Homework 6: Hypothesis Testing

Please complete this notebook by filling in the cells provided. When you're done, follow the instructions in this short explainer video (https://www.youtube.com/watch?v=gMt_Rq43y_4&ab_channel=FahadKamran) to submit your homework.

If you cannot submit online, come to office hours for assistance. The office hours schedule appears on data8.org/fa16/weekly.html (http://data8.org/fa16/weekly.html).

This assignment is due Thursday, October 13 at 7PM (note the new, later time!). You will receive an early submission bonus point if you turn it in by Wednesday, October 12 at 7PM. Directly sharing answers is not okay, but discussing problems with course staff or with other students is encouraged.

Reading:

- Textbook chapter 10 (https://www.inferentialthinking.com/chapters/10/testing-hypotheses.html) and section 7.4 (https://www.inferentialthinking.com/chapters/07/4/joining-tables-by-columns.html)

Run the cell below to prepare the notebook and the tests. **Passing the automatic tests does not guarantee full credit on any question.** The tests are provided to help catch some common errors, but it is *your* responsibility to answer the questions correctly.

In [ ]:
```
# Run this cell to set up the notebook, but please don't change 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.assignment import load_assignment
tests = load_assignment('hw06.ok')
```

1. Catching Cheaters

Suppose you are a casino owner, and your casino runs a very simple game of chance. The customer names heads or tails, and then a dealer flips a coin. The customer wins $9 from the casino if they guess the right side and loses $10 if they're wrong.

Question 1

Assuming no one is cheating and the coin is fair, if a customer bets once on heads, what is the chance they win that bet? Over many bets on heads, what is the average amount of money the customer wins from a single bet? Does your answer change if they bet on tails instead?

Write your answer here, replacing this text.

A certain customer plays the game 20 times, betting heads each time, and wins 13 of the bets. You suspect that the customer is cheating! That is, you think that their chance of winning could be something other than the normal chance of winning.

You decide to test this using the outcomes of the 20 games you observed.

Question 2

Define the null hypothesis and alternative hypothesis for this investigation.

Null hypothesis: ...

Alternative hypothesis: ...

Question 3

Define a test statistic function named `test_statistic`. Your function should take as its argument the number of times the customer won. It should return the value of the test statistic for those data. You'll have to decide how to define the test statistic. **Then**, compute the value of your test statistic for the 20 games played by the customer we're investigating. Call this number `actual_test_statistic`.

In [ ]:
```
def test_statistic(num_wins):
    ...

actual_test_statistic = ...
actual_test_statistic
```

Question 4

Write a function called `simulate_under_null`. It should take no arguments. It should return the number of wins in 20 games simulated under the assumption that the null hypothesis is true. (That number is random.)

In [ ]:
```
def simulate_under_null():
    ...
```

In [8]:
```
_ = tests.grade('q1_4')
```

Question 5

Run your simulation 10,000 times and make a histogram of the test statistics. (Be sure to pick bins that accurately portray the distribution.)

In [ ]:
```
...
```

Question 6

Compute a P-value for this test by looking at the histogram. (Don't write code. It's okay if your answer is off by a bit.)

In [ ]:
```
p_value = ...
p_value
```

Question 7

Suppose you use a P-value cutoff of 1%, according to the arbitrary conventions of hypothesis testing. What do you conclude?

Write your answer here, replacing this text.

Question 8

Is `p_value` the probability that the customer cheated, or the probability that the customer didn't cheat, or neither?

Write your answer here, replacing this text.

Question 9

Is 1% (the P-value cutoff) the probability that the customer cheated, or the probability that the customer didn't cheat, or neither?

Write your answer here, replacing this text.

Question 10

Is either `p_value` or 1% the probability of seeing a test statistic as extreme or more extreme than this one if the null hypothesis were true?

Write your answer here, replacing this text.

Question 11

Suppose you run this test for 400 different customers after observing each customer play 20 games. When you reject the null hypothesis for a customer, you accuse that customer of cheating. If no customer were actually cheating, how many would you expect to accuse, on average (if any)? Explain your answer.

Write your answer here, replacing this text.

2. Pell Grants by State

The US National Center for Education Statistics compiles information about US colleges and universities in the Integrated Postsecondary Education Data System (IPEDS). Here's a spreadsheet (https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=0ahUKEwjMocT62pHOAhUJ1GMKHenBCccQF z7Bvv2fQSkKSHeYLtkBg) describing the tables in the IPEDS. The full datasets are available here (http://nces.ed.gov/ipeds/datacenter/DataFiles.aspx).

In this assignment, we'll use IPEDS data to compute the proportion of college students in each US state who receive Pell grants, which are a kind of financial aid. The data come from 2013.

The data we need are spread across two IPEDS tables, so we'll have to use `join` to bring them together.

First, run the cell below to load the IPEDS data. (We've pared down the data to just a few columns for this exercise, but the original datasets are quite rich.)

In [2]:
```
sfa = Table.read_table("sfa.csv")
hd = Table.read_table("hd.csv")

sfa.show(5)
hd.show(5)
```

We want to compute the proportion of students in each *state* who receive Pell grants. Right now we know:

- how many students are at each *school* (the `sfa` table);
- how many students received Pell grants at each school (also the `sfa` table); and
- what state each school is in (the `hd` table).

Let's work backward. If we know the total number of students in each state and the total number of Pell grant recipients in each state, we can compute the proportions. If we know how many students and Pell grant recipients were at each school, and we know what state each school is in, then we can group by state to compute the total number of students and Pell grant recipients per state.

That means we first need to compile the state, student, and Pell grant recipient information for each school into a single table.

To match data across tables, each school is assigned a unique identifier in the column named "Institution ID".

Question 1

Make a table called `with_state` that includes one row for each school that's present in *both* `sfa` and `hd`. Each row should have the school's ID, its number of undergraduate students, its number of Pell grant recipients, and its state. (It's okay if it has other columns besides those four.) Use the same names for those columns as the corresponding columns in the original data tables.

In [3]:
```
with_state = ...
with_state
```

In [4]:
```
_ = tests.grade('q2_1')
```

Question 2

Make a table called `students_and_grants_by_state` that has the total number of undergraduates and Pell grant recipients in each *state*. Use the same names for those columns as the corresponding columns in the original data tables.

In [6]:
```
students_and_grants_by_state = ...
students_and_grants_by_state
```

In [7]:
```
_ = tests.grade('q2_2')
```

Question 3

Create a table called `pell_proportions` with two columns: "State (abbreviated)" is the name of each state, and "Pell proportion" is the proportion of students in each state who receive Pell grants.

```
In [ ]: pell_proportions = ...
        pell_proportions
```

```
In [8]: _ = tests.grade('q2_3')
```

Question 4

Make a histogram of Pell grant proportions.

```
In [12]: ...
```

The file `us.json` contains data about the boundaries of each US state. We have loaded it as a map by calling `Map.read_geojson`.

Note: The data come from NOAA (http://www.nws.noaa.gov/geodata/catalog/national/html/us_state.htm). We converted them to this format using a tool called `ogr2ogr` (see here (http://ben.balter.com/2013/06/26/how-to-convert-shapefiles-to-geojson-for-use-on-github/)) and used mapshaper (http://www.mapshaper.org) to reduce the resolution to 1% of the original.

The Map method `color` colors regions in a map according to numbers you specify. It takes 2 arguments:

1. A table whose first column names each region, and whose second column gives the intensity of the color you want for that region.
2. The named argument `key_on=...`, where the argument itself is a string that tells `color` how you're identifying each region. In this case, both `pell_proportions` and the map know about each state's abbreviation, so use `key_on="feature.properties.STATE"`. (This part looks a little bit magical, which is just a flaw in the design of the `datascience` library. If it confuses you, don't worry too much.)

Question 5

Create a map of the US, with each state colored according to the proportion of undergraduates in that state who are Pell grant recipients, with higher-proportion states getting higher-intensity colors.

```
In [ ]: us_map = Map.read_geojson('us.json')
        ...
```

Question 6

Describe any pattern you see in the data. Be sure to mention any parts of the data that *don't* fit the pattern you describe.

Write your answer here, replacing this text.

3. Testing Dice

Students in a Data Science class want to figure out whether a six-sided die is fair or not. On a fair die, each face of the die appears with chance 1/6 on each roll, regardless of the results of other rolls. Otherwise, a die is called unfair. We can describe a die by the probability of landing on each face. For example, this table describes a die that is unfairly weighted toward 1:

Face	Probability
1	.5
2	.1
3	.1
4	.1
5	.1
6	.1

For this exercise, you can use the function `proportion_from_distribution` defined in lecture and the textbook. We've defined it in the setup cell above.

Question 1

Define a null hypothesis and an alternative hypothesis for this question.

Null hypothesis: ...

Alternative hypothesis: ...

We decide to test the die by rolling it 5 times. The proportions of the 6 faces in these 5 rolls are stored in a table with 6 rows. For example, here is the table we'd make if the die rolls ended up being 1, 2, 3, 3, and 5:

Face	Proportion
1	.2
2	.2
3	.4
4	.0
5	.2
6	.0

The function `mystery_test_statistic`, defined below, takes a single table like this as its argument and returns a number (which we will use as a test statistic).

In [2]:
```
# Note: We've intentionally used obscurantist function and
# variable names to avoid giving away answers.  It's rarely
# a good idea to use names like "x" in your code.

def mystery_test_statistic(sample):
    x = sum(sample.column("Face")*sample.column("Proportion"))
    y = np.mean(np.arange(1, 6+1, 1))
    return abs(x - y)
```

Question 2

Describe in English what the test statistic is. Is it equivalent to the total variation distance between the observed face distribution and the fair die distribution?

Write your answer here, replacing this text.

The function `simulate_observations_and_test` takes as its argument a table describing the probability distribution of a die. It simulates one set of 5 rolls of that die, then tests the null hypothesis about that die using our test statistic function above. It returns `False` if it *rejects* the null hypothesis about the die, and `True` otherwise.

In [4]:
```
# The probability distribution table for a fair die:
fair_die = Table().with_columns(
        "Face", np.arange(1, 6+1),
        "Probability", [1/6, 1/6, 1/6, 1/6, 1/6, 1/6])

def simulate_observations_and_test(actual_die):
    """Simulates die rolls from actual_die and tests the hypothesis that
 the die is fair.

    Returns True if that hypothesis is accepted, and False otherwise."""
    sample_size = 5
    p_value_cutoff = .2
    num_simulations = 250

    observation_set = proportions_from_distribution(actual_die, "Probabi
lity", sample_size)
    actual_statistic =
mystery_test_statistic(observation_set.relabeled("Random Sample", "Propo
rtion"))

    simulated_statistics = make_array()
    for _ in np.arange(num_simulations):
        one_observation_set_under_null = proportions_from_distribution(f
air_die, "Probability", sample_size)
        simulated_statistic = mystery_test_statistic(one_observation_set
_under_null.relabeled("Random Sample", "Proportion"))
        simulated_statistics = np.append(simulated_statistics, simulated
_statistic)
    p_value = np.count_nonzero(actual_statistic < simulated_statistics)
/ num_simulations

    return p_value >= p_value_cutoff

# Calling the function to simulate a test of a fair die:
simulate_observations_and_test(fair_die)
```

Question 3

By examining `simulate_observations_and_test`, compute the probability that it returns `False` when its argument is `fair_die` (which is defined above the function). You can call the function a few times to see what it does, but **don't** perform a simulation to compute this probability. Use your knowledge of hypothesis tests.

In [2]:
```
probability_of_false = ...
```

Question 4

From the perspective of someone who wants to know the truth about the die, is it good or bad for the function to return `False` when its argument is `fair_die`?

Write your answer here, replacing this text.

Question 5

Verify your answer to question 3 by simulation, computing an approximate probability that `simulation_observations_and_test` returns `False`.

Note: This will be a little slow. 300 repetitions of the simulation should suffice to get an answer that's roughly correct and should require a minute or so of computation.

In [4]:
```
num_test_simulations = 300

# For reference, the staff solution involved 6 lines of code before thi
s.
...
approximate_probability_of_false = ...
approximate_probability_of_false
```

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