Homework 6: Hypothesis Testing

Please complete this notebook by filling in the cells provided. When you're done:

1. Select `Run All` from the `Cell` menu to ensure that you have executed all cells.
2. Select `Download as HTML (.html)` from the `File` menu
3. Inspect your file to make sure it looks like it should.
4. Upload your file to Moodle.

In [1]:
```
# 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)
```

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 USD from the casino if they guess the right side and loses 10 USD 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?

If a customer bets once on heads, the chance they win the bet is 1/2. Since the probability to get the heads from a single bet is 1/2, the average chance of winning is 1/2 which has the same number of wins and loses. The customer gets 9 dollars for winning and loses 10 dollars for losing. Therefore, if we assume the customer to have the same number of wins and loses, they can getthe average -0.5 USD from a single bet. since there is only two choices with equal chance, they will get the same average amount of money even when they bet on tails instead of the heads.

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 = the customer did not cheat. the probability to win or lose are both 1/2.

alternative hypothesis = there is a chance for the customer to cheat.

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 [2]:
```
def test_statistic(num_wins):
    '''take the number of times the customer won
    and return the value of the test statistic for the data'''
    observed_statistic = abs ((num_wins / 20) - 0.5)
    return observed_statistic

## the customer won 13 times for 20 games
actual_test_statistic = test_statistic(13)
actual_test_statistic
```

Out[2]:
```
0.15000000000000002
```

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 [3]:
```
def simulate_under_null():
    '''return the number of wins in 20 games assuming that the null hypothes
    proportions = sample_proportions(20, make_array(.5,.5))
    number_of_wins = 20*proportions.item(0)
    return int(number_of_wins)

simulate_under_null()
```

Out[3]:
```
11
```

Question 5

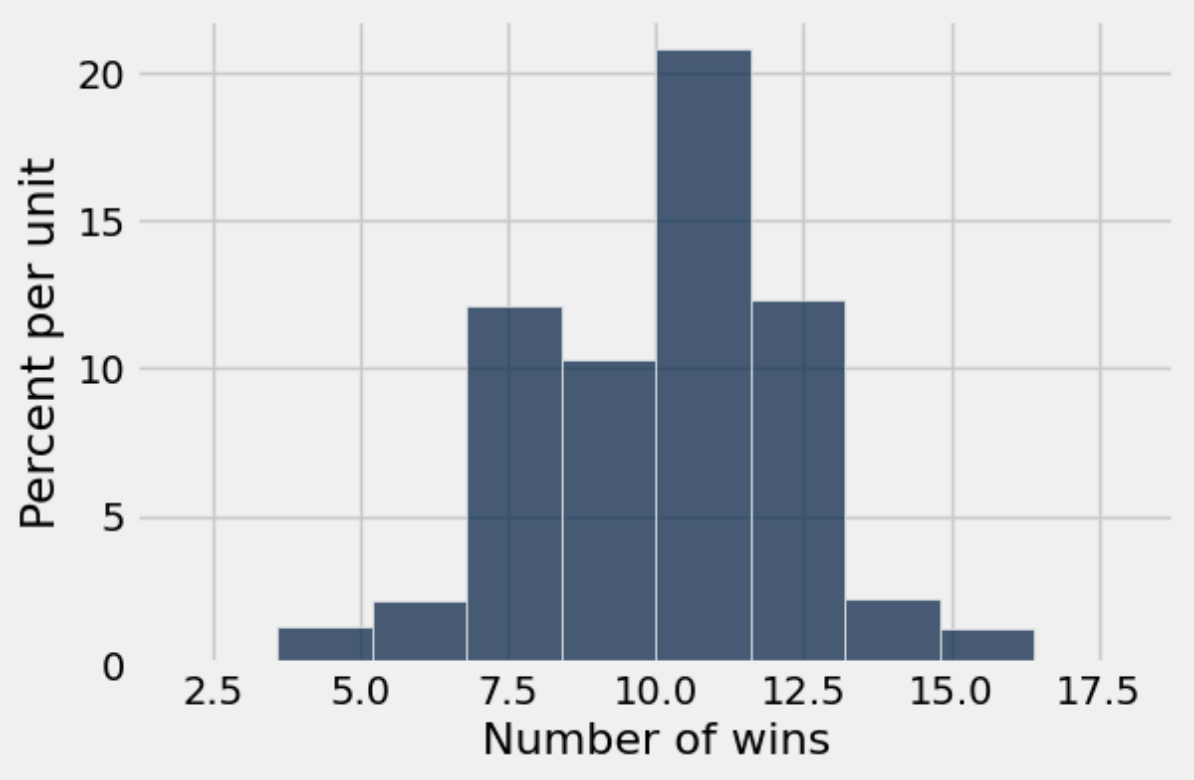
Run your simulation 10,000 times and make a histogram of the test statistics. (Be sure to pick bins that accurately portray the distribution.)

In [4]:
```
sampled_stats = make_array()
repetition = 10000

for i in np.arange(repetition):
    game_simulation = simulate_under_null()
    sampled_stats = np.append(sampled_stats, game_simulation)
```

```
sampled_stats

null_dist = Table().with_column('Number of wins', sampled_stats)
null_dist.hist()
```



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 [6]:
```
p_value = 0.13
p_value
```

Out[6]:
```
0.13
```

Question 7

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

Since the p-value is larger than 0.01, we accept the null hypothesis. The probability for the wins is based on randomness.

Question 8

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

Neither. A p-value is not the probability that the null or alternative hypothesis is correct. It is the absolute difference of test statistics and the observed test statistics.

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?

Neither. The p-value cutoff means that when no difference exists, such an extreme value for the test statistic is expected less than 1% of the time. Therefore, 1% is neither the probability that the customer cheated or not.

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?

p-value

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.

If we have the p-value cutoff of 1%, we would expect to accuse 4 people for cheating, because there is a 1% chance of getting such value which we incorrectly accuse the customers of cheating under the null hypothesis.

2. Pell Grants by State

THIS SECTION IS OPTIONAL FOR EXTRA PRACTICE AND A LITTLE BIT OF EXTRA CREDIT

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 describing the tables in the IPEDS. The full datasets are available here.

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 [7]:
```
sfa = Table.read_table("/srv/data/DS_113_S23/HW/HW_6/sfa.csv")
hd = Table.read_table("/srv/data/DS_113_S23/HW/HW_6/hd.csv")

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

Institution ID	Number of Pell grant recipients	Number of undergraduates
100654	2967	4170
100663	3958	11291
100690	225	329
100706	1930	5882
100724	4240	5130

... (7151 rows omitted)

Institution ID	Name	Address	City	State (abbreviated)	ZIP
100654	Alabama A & M University	4900 Meridian Street	Normal	AL	35762
100663	University of Alabama at Birmingham	Administration Bldg Suite 1070	Birmingham	AL	35294-0110
100690	Amridge University	1200 Taylor Rd	Montgomery	AL	36117-3553
100706	University of Alabama in Huntsville	301 Sparkman Dr	Huntsville	AL	35899
100724	Alabama State University	915 S Jackson Street	Montgomery	AL	36104-0271

... (7759 rows omitted)

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 [8]:
```
## join the two tables
with_state = sfa.join("Institution ID", hd).drop(3,4,5,7)
with_state
```

Out[8]:

Institution ID	Number of Pell grant recipients	Number of undergraduates	State (abbreviated)
100654	2967	4170	AL
100663	3958	11291	AL
100690	225	329	AL
100706	1930	5882	AL
100724	4240	5130	AL
100751	5905	28026	AL
100760	1316	2020	AL
100812	1402	3414	AL
100830	1698	4239	AL
100858	3403	20175	AL

... (7146 rows omitted)

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 [9]:
```
students_and_grants_by_state = with_state.group("State (abbreviated)", np.su
students_and_grants_by_state
```

Out[9]:

State (abbreviated)	Number of Pell grant recipients sum	Number of undergraduates sum
AK	8318	31522
AL	118944	269650
AR	75434	164062
AS	914	1795
AZ	304393	648368
CA	893974	2500929
CO	124044	321048
CT	65908	185443
DC	15025	49786
DE	16392	51521

... (49 rows omitted)

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 [10]:
```
pell_proportions = Table().with_columns("State (abbreviated)", students_and_
                                        "Pell proportion", students_and_grant
pell_proportions
```

Out[10]:

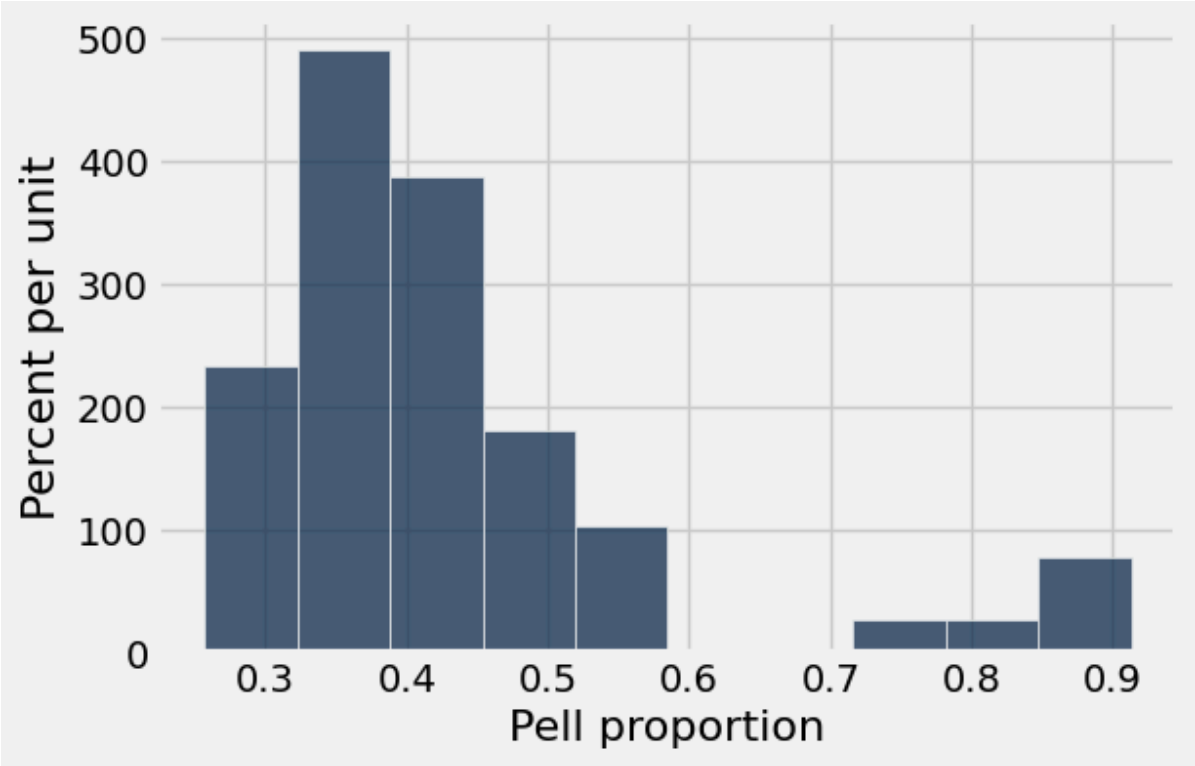
State (abbreviated)	Pell proportion
AK	0.263879
AL	0.441105
AR	0.45979
AS	0.509192
AZ	0.469476
CA	0.357457
CO	0.386372
CT	0.355408
DC	0.301792
DE	0.318162

... (49 rows omitted)

Question 4

Make a histogram of Pell grant proportions.

In [11]:
```
pell_proportions.hist("Pell proportion")
```



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.

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 [12]:
```
us_map = Map.read_geojson('/srv/data/DS_113_S23/HW/HW_6/us.json')
us_map.color(pell_proportions, key_on="feature.properties.STATE")
```

Out[12]:
Make this Notebook Trusted to load map: File -> Trust Notebook

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.

Usually Southern West states have higher proportion of undergraduates in that state who are Pell grant recipients. Among the states, Mississippi, Georgia, and Florida have

the highest proportion than the others and the states closer to them have higher porportion than the farther states. However, Oregon, Montana, and Idaho have higher proportion even when they are far away from Mississippi, Georgia, and Florida.

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:

In [13]:
```
Table().with_columns(
    "Face", np.arange(1,7),
    "Probability", make_array(.5,.1,.1,.1,.1,.1)
)
```

Out[13]:

Face	Probability
1	0.5
2	0.1
3	0.1
4	0.1
5	0.1
6	0.1

For this exercise, you can use the function `proportion_from_distribution` defined in lecture and the textbook. It is like .sample() or np.random.choice except that the items do not have the same chance of being selected. You must define the probabilities for different elements.

Question 1

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

Null hypothesis: The probability for each face of the die to appear is 1/6. p1=p2=p3=p4=p5=p6

Alternative hypothesis: Each face of the die appears with different chances on each roll, which is an unfair die.

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:

In [14]:
```
Table().with_columns(
    "Face", np.arange(1,7),
    "Proportion", make_array(.2,.2,.4,0,.2,0)
)
```

Out[14]:

Face	Proportion
1	0.2
2	0.2
3	0.4
4	0
5	0.2
6	0

The function `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 [15]:
```
def test_statistic(sample):
    avg_roll = sum(sample.column("Proportion") * sample.column("Face"))
    avg_for_fair_die = np.mean(np.arange(1, 6+1, 1))
    return abs(avg_roll - avg_for_fair_die)
```

Question 2

Describe in English what the test statistic is.

Test statistic describes how far the observed data is from the null hypothesis. Small values of this statistic favor the null hypothesis, and large values favor the alternative.

The code below takes five rolls from a fair die and then uses them as observations to conduct a simulation hypothesis test where the null hypothesis is that the rolls came from a fair die (which they did). When you run the code, it tells you the conclusion for the hypothesis test: true (the die is believed to be fair) or false (the die is believed to be unfair).

Please read the code and do your best to figure out how it works.

In [16]:
```
# 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

    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, "Probability
    actual_statistic = test_statistic(observation_set.relabeled("Random Samp

    simulated_statistics = make_array()
    for _ in np.arange(num_simulations):
        one_observation_set_under_null = proportions_from_distribution(fair_
        simulated_statistic = test_statistic(one_observation_set_under_null.
        simulated_statistics = np.append(simulated_statistics, simulated_sta
    p_value = np.count_nonzero(actual_statistic < simulated_statistics) / nu
```

```
    return p_value >= p_value_cutoff

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

Out[16]: True

Question 3

Since the function takes its observation from a fair die, how is it possible that it would then reject the null hypothesis that the rolls came from a fair die? What would this mean?

Although the chances for each faces of the die to appear are the same, since the observation is based on randomness, it is possible to reject the null hypothesis that the rolls came from a fair die.

Question 4

By examining the variable settings at the beginning of the function `simulate_observations_and_test`, compute the probability that it returns `False` when it is run on `fair_die`. 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 [17]:
```
probability_of_false = 0.2
```

Question 5

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`?

The probabiloty to return False when the argument is fair_die is 20%, which is a lot for a fair die to have. From the perspective of a person who wants to know the truth about the die, it would be good for the function to return False, because if then, the person would know the fair_die is not always have the same chance to get each faces of the die.

Question 6

Verify your answer to question 4 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 [18]:
```
num_test_simulations = 300
sampled_stats = make_array()

for i in np.arange(num_test_simulations):
    die_simulation = simulate_observations_and_test(fair_die)
    sampled_stats = np.append(sampled_stats, die_simulation)
```

```
approximate_probability_of_false = 1-(np.count_nonzero(sampled_stats)/300)
approximate_probability_of_false
```

Out[18]: 0.20666666666666667