Notebook

June 8, 2021

[ ]:
```
# Initialize Otter
import otter
grader = otter.Notebook("PS88_lab_week3.ipynb")
```

1 PS 88 Week 3 Lab: Simulations and Pivotal Voters

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from datascience import Table
from ipywidgets import interact
%matplotlib inline
```

1.1 Part 1: Plotting expected utility

We can use Python to do expected utility calculations and explore the relationship between parameters in decision models and optimal choices.

In class we showed that the expected utility for voting for a preferred candidate can be written $p_1 b - c$. A nice way to do calculations like this is to first assign values to the variables:

[2]:
```
p1=.6
b=100
c=2
p1*b-c
```

[2]: 58.0

Question 1.1. Write code to compute the expected utility to voting when $p_1 = .5$, $b = 50$, and $c = .5$

[3]:
```
#Answer to 1.1 here
p1=.5
b=50
c=.5
p1*b-c
```

[3]: 24.5

We don't necessarily care about these expected utilities on their own, but how they compare to the expected utility to abstaining.

Question 1.2. If $b = 50$ and $p_0 = .48$, write code to compute the expected utility to abstaining.

[4]:
```
# Code for 1.2 here
p0=.48
b=50
p0*b
```

[4]: 24.0

Question 1.3. Given 1.1 and 1.2, is voting the expected utility maximizing choice given given these parameters?

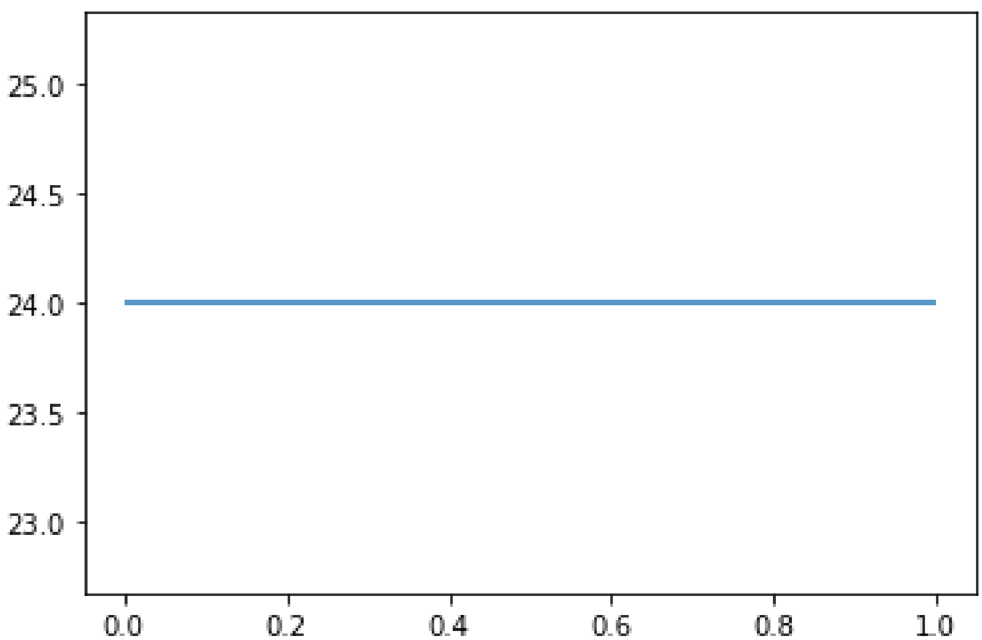
Answer to 1.3 here

We can also use the graphic capabilities of Python to learn more about how these models work.

[ ]:
```

```

[5]:
```
plt.hlines(p0*b, 0,1, label='Abstaining Utility')
```

[5]: <matplotlib.collections.LineCollection at 0x7f13d079a2e0>



```
[6]: import matplotlib.pyplot as plt
     import numpy as np
     from ipywidgets import interact, IntSlider, FloatSlider
     def plotEU(b):
         plt.hlines(p0*b, 0,2, label='Abstaining Utility')
         c = np.arange(0,2, step=.01)
         y = p1*b-c
         plt.ticklabel_format(style='plain')
         plt.xticks(rotation=45)
         plt.plot(c,y, label='Voting Expected Utility')
     #    plt.vlines(no_lobbying+1e7-1e5*p, -2e7, 0, linestyles="dashed")
         plt.xlabel('Voting Cost')
         plt.ylabel('Expected Utility')
         plt.legend()
     interact(plotEU, b=IntSlider(min=0,max=300, value=100))
```

```
interactive(children=(IntSlider(value=100, description='b', max=300), Output()),␣
 ↪_dom_classes=('widget-interac…
```

```
[6]: <function __main__.plotEU(b)>
```

1.2 Part 2: Simulating votes

How can we estimate the probability of a vote mattering? One route is to use probability theory, which in realistic settings (like the electoral college in the US) requires lots of complicated mathematical manipulation. Another way, which will often be faster and uses the tools you are learning in Data 8, is to run simulations.

As we will see throughout the class, simulation is an incredibly powerful tool that can be used for many purposes. For example, later in the class we will use simulation to see how different causal processes can produce similar data.

For now, we are going to use simulation to simulate the probability a vote matters. The general idea is simple. We will create a large number of "fake electorates" with parameters and randomness that we control, and then see how often an individual vote matters in these simulations.

Before we get to voting, let's do a simple exercise as warmup. Suppose we want to simulate flipping a coin 10 times. To do this we can use the `random.binomial` function from `numpy` (imported above as `np`). This function takes two arguments: the number of flips (`n`) and the probability that a flip is "heads" (`p`). More generally, we often call n the number of "trials" and p the probability of "success".

The following line of code simulates flipping a "fair" (i.e., $p = .5$) coin 10 times. Run it a few times.

```
[7]: # First number argument is the number of times to flip, the second is the␣
      ↪probability of a "heads"
     np.random.binomial(n=10, p=.5)
```

```
[7]: 6
```

We can simulate 100 coin flips at a time by changing the `n` argument to 100:

[8]:
```
np.random.binomial(n=100, p=.5)
```

[8]: 55

In the 2020 election, about 158.4 million people voted. This is a big number to have to keep typing, so let's define a variable:

[9]:
```
voters2020 = 158400000
```

Question 2a. Write a line of code to simulate 158.4 million people flipping a coin and counting how many heads there are.

[11]:
```
# Code for 2a here
sim = np.random.binomial(n=voters2020, p=.5) #SOLUTION
sim
```

[11]: 79204833

[ ]:
```
grader.check("q2a")
```

Of course, we don't care about coin flipping per se, but we can think about this as the number of "yes" votes if we have n people who vote for a candidate with probability p. In the 2020 election, about 51.3% of the voters voted fro Joe Biden. Let's do a simulated version of the election: by running `np.random.binomial` with 160 million trials and a probability of "success" of 51.3%.

Question 2b. Write code for this trial

[15]:
```
# Code for 2b
joe_count = np.random.binomial(n=voters2020, p=.513) #SOLUTION
joe_count
```

[15]: 81254368

[ ]:
```
grader.check("q2b")
```

In reality, Biden won 81.3 million votes.

Question 2c. How close was your answer to the real election? Compare this to the cases where you flipped 10 coins at a time.

Type your answer here, replacing this text.

SOLUTION: Answer to 2c here

1.3 Part 3. Pivotal votes.

Suppose that you are a voter in a population with 10 people who are equally likely to vote for candidate A or candidate B, and you prefer candidate A. If you turn out to vote, you will be pivotal if the other 10 are split evenly between the two candidates. How often will this happen?

We can answer this question by running a whole bunch of simulations where we effectively flip 10 coins and count how many heads there are.

The following line runs the code to do 10 coin flips with `p=5` 10,000 times, and stores the results in an array.(Don't worry about the details here: we will cover how to write "loops" like this later.)

[19]:
```
ntrials=10000
trials10 = [np.random.binomial(n=10, p=.5) for _ in range(ntrials)]
```

Here is the ouput:

[20]:
```
trials10
```

[20]:
```
[6,
 5,
 5,
 7,
 6,
 6,
 6,
 4,
 4,
 5,
 7,
 1,
 3,
 7,
 5,
 7,
 4,
 5,
 5,
 3,
 6,
 4,
 6,
 2,
 3,
 5,
 4,
 8,
 7,
 7,
 2,
 6,
 7,
 6,
 4,
```

```
5,
8,
4,
5,
5,
5,
4,
7,
5,
4,
2,
5,
3,
6,
6,
7,
3,
2,
4,
5,
6,
4,
4,
4,
3,
4,
7,
3,
3,
2,
6,
9,
3,
6,
4,
4,
7,
8,
6,
5,
5,
6,
2,
3,
6,
7,
8,
```

```
5,
4,
5,
4,
3,
6,
4,
2,
6,
8,
7,
4,
7,
5,
7,
5,
2,
4,
3,
4,
4,
6,
7,
4,
6,
3,
3,
6,
4,
5,
9,
5,
7,
4,
4,
3,
5,
8,
5,
2,
6,
2,
4,
8,
3,
4,
1,
```

```
6,
2,
4,
2,
3,
8,
7,
6,
8,
5,
3,
6,
6,
6,
8,
6,
6,
4,
6,
4,
1,
8,
4,
7,
5,
7,
5,
7,
5,
8,
7,
7,
8,
6,
4,
4,
3,
3,
1,
3,
5,
8,
5,
5,
5,
5,
5,
```

```
5,
3,
4,
6,
2,
6,
4,
5,
5,
5,
6,
6,
4,
6,
5,
3,
4,
5,
6,
5,
4,
4,
5,
6,
4,
4,
6,
6,
2,
5,
4,
5,
3,
5,
6,
4,
6,
4,
5,
5,
5,
8,
5,
3,
6,
3,
7,
```

```
2,
3,
7,
3,
6,
5,
5,
5,
4,
3,
4,
6,
6,
6,
7,
6,
7,
3,
8,
5,
7,
2,
6,
5,
4,
4,
2,
7,
4,
7,
7,
6,
7,
3,
3,
2,
4,
2,
7,
5,
3,
4,
7,
8,
7,
7,
6,
```

```
4,
5,
3,
6,
2,
5,
3,
6,
5,
7,
6,
2,
3,
5,
5,
5,
8,
7,
6,
6,
3,
3,
9,
7,
5,
2,
7,
4,
4,
7,
7,
2,
6,
5,
5,
6,
5,
5,
8,
3,
8,
3,
6,
3,
5,
5,
4,
```

```
4,
8,
6,
4,
2,
4,
3,
6,
8,
5,
5,
5,
6,
6,
4,
3,
6,
5,
3,
7,
7,
6,
5,
3,
3,
8,
7,
6,
5,
3,
5,
8,
6,
8,
6,
3,
5,
5,
8,
6,
4,
0,
6,
3,
6,
6,
8,
```

```
8,
5,
3,
6,
6,
3,
7,
3,
5,
5,
1,
5,
2,
3,
5,
5,
6,
7,
7,
4,
4,
6,
6,
5,
6,
7,
5,
5,
4,
6,
6,
5,
5,
2,
7,
3,
7,
2,
6,
6,
5,
5,
5,
5,
3,
7,
7,
```

```
7,
6,
7,
5,
6,
5,
3,
4,
4,
6,
6,
7,
5,
5,
7,
8,
7,
4,
5,
4,
5,
6,
6,
4,
2,
4,
4,
3,
3,
5,
5,
6,
4,
4,
4,
4,
6,
5,
4,
3,
2,
5,
4,
2,
6,
5,
6,
```

```
5,
5,
5,
5,
4,
4,
5,
7,
4,
7,
7,
4,
7,
6,
4,
6,
5,
4,
4,
5,
4,
7,
3,
4,
6,
8,
3,
3,
6,
6,
5,
6,
5,
6,
6,
5,
3,
3,
3,
7,
6,
6,
5,
4,
6,
4,
5,
```

```
6,
8,
5,
6,
5,
4,
4,
6,
5,
3,
5,
7,
7,
5,
3,
5,
6,
6,
5,
7,
7,
5,
1,
4,
3,
8,
6,
6,
7,
4,
4,
3,
6,
5,
6,
7,
8,
3,
7,
7,
9,
6,
6,
3,
5,
5,
4,
```

```
6,
2,
4,
6,
4,
6,
8,
5,
6,
4,
6,
6,
4,
6,
6,
5,
4,
6,
5,
6,
0,
7,
7,
5,
6,
4,
5,
6,
3,
5,
5,
5,
4,
5,
3,
6,
6,
3,
6,
6,
5,
5,
6,
5,
5,
1,
4,
```

```
5,
6,
9,
6,
5,
6,
5,
7,
2,
5,
2,
4,
4,
3,
6,
6,
5,
2,
8,
3,
4,
4,
7,
3,
6,
6,
3,
5,
5,
4,
7,
5,
6,
6,
4,
3,
8,
5,
5,
3,
6,
4,
1,
4,
6,
5,
6,
```

```
6,
6,
4,
4,
4,
3,
7,
3,
6,
3,
4,
5,
6,
5,
6,
6,
5,
4,
3,
4,
6,
6,
4,
2,
7,
2,
7,
4,
5,
5,
8,
6,
3,
5,
1,
4,
5,
6,
5,
5,
6,
4,
3,
6,
6,
6,
6,
```

```
5,
7,
6,
4,
8,
6,
6,
6,
5,
4,
6,
5,
2,
3,
5,
7,
3,
5,
6,
4,
7,
6,
5,
5,
6,
6,
7,
4,
5,
4,
6,
8,
6,
5,
7,
6,
8,
4,
5,
4,
7,
1,
6,
5,
7,
6,
6,
```

```
5,
7,
4,
3,
4,
1,
7,
7,
7,
5,
5,
5,
5,
7,
3,
7,
2,
5,
6,
6,
3,
4,
5,
3,
4,
5,
5,
5,
5,
6,
5,
4,
4,
4,
2,
4,
5,
4,
5,
1,
4,
8,
5,
6,
7,
3,
2,
```

```
5,
5,
6,
6,
7,
4,
6,
4,
5,
5,
7,
4,
5,
4,
4,
7,
5,
4,
7,
6,
6,
5,
5,
4,
4,
1,
3,
6,
4,
7,
4,
3,
3,
6,
6,
4,
0,
4,
5,
5,
3,
5,
7,
5,
7,
5,
6,
```

```
3,
4,
5,
4,
5,
6,
6,
4,
5,
5,
6,
3,
6,
3,
5,
4,
6,
6,
5,
6,
5,
4,
4,
3,
6,
7,
5,
7,
4,
7,
5,
6,
7,
5,
5,
6,
5,
5,
5,
3,
4,
8,
5,
5,
5,
4,
4,
```

```
9,
6,
6,
5,
3,
4,
1,
7,
7,
4,
3,
5,
6,
5,
4,
8,
4,
8,
3,
6,
4,
4,
6,
4,
4,
3,
5,
6,
6,
6,
6,
4,
1,
5,
6,
5,
5,
5,
5,
7,
5,
2,
2,
3,
6,
6,
6,
```

```
3,
5,
4,
2,
2,
5,
6,
5,
6,
6,
5,
2,
7,
6,
5,
6,
3,
6,
3,
6,
3,
2,
7,
5,
8,
6,
7,
7,
7,
6,
5,
6,
5,
6,
7,
5,
5,
5,
2,
4,
5,
7,
5,
6,
8,
7,
7,
```

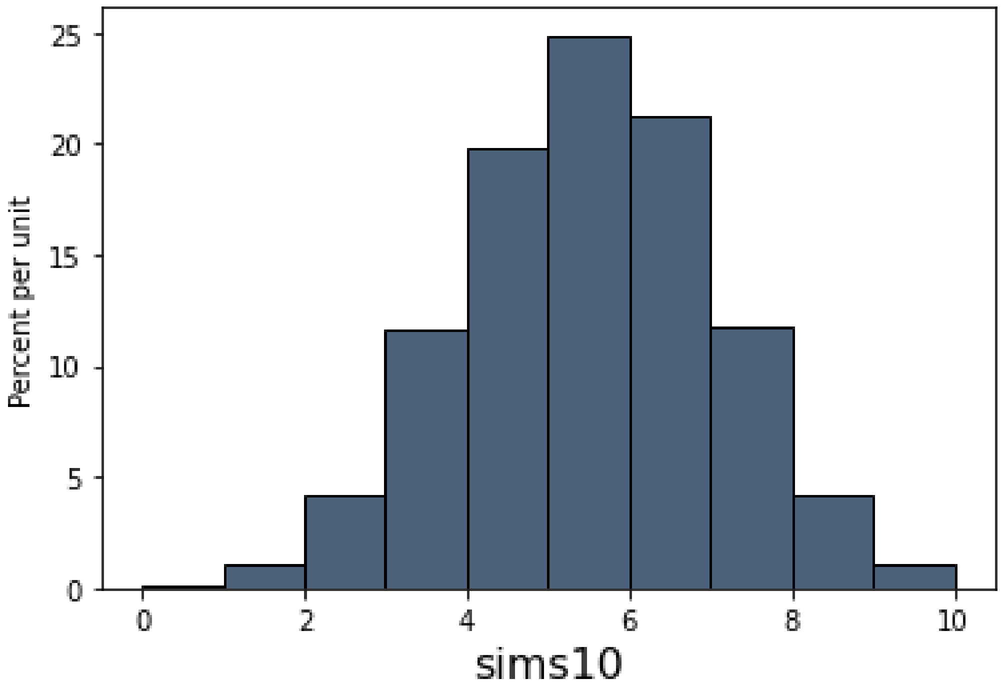
```
4,
4,
4,
3,
7,
6,
6,
5,
5,
2,
5,
6,
6,
4,
7,
2,
7,
5,
6,
5,
7,
3,
8,
3,
3,
…]
```

Let's put these in a table, and then make a histogram to see how often each trial number happens. To make sure we just get a count of how many are at each interval, we need to get the "bins" right.

[21]:
```
list(range(11))
```

[21]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

[22]:
```
simtable = Table().with_column("sims10",trials10)
simtable.hist("sims10", bins=range(11))
```



Let's see what happens with 20 coin flips. First we create a bunch of simulations:

[24]:
```
trials20 = [np.random.binomial(n=20, p=.5) for _ in range(ntrials)]
```

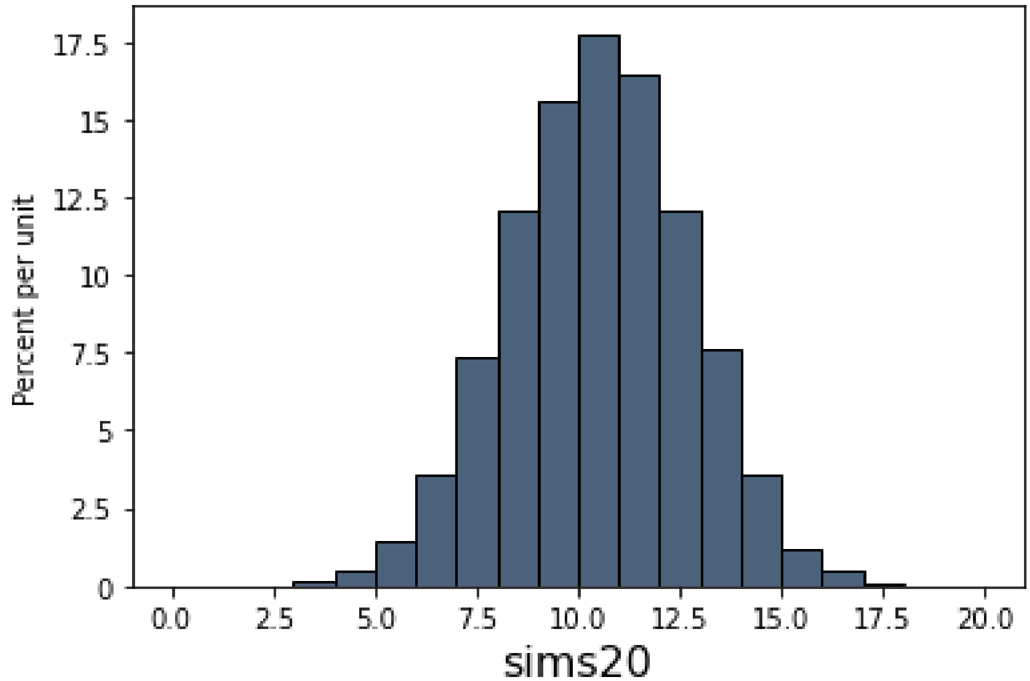
And then add the new trials to simtable using the .with_column() function.

[25]:
```
simtable=simtable.with_column("sims20", trials20)
simtable
```

[25]:
```
sims10 | sims20
6      | 11
5      | 10
5      | 10
7      | 13
6      | 11
6      | 11
6      | 11
4      | 8
4      | 8
5      | 10
… (9990 rows omitted)
```

Question 3.1 Make a histogram of the number of heads in the trials with 20 flips. Make sure to set the bins so there each one contains exactly one integer.

[26]:
```
simtable.hist("sims20", bins=range(21)) # SOLUTION
```

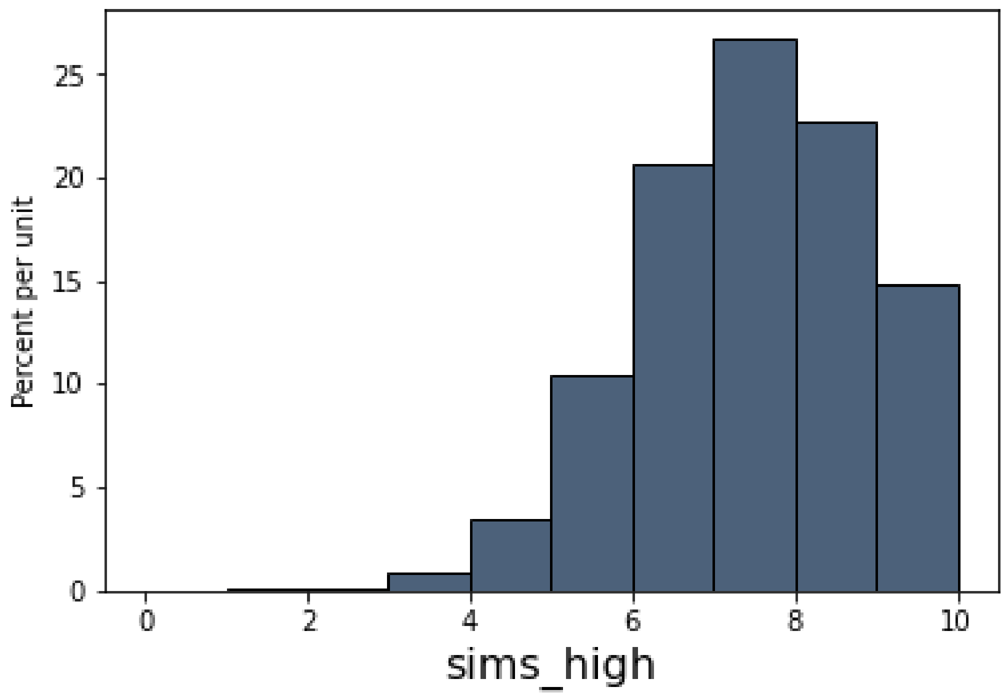


Let's see what this looks like with a different probability of success. Here is a set of 10 trials with a higher probaility of success ($p = .7$)

[28]:
```
trials_high = [np.random.binomial(n=10, p=.7) for _ in range(ntrials)]
```

Question 3.2. Add this array to `simtable`, as a variable called `sims_high`, and create a histogram which shows the frequency of heads in these trials

[29]:
```
simtable=simtable.with_column("sims_high", trials_high) # SOLUTION
simtable.hist("sims_high", bins=range(11)) # SOLUTION
```



[30]:
```
simtable
```

[30]:
```
sims10 | sims20 | sims_high
6      | 11     | 7
5      | 10     | 7
5      | 10     | 7
7      | 13     | 5
6      | 11     | 6
6      | 11     | 6
6      | 11     | 6
4      | 8      | 8
4      | 8      | 8
5      | 10     | 7
… (9990 rows omitted)
```

Question 3.3. Compare this to the graph where $p = .5$

Type your answer here, replacing this text.

SOLUTION: Answer to 3.3 here

Next we want to figure out exactly how often a voter is pivotal in different situations. To do this, let's create a variable called `pivot10` which is true when there are exactly 5 other voters choosing each candidate.

[31]:
```
simtable = simtable.with_column("pivot10", simtable.column("sims10")==5)
simtable
```

[31]:
```
sims10 | sims20 | sims_high | pivot10
6      | 11     | 7         | False
5      | 10     | 7         | True
5      | 10     | 7         | True
7      | 13     | 5         | False
6      | 11     | 6         | False
6      | 11     | 6         | False
6      | 11     | 6         | False
4      | 8      | 8         | False
4      | 8      | 8         | False
5      | 10     | 7         | True
… (9990 rows omitted)
```

We can then count the number of trials where a voter was pivotal.

[32]:
```
sum(simtable.column("pivot10"))
```

[32]:
```
2489
```

Since there were 10,000 trials, we can convert this into a percentage:

[33]:
```
sum(simtable.column("pivot10"))/ntrials
```

[33]:
```
0.2489
```

Question 3.4. Write code to determine what proportion of the time a voter is pivotal when $n = 20$

[34]:
```
simtable=simtable.with_column("pivot20", simtable.column("sims20")==10) #␣
 ↪SOLUTION
pivotal_freq = sum(simtable.column("pivot20"))/ntrials # SOLUTION
pivotal_freq
```

[34]:
```
0.1776
```

[ ]:
```
grader.check("q3d")
```

To explore how chaning the size of the electorate and the probabilities of voting affect the probability of being pivotal without having to go through all of these steps, we will define a function which does one simulation and then checks whether a new voter would be pivotal.

[37]:
```
def one_pivot(n,p):
    return 1*(np.random.binomial(n=n,p=p)==n/2)
```

Run this a few times.

[38]:
```
one_pivot(n=10, p=.6)
```

[38]:
```
1
```

Let's see how the probability of being pivotal changes with a higher n. To do so, we will use the same looping trick to store 10,000 simulations in an array called piv_trials100. (Note we defined `ntrials=10,000` above)

[39]:
```
def pivotal_prob(p):
    return sum(one_pivot(n=100, p=.5) for _ in range(ntrials))/ntrials
interact(pivotal_prob, p=(0,1, .1))
```

```
interactive(children=(FloatSlider(value=0.0, description='p', max=1.0),␣
↪Output()), _dom_classes=('widget-inter…
```

[39]:
```
<function __main__.pivotal_prob(p)>
```

Or a lower p

[40]:
```
piv_trials100 = [one_pivot(n=100, p=.4) for _ in range(ntrials)]
sum(piv_trials100)/ntrials
```

[40]:
```
0.0094
```

Question 3.5 Write a line of code to simulate how often a voter will be pivotal in an electorate with 50 voters and $p = .55$

[42]:
```
piv_trials35 = [one_pivot(n=50, p=.55) for _ in range(ntrials)] # SOLUTION
pivotal_freq = sum(piv_trials35)/ntrials # SOLUTION
pivotal_freq
```

[42]:
```
0.0904
```

[ ]:
```
grader.check("q3e")
```

Question 3.6 (Optional) Try running the one_pivot function with an odd number of voters. What happens and why?

[44]:
```
one_pivot(n=11, p=.5) # SOLUTION
```

[44]:
```
0
```

1.4 Part 4. Pivotal votes with groups

To learn about situations like the electoral college, let's do a simluation with groups. Imagine there are three groups, who all make a choice by majority vote. The winning candidate is the one who wins a majority vote of the majority of groups, in this case at least two groups.

Questions like this become interesting when the groups vary, maybe in size or in predisposition towards certain candidates. To get started, we will look at an example where all the groups have 50 voters. Group 1 leans against candidate A, group B is split, and group C leans towards group A.

We start by making a table with the number of votes for candidate A in each group. All groups have 50 members, but they have different probabilities of voting for A.

[46]:
```
#Group sizes
n1=50
n2=50
n3=50
# Probability of voting for A, by group
p1=.4
p2=.5
p3=.6
np.random.seed(88)
# Creating arrays for simulations for each group
group1 = [np.random.binomial(n=n1, p=p1) for _ in range(ntrials)]
group2 = [np.random.binomial(n=n2, p=p2) for _ in range(ntrials)]
group3 = [np.random.binomial(n=n3, p=p3) for _ in range(ntrials)]
#Putting the arrays into a table
grouptrials = Table().with_columns("votes1",group1,
                                  "votes2", group2,
                                  "votes3",group3)
grouptrials
```

[46]:

votes1	votes2	votes3
21	26	29
20	25	30
20	25	30
24	29	26
22	27	28
22	27	28
22	27	28
17	22	33
17	22	33
20	25	30

… (9990 rows omitted)

Next we create a variable to check whether an individual voter would be pivotal if placed in each group.

[47]:
```
grouptrials = grouptrials.with_columns("voter piv1", 1*(grouptrials.
 ↪column("votes1")==n1/2),
                                        "voter piv2", 1*(grouptrials.
 ↪column("votes2")==n2/2),
```

```
                                              "voter piv3", 1*(grouptrials.
 ↪column("votes3")==n3/2))
grouptrials
```

[47]:
```
votes1 | votes2 | votes3 | voter piv1 | voter piv2 | voter piv3
21     | 26     | 29     | 0          | 0          | 0
20     | 25     | 30     | 0          | 1          | 0
20     | 25     | 30     | 0          | 1          | 0
24     | 29     | 26     | 0          | 0          | 0
22     | 27     | 28     | 0          | 0          | 0
22     | 27     | 28     | 0          | 0          | 0
22     | 27     | 28     | 0          | 0          | 0
17     | 22     | 33     | 0          | 0          | 0
17     | 22     | 33     | 0          | 0          | 0
20     | 25     | 30     | 0          | 1          | 0
… (9990 rows omitted)
```

Let's check how often voters in group 1 are pivotal

[48]:
```
sum(grouptrials.column("voter piv1"))/ntrials
```

[48]: 0.0408

Question 4a. Check how often voters in groups 2 and 3 are pivotal.

[49]:
```
group2pivotal = sum(grouptrials.column("voter piv2"))/ntrials #SOLUTION
group3pivotal = sum(grouptrials.column("voter piv3"))/ntrials #SOLUTION
group2pivotal, group3pivotal
```

[49]: (0.1163, 0.0408)

[ ]:
```
grader.check("q4a")
```

Question: you should get that two of the groups have a similar probability of being pivotal, but one is different. Which is different any why?

Type your answer here, replacing this text.

SOLUTION

Now let's check if each group is pivotal, i.e., if the group changing their vote changes which candidate wins the majority of groups. [Note: tricky stuff about ties here is important]

[52]:
```
group1piv = 1*((grouptrials.column("votes2") <= n2/2)*(grouptrials.
 ↪column("votes3") >= n3/2)+
   (grouptrials.column("votes2") >= n2/2)*(grouptrials.column("votes3") <= n3/
 ↪2))
group2piv = 1*((grouptrials.column("votes1") <= n2/2)*(grouptrials.
 ↪column("votes3") >= n3/2)+
```

```
    (grouptrials.column("votes1") >= n2/2)*(grouptrials.column("votes3") <= n3/
 ↪2))
group3piv = 1*((grouptrials.column("votes1") <= n2/2)*(grouptrials.
 ↪column("votes2") >= n3/2)+
    (grouptrials.column("votes1") >= n2/2)*(grouptrials.column("votes2") <= n3/
 ↪2))

grouptrials = grouptrials.with_columns("group piv1", group1piv,
                                       "group piv2", group2piv,
                                       "group piv3", group3piv)
grouptrials
```

[52]:
```
votes1 | votes2 | votes3 | voter piv1 | voter piv2 | voter piv3 | group piv1 |
group piv2 | group piv3
21     | 26     | 29     | 0          | 0          | 0          | 0          | 1
| 1
20     | 25     | 30     | 0          | 1          | 0          | 1          | 1
| 1
20     | 25     | 30     | 0          | 1          | 0          | 1          | 1
| 1
24     | 29     | 26     | 0          | 0          | 0          | 0          | 1
| 1
22     | 27     | 28     | 0          | 0          | 0          | 0          | 1
| 1
22     | 27     | 28     | 0          | 0          | 0          | 0          | 1
| 1
22     | 27     | 28     | 0          | 0          | 0          | 0          | 1
| 1
17     | 22     | 33     | 0          | 0          | 0          | 1          | 1
| 0
17     | 22     | 33     | 0          | 0          | 0          | 1          | 1
| 0
20     | 25     | 30     | 0          | 1          | 0          | 1          | 1
| 1
… (9990 rows omitted)
```

How often is each group pivotal?

[53]:
```
group1_pivotal_rate = sum(grouptrials.column("group piv1"))/ntrials # SOLUTION
group2_pivotal_rate = sum(grouptrials.column("group piv2"))/ntrials # SOLUTION
group3_pivotal_rate = sum(grouptrials.column("group piv3"))/ntrials # SOLUTION
group1_pivotal_rate, group2_pivotal_rate, group3_pivotal_rate
```

[53]:
```
(0.6486, 1.0, 0.5085)
```

[ ]:
```
grader.check("q4c")
```

Two groups should have similar probabilities, with one group fairly different. Why is

the case?

Type your answer here, replacing this text.

SOLUTION: Answer to 2c here

A voter will be pivotal "overall" if they are pivotal within the group and the group is pivotal in the election. We can compute this by multiplying whether a voter is pivotal by whether their group is pivotal: the only voters who will be pivotal (represented by a 1) will have a 1 in both columns.

[60]:
```
grouptrials = grouptrials.with_columns("overall piv 1",
                                       grouptrials.column("voter␣
↪piv1")*grouptrials.column("group piv1"),
                                        "overall piv 2",
                                       grouptrials.column("voter␣
↪piv2")*grouptrials.column("group piv2"),
                                        "overall piv 3",
                                       grouptrials.column("voter␣
↪piv3")*grouptrials.column("group piv3"))

grouptrials
```

[60]:
```
votes1 | votes2 | votes3 | voter piv1 | voter piv2 | voter piv3 | group piv1 |
group piv2 | group piv3 | overall piv 1 | overall piv 2 | overall piv 3
21     | 26     | 29     | 0          | 0          | 0          | 0          | 1
| 1          | 0          | 0             | 0             | 0
20     | 25     | 30     | 0          | 1          | 0          | 1          | 1
| 1          | 0          | 0             | 1             | 0
20     | 25     | 30     | 0          | 1          | 0          | 1          | 1
| 1          | 0          | 0             | 1             | 0
24     | 29     | 26     | 0          | 0          | 0          | 0          | 1
| 1          | 0          | 0             | 0             | 0
22     | 27     | 28     | 0          | 0          | 0          | 0          | 1
| 1          | 0          | 0             | 0             | 0
22     | 27     | 28     | 0          | 0          | 0          | 0          | 1
| 1          | 0          | 0             | 0             | 0
22     | 27     | 28     | 0          | 0          | 0          | 0          | 1
| 1          | 0          | 0             | 0             | 0
17     | 22     | 33     | 0          | 0          | 0          | 1          | 1
| 0          | 0          | 0             | 0             | 0
17     | 22     | 33     | 0          | 0          | 0          | 1          | 1
| 0          | 0          | 0             | 0             | 0
20     | 25     | 30     | 0          | 1          | 0          | 1          | 1
| 1          | 0          | 0             | 1             | 0
… (9990 rows omitted)
```

What is the probability of a voter in each group being pivotal?

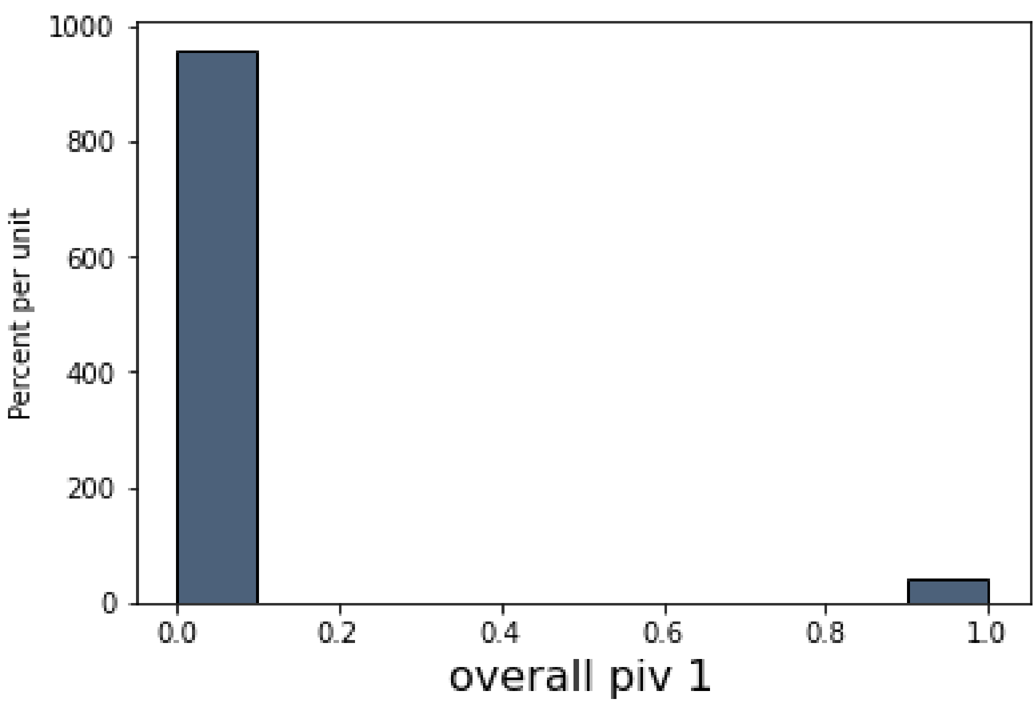
[61]:
```
voter_1_pivotal_prob = sum(grouptrials.column("overall piv 1"))/ntrials
voter_2_pivotal_prob = sum(grouptrials.column("overall piv 2"))/ntrials #␣
 ↪SOLUTION
voter_3_pivotal_prob = sum(grouptrials.column("overall piv 3"))/ntrials #␣
 ↪SOLUTION

voter_1_pivotal_prob, voter_2_pivotal_prob, voter_3_pivotal_prob
```

[61]:
```
(0.0408, 0.1163, 0.0408)
```

[ ]:
```
grader.check("q4e")
```

We can graph the frequency with which a voter in a group is pivotal using `.hist("COLUMNNAME")`. Below, we graph the frequency of a voter in group one being pivotal. You can try graphing the frequency for voters in other groups by changing the column name below.

[66]:
```
grouptrials.hist("overall piv 1")
```



How frequently do we see the combination of different voter and group pivotal status? In the cells below, we calculate both the absolute frequency as well as percentage of times in which a voter is or is not pivotal within their group as well as their group is or is not. For example, for the cell in which `col_0` and `row_0` equal 0, neither the voter nor the group was pivotal.

[67]:
```
pd.crosstab(grouptrials.column("voter piv1"), grouptrials.column("group piv1"))
```

[67]:
```
col_0     0     1
row_0
0      3514  6078
1         0   408
```

This cell mimics the above, except that by normalizing, we see the frequencies as a percentage overall.

[68]:
```
pd.crosstab(grouptrials.column("voter piv1"), grouptrials.column("group piv1"),␣
↪normalize=True)
```

[68]:
```
col_0       0       1
row_0
0      0.3514  0.6078
1      0.0000  0.0408
```

Here is a function that ties it all together. It creates a table with group population parameter sizes as well as parameters for the probability of each kind of voter voting for candidate A.

[69]:
```
def maketable(n1=50, n2=50, n3=50, p1=.4, p2=.5, p3=.6, ntrials=10000):
    group1 = [np.random.binomial(n=n1, p=p1) for _ in range(ntrials)]
    group2 = [np.random.binomial(n=n2, p=p2) for _ in range(ntrials)]
    group3 = [np.random.binomial(n=n3, p=p3) for _ in range(ntrials)]
    grouptrials = Table().with_columns("votes1",group1,
                                 "votes2", group2,
                                 "votes3",group3)
    grouptrials = grouptrials.with_columns("voter piv1", 1*(grouptrials.
↪column("votes1")==n1/2),
                                       "voter piv2", 1*(grouptrials.
↪column("votes2")==n2/2),
                                       "voter piv3", 1*(grouptrials.
↪column("votes3")==n3/2))
    group1piv = 1*((grouptrials.column("votes2") <= n2/2)*(grouptrials.
↪column("votes3") >= n3/2)+
   (grouptrials.column("votes2") >= n2/2)*(grouptrials.column("votes3") <= n3/
↪2))
    group2piv = 1*((grouptrials.column("votes1") <= n2/2)*(grouptrials.
↪column("votes3") >= n3/2)+
   (grouptrials.column("votes1") >= n2/2)*(grouptrials.column("votes3") <= n3/
↪2))
    group3piv = 1*((grouptrials.column("votes1") <= n2/2)*(grouptrials.
↪column("votes2") >= n3/2)+
   (grouptrials.column("votes1") >= n2/2)*(grouptrials.column("votes2") <= n3/
↪2))
    grouptrials = grouptrials.with_columns("group piv1", group1piv,
                                       "group piv2", group2piv,
                                       "group piv3", group3piv)
```

```
    grouptrials = grouptrials.with_columns("overall piv1",
                                           grouptrials.column("voter␣
↪piv1")*grouptrials.column("group piv1"),
                                            "overall piv2",
                                           grouptrials.column("voter␣
↪piv2")*grouptrials.column("group piv2"),
                                            "overall piv3",
                                           grouptrials.column("voter␣
↪piv3")*grouptrials.column("group piv3"))
    return grouptrials
```

[70]:
```
test = maketable()
test
```

[70]:
```
votes1 | votes2 | votes3 | voter piv1 | voter piv2 | voter piv3 | group piv1 |
group piv2 | group piv3 | overall piv1 | overall piv2 | overall piv3
21     | 30     | 33     | 0          | 0          | 0          | 0          | 1
| 1          | 0            | 0            | 0
20     | 29     | 33     | 0          | 0          | 0          | 0          | 1
| 1          | 0            | 0            | 0
20     | 30     | 32     | 0          | 0          | 0          | 0          | 1
| 1          | 0            | 0            | 0
24     | 25     | 28     | 0          | 1          | 0          | 1          | 1
| 1          | 0            | 1            | 0
22     | 21     | 24     | 0          | 0          | 0          | 0          | 0
| 0          | 0            | 0            | 0
22     | 26     | 31     | 0          | 0          | 0          | 0          | 1
| 1          | 0            | 0            | 0
22     | 19     | 29     | 0          | 0          | 0          | 1          | 1
| 0          | 0            | 0            | 0
17     | 29     | 32     | 0          | 0          | 0          | 0          | 1
| 1          | 0            | 0            | 0
17     | 25     | 27     | 0          | 1          | 0          | 1          | 1
| 1          | 0            | 1            | 0
20     | 22     | 26     | 0          | 0          | 0          | 1          | 1
| 0          | 0            | 0            | 0
… (9990 rows omitted)
```

What happens as we change the number of voters in each group relative to one another? In the following cell, use the sliders to change the number of voters in each group.

[71]:
```
def voter_piv_rate(n1, n2, n3):
    sims = maketable(n1, n2, n3)
    for i in range(1,4):
        print("Voter and Group are Both Pivotal Frequency", sum(sims.
↪column(f"overall piv{i}"))/ntrials)
        sims.hist(f"overall piv{i}")
```

```
        plt.show()
interact(voter_piv_rate, n1=IntSlider(min=0,max=300, value=100),␣
 ↪n2=IntSlider(min=0,max=300, value=100), n3=IntSlider(min=0,max=300,␣
 ↪value=100))
```

```
interactive(children=(IntSlider(value=100, description='n1', max=300),␣
 ↪IntSlider(value=100, description='n2', …
```

[71]:
```
<function __main__.voter_piv_rate(n1, n2, n3)>
```

What happens as you change the sliders? Can you make the frequencies the same? How?

Type your answer here, replacing this text.

SOLUTION: Answer to 4f here

If we keep the voter populations static, but change their probability of voting for candidate A, what happens?

[72]:
```
def voter_piv_rate(p1, p2, p3):
    sims = maketable(p1=p1, p2=p2, p3=p3)
    for i in range(1,4):
        print("Voter and Group are Both Pivotal Frequency", sum(sims.
 ↪column(f"overall piv{i}"))/ntrials)
        sims.hist(f"overall piv{i}")
        plt.show()
    return sims
interact(voter_piv_rate, p1=FloatSlider(min=0,max=1, value=.4, step=.1),␣
 ↪p2=FloatSlider(min=0,max=1, value=.5, step=.1), p3=FloatSlider(min=0,max=1,␣
 ↪value=.6, step=.1))
```

```
interactive(children=(FloatSlider(value=0.4, description='p1', max=1.0),␣
 ↪FloatSlider(value=0.5, description='p…
```

[72]:
```
<function __main__.voter_piv_rate(p1, p2, p3)>
```

What happens as you change the sliders? Can you make the frequencies the same? How?

Type your answer here, replacing this text.

SOLUTION: Answer to 4g here

[ ]:
```

```

To double-check your work, the cell below will rerun all of the autograder tests.

[ ]:
```
grader.check_all()
```

1.5 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!**

These are some submission instructions.

[ ]:
```
# Save your notebook first, then run this cell to export your submission.
grader.export()
```