 # SOLUTION
```

In [37]:
```
# TEST
distribution_1 in ['empirical', 'probability']
```

Out[37]: True

In [38]:
```
# TEST
distribution_2 in ['empirical', 'probability']
```

Out[38]: True

In [39]:
```
# TEST
type(player_count_1) == int and type(player_count_2) == int
```

Out[39]: True

In [40]:
```
# TEST
type(area_total_1) == int and type(area_total_2) == int
```

Out[40]: True

In [41]:
```
# HIDDEN TEST
distribution_1
```

Out[41]: 'empirical'

In [42]:
```
# HIDDEN TEST
distribution_2
```

Out[42]: 'empirical'

In [43]:
```
# HIDDEN TEST
player_count_1
```

Out[43]: 562

In [44]:
```
# HIDDEN TEST
player_count_2
```

Out[44]: 50

In [45]:
```
# HIDDEN TEST
area_total_1
```

Out[45]: 100

In [46]:
```
# HIDDEN TEST
area_total_2
```

Out[46]: 100

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.

```
BEGIN QUESTION
name: q3_5
manual: true
```

SOLUTION: The sample histogram and population histogram look similar for values below 0.3. For values above 0.3, the sample histogram looks less accurate. The players in the population with values above 0.3 are rarer, so the sample gives us a worse estimate of that part of the distribution.

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/ (https://earthquake.usgs.gov/earthquakes/search/))

In [47]:
```
earthquakes = Table().read_table('earthquakes_2019.csv').select(['time',
'mag', 'place'])
earthquakes
```

Out[47]:

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.

In [48]:
```
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]
```

Out[48]:
```
[6.458999999999999, 5.279000000000001]
```

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.

```
BEGIN QUESTION
name: q4_1
manual: true
```

SOLUTION: These samples are deterministic samples, not random samples, so we have no reason to believe they will represent the population or have a statistic close to the population parameter. Sample 1 is especially bad, because we are taking the mean of the highest-magnitude earthquakes. Sample 2 might represent the population a little bit better if earthquakes are randomly distributed through time and there is nothing particularly unique about December earthquakes, but only sampling December earthquake still has its own deterministic bias.

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.

```
BEGIN QUESTION
name: q4_2
manual: false
```

In [49]:
```
representative_sample = earthquakes.sample(200) #SOLUTION
representative_mean = np.mean(representative_sample.column('mag')) #SOLU
TION
representative_mean
```

Out[49]: 5.3229

In [50]:
```
# TEST
# The sample should be of size 200.
representative_sample.num_rows == 200
```

Out[50]: True

In [51]:
```
# TEST
# Your sample should have the same columns as the original table, and
# all data in the sample should be present in the original table.
all(np.in1d(representative_sample.column('mag'), earthquakes.column('ma
g')))
```

Out[51]: True

In [52]:
```
# TEST
# The mean can't be bigger than the biggest magnitude, or smaller than t
he smallest!
representative_mean < max(representative_sample.column('mag')) and repre
sentative_mean > min(representative_sample.column('mag'))
```

Out[52]: True

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`.

```
BEGIN QUESTION
name: q4_3
manual: false
```

In [53]:
```
maximums = make_array() #SOLUTION
for i in np.arange(5000):
    # BEGIN SOLUTION
    sample = earthquakes.sample(200)
    sample_max_magnitude = max(sample.column('mag'))
    maximums = np.append(maximums, sample_max_magnitude)
    # END SOLUTION
```

In [54]:
```
# TEST
# It looks like your maximums array is empty!
len(maximums) != 0
```

Out[54]: True

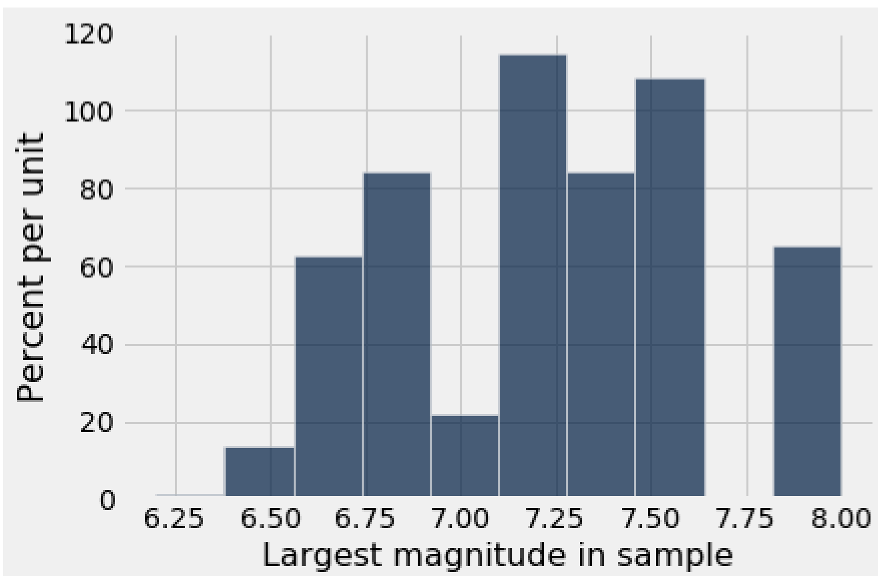
In [55]:
```
# TEST
len(maximums) == 5000
```

Out[55]: True

In [56]:
```
# TEST
# The biggest simulated maximum can't be bigger than the actual maximum!
max(maximums) <= max(earthquakes.column('mag'))
```

Out[56]: True

In [57]:
```
#Histogram of your maximums
Table().with_column('Largest magnitude in sample', maximums).hist('Large
st magnitude in sample')
```



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.

```
BEGIN QUESTION
name: q4_4
manual: false
```

In [58]:
```
strongest_earthquake_magnitude = max(earthquakes.column('mag')) #SOLUTIO
N
strongest_earthquake_magnitude
```

Out[58]: 8.0

In [59]:
```
# TEST
isinstance(strongest_earthquake_magnitude, float)
```

Out[59]: True

In [60]:
```
# HIDDEN TEST
np.isclose(strongest_earthquake_magnitude, 8.0)
```

Out[60]: True

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.

```
BEGIN QUESTION
name: q4_5
manual: true
```

SOLUTION: While we get pretty close to the actual max in the histogram, we can probably not get the actual maximum using a sample size of 200. One con of this approach is that our estimate will always be less than or equal to the actual maximum.

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.

```
BEGIN QUESTION
name: q5_1
manual: false
```

In [61]:
```
deck_model_probabilities = make_array(4/13, 9/13) #SOLUTION
deck_model_probabilities
```

Out[61]:
```
array([0.30769231, 0.69230769])
```

In [62]:
```
# TEST
# The array should have length 2
len(deck_model_probabilities) == 2
```

Out[62]:
```
True
```

In [63]:
```
# TEST
# The elements in the array should add up to 1.
sum(deck_model_probabilities) == 1
```

Out[63]:
```
True
```

In [64]:
```
# HIDDEN TEST
deck_model_probabilities.item(0) == 4/13
```

Out[64]:
```
True
```

In [65]:
```
# HIDDEN TEST
deck_model_probabilities.item(1) == 9/13
```

Out[65]:
```
True
```

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)

```
BEGIN QUESTION
name: q5_2
manual: false
```

In [66]:
```
statistic_choice = 1 #SOLUTION
statistic_choice
```

Out[66]: 1

In [67]:
```
# TEST
statistic_choice in [1,2,3]
```

Out[67]: True

In [68]:
```
# HIDDEN TEST
statistic_choice == 1
```

Out[68]: True

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` .

```
BEGIN QUESTION
name: q5_3
manual: false
```

In [69]:
```
def deck_simulation_and_statistic(sample_size, model_proportions):
    # BEGIN SOLUTION
    simulation = sample_proportions(sample_size, model_proportions)
    statistic = sample_size * simulation.item(0)
    return statistic
    # END SOLUTION

deck_simulation_and_statistic(13, deck_model_probabilities)
```

Out[69]: 4.0

In [70]:
```
# TEST
# The statistic should be between 0 and 13 face cards for
# a sample size of 13
num_face = deck_simulation_and_statistic(13, deck_model_probabilities)
0 <= num_face <= 13
```

Out[70]: True

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`.

```
BEGIN QUESTION
name: q5_4
manual: false
```

In [71]:
```
repetitions = 5000
# BEGIN SOLUTION
deck_statistics = make_array()

for i in np.arange(repetitions):
    one_deck_stat = deck_simulation_and_statistic(13, deck_model_probabi
lities)
    deck_statistics = np.append(deck_statistics, one_deck_stat)

# END SOLUTION

deck_statistics
```

Out[71]: array([4., 5., 4., ..., 5., 6., 4.])

In [72]:
```
# TEST
# There should be exactly as many elements in deck_statistics
# as the number 'repetitions'
len(deck_statistics) == repetitions
```

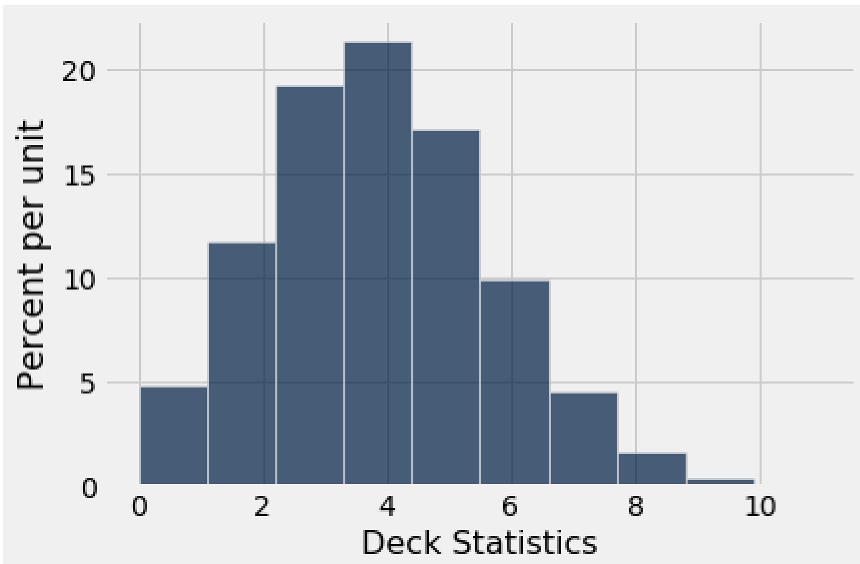
Out[72]: True

In [73]:
```
# TEST
# Each element of deck_statistics should be between 0
# and 13 inclusive
all([0 <= k <= 13 for k in deck_statistics])
```

Out[73]: True

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

In [74]:
```
#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 more likely? Explain your answer using the distribution drawn in the previous problem.

```
BEGIN QUESTION
name: q5_5
manual: true
```

SOLUTION:

No; given Jade's model, drawing 8 or more face cards happens around 2% of the time under simulation. This points us to think that our alternative, that the probability of drawing a face card is more than 4/13, is more likely.

6. Submission

Once you're finished, select "Save and Checkpoint" in the File menu and then execute the `submit` cell below. The result will contain a link that you can use to check that your assignment has been submitted successfully. If you submit more than once before the deadline, we will only grade your final submission. If you mistakenly submit the wrong one, you can head to [okpy.org (https://okpy.org/)](https://okpy.org/) and flag the correct version. To do so, go to the website, click on this assignment, and find the version you would like to have graded. There should be an option to flag that submission for grading!

In [75]:
```
_ = ok.submit()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-75-cc46ca874451> in <module>
----> 1 _ = ok.submit()

NameError: name 'ok' is not defined
```

In [76]:
```
# For your convenience, you can run this cell to run all the tests at once!
import os
print("Running all tests...")
_ = [ok.grade(q[:-3]) for q in os.listdir("tests") if q.startswith('q') and len(q) <= 10]
print("Finished running all tests.")
```

```
Running all tests...
Finished running all tests.
```