Statistics and Hypothesis Testing

Welcome to lab 5! This lab covers the use of computer simulation, in combination with data, to answer questions about the world. You'll answer two questions:

1. Suppose we estimate a parameter using a statistic computed from a sample of data. That statistic could have turned out different if the random sample had turned out different. What is the probability distribution of values of that statistic?
2. Was a panel of jurors selected at random from the population of eligible jurors?

In [1]:
```
# Run this cell to set up the notebook, but please don't change it.

# These lines import the Numpy and Datascience modules.
import numpy as np
from datascience import *

# These lines do some fancy plotting magic.
import matplotlib
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
%matplotlib inline
import warnings
warnings.simplefilter('ignore', FutureWarning)
```

1. Dungeons and Dragons and Sampling

In the game Dungeons & Dragons, each player plays the role of a fantasy character.

A player performs actions by rolling a 20-sided die, adding a "modifier" number to the roll, and comparing the total to a threshold for success. The modifier depends on her character's competence in performing the action.

For example, suppose Alice's character, a barbarian warrior named Roga, is trying to knock down a heavy door. She rolls a 20-sided die, adds a modifier of 11 to the result (because her character is good at knocking down doors), and succeeds if the total is greater than 15.

Question 1

Write code that simulates that procedure. Compute three values: the result of Alice's roll (`roll_result`), the result of her roll plus Roga's modifier (`modified_result`), and a boolean value indicating whether the action succeeded (`action_succeeded`). **Do not fill in any of the results manually**; the entire simulation should happen in code.

Hint: A roll of a 20-sided die is a number chosen uniformly from the array `make_array(1, 2, 3, 4, ..., 20)` . So a roll of a 20-sided die *plus 11* is a number chosen uniformly from that array, plus 11.

In [2]:
```
die = np.arange(1,21)
```

```
def dungeon ():
    '''return the result whether the player succeeded or failed'''
    roll_result = np.random.choice(die,1)
    modified_result = roll_result + 11

    if modified_result > 15:
        action_succeeded = "succeeded"
    else :
        action_succeeded = "failed"

    # The next line just prints out your results in a nice way
    # once you're done.  You can delete it if you want.
    print("On a modified roll of ", modified_result, ", Alice's action",acti

dungeon ()
```

```
On a modified roll of  [22] , Alice's action succeeded
```

Question 2

Run your cell 7 times to manually estimate the chance that Alice succeeds at this action. (Don't use math or an extended simulation.). Your answer should be a fraction.

In [3]:
```
dungeon ()
dungeon ()
dungeon ()
dungeon ()
dungeon ()
dungeon ()
dungeon ()

rough_success_chance = 4/7
```

```
On a modified roll of  [31] , Alice's action succeeded
On a modified roll of  [26] , Alice's action succeeded
On a modified roll of  [22] , Alice's action succeeded
On a modified roll of  [15] , Alice's action failed
On a modified roll of  [23] , Alice's action succeeded
On a modified roll of  [12] , Alice's action failed
On a modified roll of  [20] , Alice's action succeeded
```

Suppose we don't know that Roga has a modifier of 11 for this action. Instead, we observe the modified roll (that is, the die roll plus the modifier of 11) from each of 7 of her attempts to knock down doors. We would like to estimate her modifier from these 7 numbers.

Question 3

Write a Python function called `simulate_observations`. It should take no arguments, and it should return an array of 7 numbers. Each of the numbers should be the modified roll from one simulation. **Then**, call your function once to compute an array of 7 simulated modified rolls. Name that array `observations`.

In [4]:
```
modifier = 11
num_observations = 7

def simulate_observations():
```

```
    '''return an array of 7 numbers'''
    simulate = make_array()
    for i in np.arange(num_observations):
        roll_result = np.random.choice(die,1)
        modified_result = roll_result + modifier
        simulate = np.append(simulate, modified_result)
    return simulate

observations = simulate_observations()
observations
```

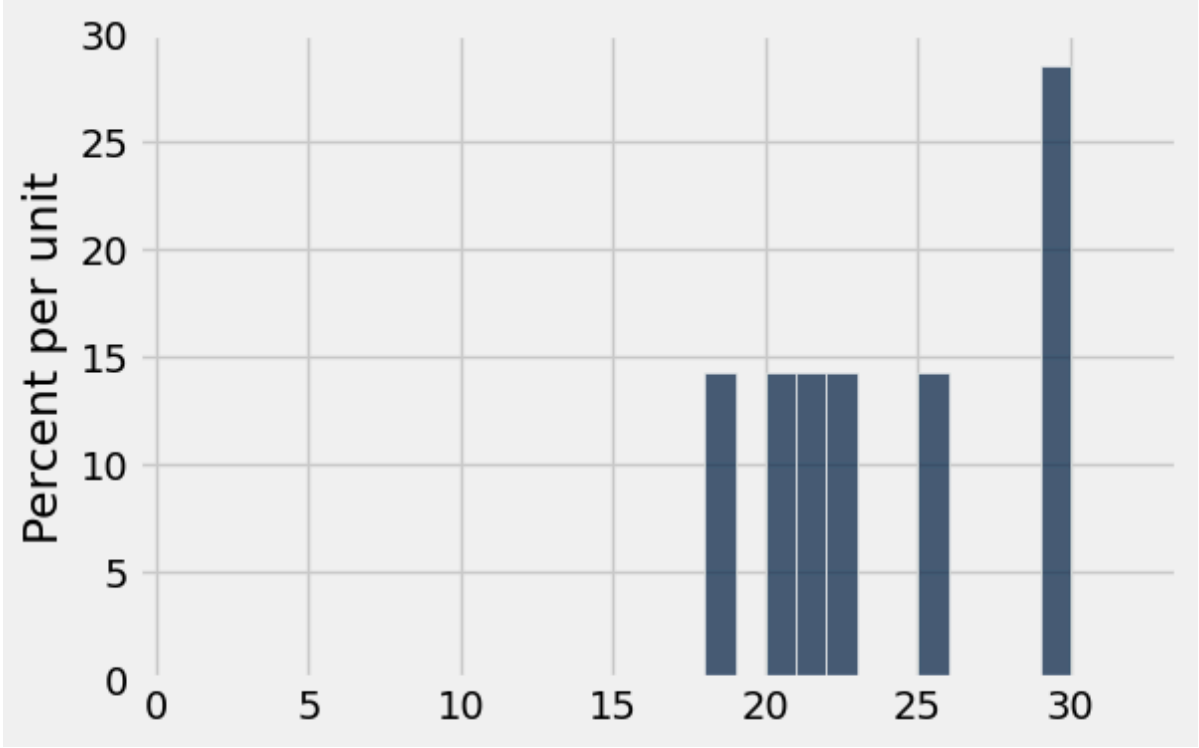
Out[4]:
```
array([ 22.,  29.,  21.,  18.,  25.,  20.,  29.])
```

Question 4

Using your array of 7 observations, make a histogram to display the *probability distribution* of the modified rolls we might see. Check with a neighbor or Kenneth to make sure you have the right histogram.

In [5]:
```
# We suggest using these bins.
roll_bins = np.arange(1, 20+modifier+2, 1)

Table().with_column("",observations).hist(bins=roll_bins)
```



Now let's imagine we don't know the modifier and try to estimate it from `observations`.

One straightforward (but clearly suboptimal) way to do that is to find the *smallest* total roll, since the smallest roll on a 20-sided die is 1.

Question 5

Using that method, estimate `modifier` from `observations`. Name your estimate `min_estimate`.

In [6]:
```
min_estimate = min(observations)-1
min_estimate
```

Out[6]:
```
17.0
```

Another way to estimate the modifier involves the mean of `observations`.

Question 6

Figure out a good estimate based on that quantity. **Then**, write a function named `mean_based_estimator` that computes your estimate. It should take an array of modified rolls (like the array `observations`) as its argument and return an estimate of `modifier` based on those numbers.

In [7]:
```
def mean_based_estimator(nums):
    """Estimate the roll modifier based on observed modified rolls in the ar
    mean_estimate = np.average(nums)-10.5
    return mean_estimate

# Here is an example call to your function.  It computes an estimate
# of the modifier from our 7 observations.
mean_based_estimate = mean_based_estimator(observations)
mean_based_estimate
```

Out[7]:
```
12.928571428571427
```

Question 7

Recall the warplanes estimation problem from lecture. There, it made sense to multiply the mean by 2 to estimate the largest serial number. That doesn't make sense for this estimation problem. Why not?

Answer:

Now we want to know how well our two estimators work. To do that, we'll imagine what *could have happened* when we got our observations. Concretely, that means we'll generate new sets of observations 1000 times (that is, 1000 times, we'll simulate 7 modified rolls). Each time, we'll use both estimators to compute estimates of the modifier for each simulated observation set. Then, we'll display the distribution of those estimates. That will show us how often each estimator does how well.

Question 8

Write code to compute a table called `estimates`. It should have 1000 rows and 2 columns. Each row should correspond to a different sample of 7 simulated modified rolls. The first column, `"min estimate"`, should contain estimates using the `min` of each sample. The second column, `"mean-based estimate"`, should contain estimates using your `mean_based_estimator` function on each sample.

In [8]:
```
min_estimates = make_array()
mean_based_estimates = make_array()
for i in range(1000):
```

```
    new_observations = simulate_observations()
    new_min_estimate = min(new_observations)-1
    new_mean_based_estimate = mean_based_estimator(new_observations)
    min_estimates = np.append(min_estimates, new_min_estimate)
    mean_based_estimates = np.append(mean_based_estimates, new_mean_based_es

estimates = Table().with_columns("min estimate", min_estimates,
                                 "mean-based estimate", mean_based_estimates
estimates
```

Out[8]:

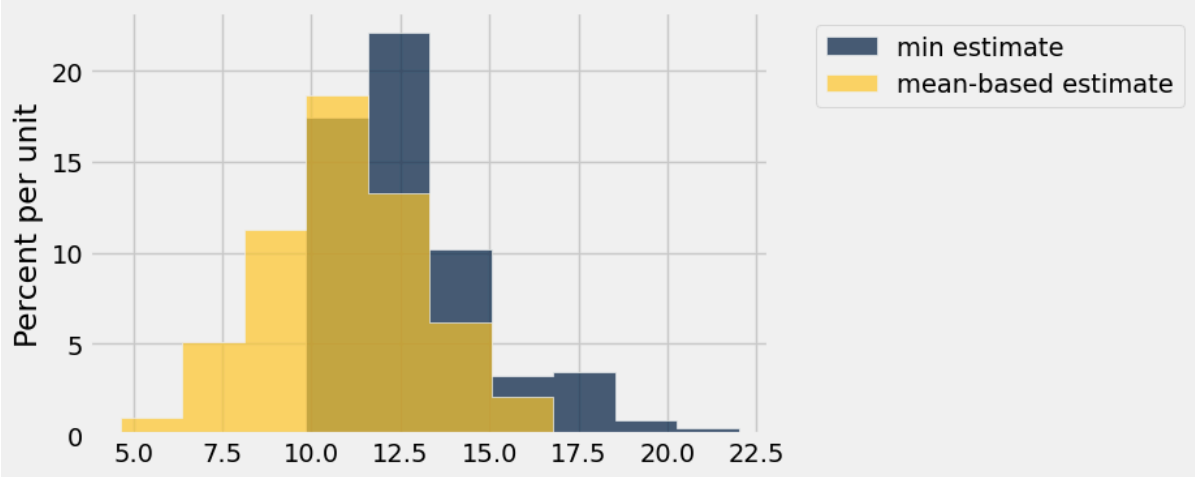
min estimate	mean-based estimate
15	10.0714
11	10.3571
11	10.3571
12	10.3571
12	13.0714
16	10.0714
11	12.2143
15	10.9286
13	9.5
14	10.7857

... (990 rows omitted)

Question 9

Make a plot of the empirical distributions of both statistics. (You should make a *single plot* containing two histograms. Calling the `hist` method of a table that has 2 numerical columns will do this.)

In [9]:
```
#Make your histogram here
estimates.hist()
```



Question 10

Why do the `min`-based estimates bottom out at 12, while some of the `mean`-based estimates are smaller than that?

Answer: Because there are 1 to 20 numbers that can come out from the die and if we add 11, the numbers are larger. The chance of getting the die 1 when we roll the die once is 1/20. Since the min-based estimates subtract 1 from the minimum observation, most of the numbers would be larger than the actual modifier. However, the mean-based estimates are the result of subtracting the average of the die from the average of the observations. Therefore, this would be closer to the actual modifier which makes most of the mean-based estimates being smaller than the min-based estimates.

2. *Swain v. Alabama*

In lecture, we tested the hypothesis that a jury panel in Alameda County was a random sample from the population of Alameda. Let's go through a similar example in another context.

Background

Swain v. Alabama was a US Supreme Court case decided in 1965. A black man, Swain, was accused of raping a white woman, and had been convicted by an all-white jury. The jury was selected from a panel of 100 people that contained only 8 black people and 92 people of other ethnicities. That panel was supposed to be a random sample from the eligible population of Talladega County, which was 26% black and 74% other ethnicities (and, by law, all male). We will assume there were 16,000 people in all of Talladega County eligible to serve on the jury.

Swain's lawyers argued that this reflected a bias in the jury panel selection process. A five-justice majority of the Supreme Court disagreed, asserting that,

> "The overall percentage disparity has been small and reflects no studied attempt to include or exclude a specified number of Negros."

Is 8 enough smaller than 26 to indicate that the jury panel wasn't just a random sample from the eligible population?

It *could* have happened by chance, so this may seem like a judgement call. Indeed, five Supreme Court justices apparently thought it was a matter of judgement.

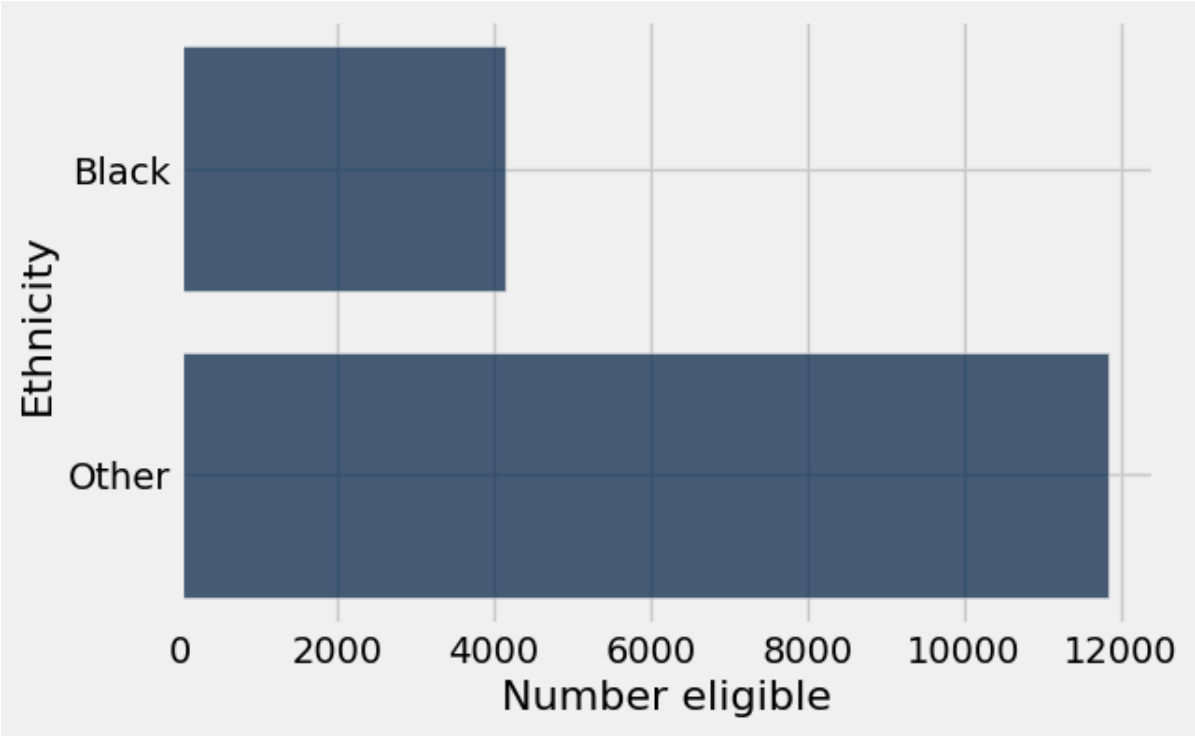
In fact, with a computer simulation, we can answer that is is very unlikely to see a panel of 8 randomly selected from this population. (Note that the scarcity of computers at the time was no excuse. Any statistician could have done this calculation by hand in 1965.)

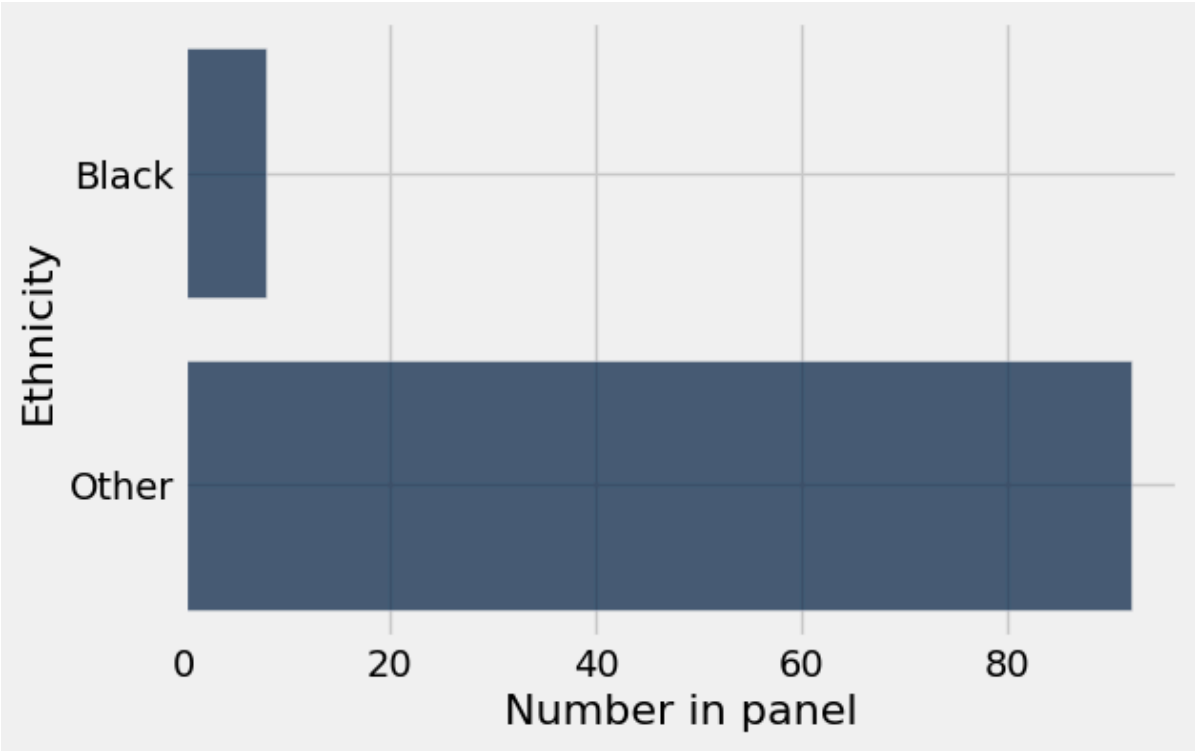
Here are the data:

In [10]:
```
jury = Table.read_table("~/DS_113_S23/Labs/Lab_5/jury.csv")
jury.barh('Ethnicity', "Number eligible")
jury.barh('Ethnicity', "Number in panel")
jury
```

Out[10]:

Ethnicity	Number eligible	Number in panel
Black	4160	8
Other	11840	92





Question 1

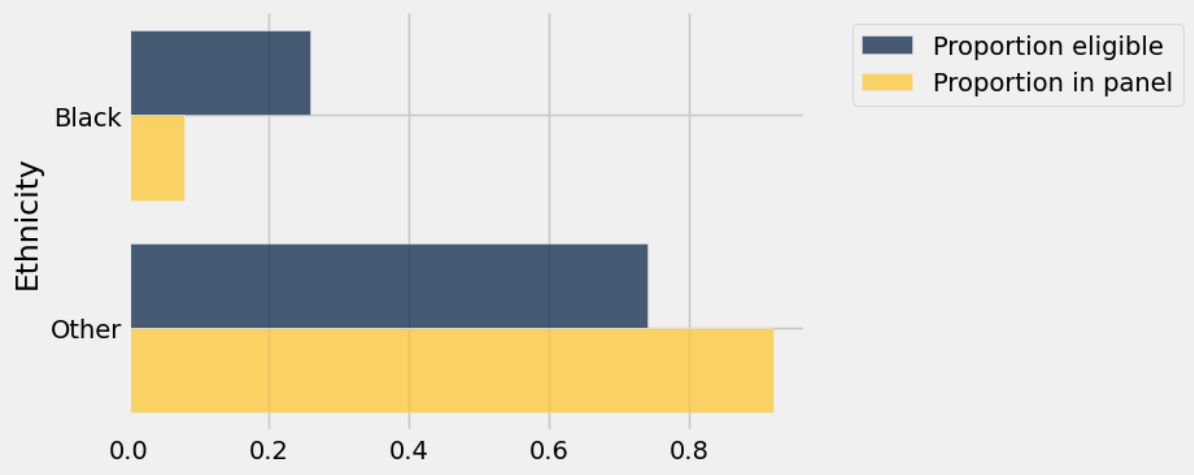
It's helpful to draw both bars in one plot to really see the visual distinction. Why would this not be a good idea with the data that we have so far? Think about what the bar graphs would look like if we plotted them all onto the same chart.

Answer: The largest number is 11840 and the smallest number is 8. If we draw both bars in one plot, we cannot see the amount for the number in panel.

Question 2

Fix the problem by making one bar chart displaying the *proportions* of ethnicities among the eligible population and the panel. **Before you make the chart**, create a table called `with_proportions` that's a copy of `jury` with columns containing these proportions. (The column names are filled in for you.)

In [53]:
```
with_proportions = jury.with_columns(
        "Proportion eligible", (jury["Number eligible"]/sum(jury["Number eli
        "Proportion in panel", (jury["Number in panel"]/sum(jury["Number in
with_proportions.select("Ethnicity","Proportion eligible","Proportion in par
```



To test the hypothesis that the actual panel was a random sample from the eligible population, we first write down a *null hypothesis*. We will imagine (via computer simulation) what the data would typically look like if this hypothesis were true. If the actual data don't look like that, we'll reject the null hypothesis.

Our null hypothesis is straightforward:

> **Null hypothesis:** "The actual panel in Swain's trial was a random sample from the eligible population in Talladega County."

Now, imagine drawing a random sample from the population of eligible jurors in Talladega. There is a 26% chance that any sampled individual is black. We can simulate drawing just 1 person in this way by calling this function:

In [70]:
```
def proportions_from_distribution(table, proportions_column_label, sample_si
    """Produces a random sample with replacement, using proportions from the
    column named proportions_column_label in the given table.  The size of t
    sample is sample_size.

    Each row in the given table should represent one kind of thing.
    Each kind of thing has a proportion, which is the chance that each
    member of the sample is that kind.

    Returns a copy of table with an extra column called "Proportion in randc
    sample".  This column contains the number of elements of each kind that
    ended up in the random sample."""
    proportions = np.random.multinomial(sample_size, table.column(proportior
    return table.with_column("Proportion in random sample", proportions)
```

```
# Sampling 1 person:
one_random_sample = proportions_from_distribution(with_proportions, "Proport
one_random_sample
```

Out[70]:

Ethnicity	Number eligible	Number in panel	Proportion eligible	Proportion in panel	Proportion in random sample
Black	4160	8	0.26	0.08	0.21
Other	11840	92	0.74	0.92	0.79

This table is the same as `with_proportions`, but the last column in this table contains the numbers of Black and Other people in a random sample of 1 person. The ethnicity of the person was decided randomly: there was a 26% chance of selecting a black person

If we repeated this many times, about 26% of the time we'd get a black person. Let's try that out.

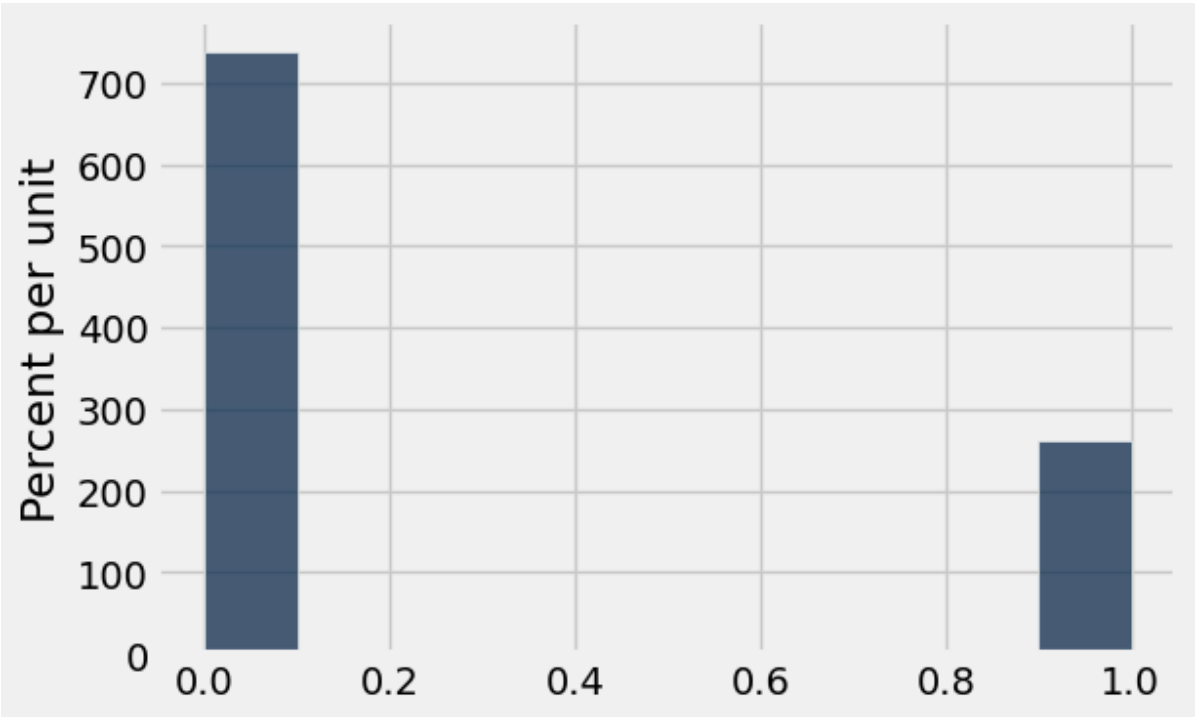
Question 3

Write a function called `simulate_single_samples`. It should take no arguments and return no value. It should simulate 10000 single draws and produce a *histogram* of the numbers of black people in each of those draws.

In [69]:

```
def simulate_single_samples():
    samples = make_array()
    for i in np.arange(10000):
        simulate_sample = proportions_from_distribution(with_proportions, "P
        samples = np.append(samples, simulate_sample.column(5).item(0))
    Table().with_column("", samples).hist()

simulate_single_samples()
```



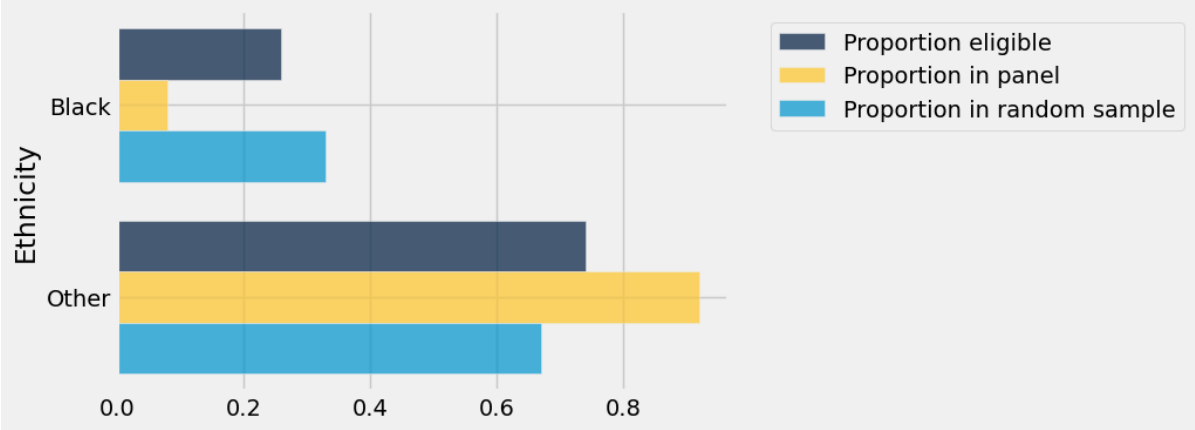
Let's see what happens when we sample 100 people instead of just 1. Remember, this is simulating how jury panels were *supposed* to be selected, according to the law.

Question 4

Call `proportions_from_distribution` to create a table called `one_sample_of_100` that represents one sample of 100 people from among the eligible jurors. This is one panel we could see *if the null hypothesis were true*. **Then,** make a single bar chart displaying the proportions of ethnicities in this sample, in the eligible population, and in the actual panel in Swain's case.

In [71]:
```
one_sample_of_100 = proportions_from_distribution(with_proportions, "Proport
one_sample_of_100.show()
one_sample_of_100.drop(1,2).barh("Ethnicity")
```

Ethnicity	Number eligible	Number in panel	Proportion eligible	Proportion in panel	Proportion in random sample
Black	4160	8	0.26	0.08	0.33
Other	11840	92	0.74	0.92	0.67



Does the panel look like it could have come from this process?

To answer that question, we'll need to sample many times, not just once. And we'll need to summarize each sample with a number (a "test statistic"). We want the number to generally look one way if the null hypothesis is true, and some other way if it's not.

A useful test statistic in cases like this is the *total variation distance* (TVD) between the distribution of ethnicities in the sample and the distribution of ethnicities in the eligible population. Intuitively, this distance should be typically small if the null hypothesis is true, because many samples will have similar proportions of ethnicities as the population from which they're taken.

To compute the TVD visually, make a bar chart of the two distributions, like the one you made above (ignoring the bars displaying the proportions in the panel). Then for each category ("Black" and "Other"), find the absolute difference between the lengths of the bars. Add up those absolute differences, and divide by 2.

Question 5

Look at the bar chart you made above. Without using any code, estimate the TVD between the distribution of ethnicities in the sample and the distribution of ethnicities in the eligible population. Then estimate the TVD between the distribution of ethnicities in

the *actual panel* and the distribution of ethnicities in the eligible population. Note which one is bigger. Check with a neighbor or a TA to verify your answer.

In [15]:
```
rough_tvd_between_sample_and_population = 0.07
rough_tvd_between_panel_and_population = 0.18
```

Question 6

Write a function called `tvd_from_eligible_population`. It should take 1 argument: an array of proportions of ethnicities. The first element in the array is the proportion of black people, and the second element in the array is the proportion of others. The function should return the TVD between that distribution of ethnicities and the distribution in the eligible population.

In [34]:
```
def tvd_from_eligible_population(proportions):
    difference_black = abs(proportions.item(0) - one_random_sample.column(3)
    difference_other = abs(proportions.item(1) - one_random_sample.column(3)
    return ((difference_black+difference_other) / 2)


# An example call to your function.  This computes the
# TVD you estimated in question 5.
print("TVD between eligible population and a random sample:", tvd_from_eligi
# ...and this computes the other TVD you estimated.
print("TVD between eligible population and the actual panel:", tvd_from_elig
```

```
TVD between eligible population and a random sample: 0.04000000000000001
TVD between eligible population and the actual panel: 0.18000000000000002
```

Now you have all the ingredients for running an hypothesis test:

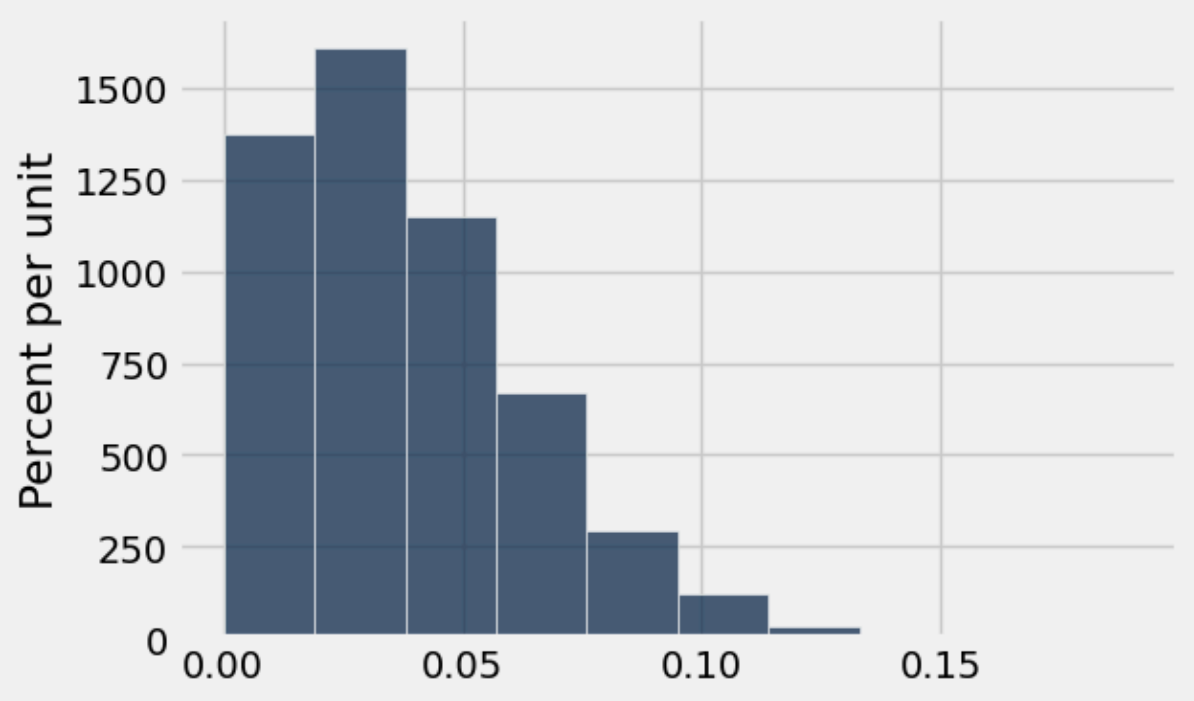
1. You can generate data you would see if the null hypothesis were true. You did that once in question 4.
2. You can compute a test statistic on each simulated dataset by calling `tvd_from_eligible_population`.

Question 7

Write a function called `simulate_tvds_under_null`. It should take no arguments and return no value. It should run **20,000** simulations under the null hypothesis, compute the test statistic for each one, and produce a histogram of those **20,000** test statistics.

In [39]:
```
def simulate_tvds_under_null():
    stat_array = make_array()
    for i in np.arange(20000):
        sample_100 = proportions_from_distribution(with_proportions, "Propor
        test_statistic = tvd_from_eligible_population(sample_100.column(5))
        stat_array = np.append(stat_array, test_statistic)
    Table().with_column("", stat_array).hist()

simulate_tvds_under_null()
```

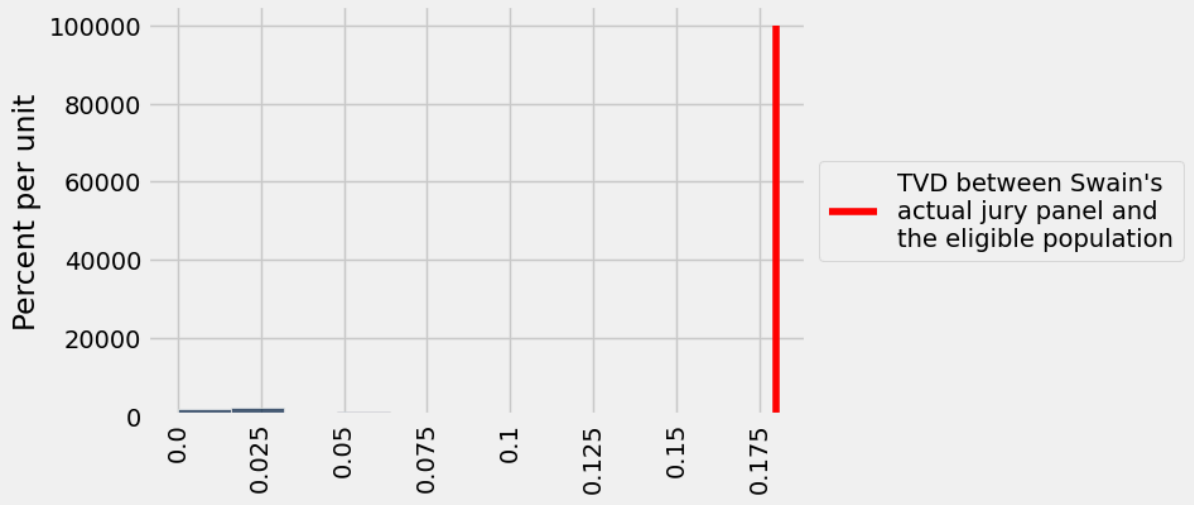


Run the cell below to see the results of your simulation augmented with a red line that shows the TVD between the actual panel and the eligible population (which you estimated in question 5).

In [40]:
```
# You don't need to read this function unless you're interested.
def draw_red_line(spot, height, description):
    """Draw a vertical red line at a spot in a graph."""
    plt.plot([spot, spot], [0, height], color="red", label=description)
    # Draw a legend off to the right to explain the red line.
    plt.legend(loc="center left", bbox_to_anchor=[1, .5])

# Now we'll redraw our histogram and draw a line at the observed TVD.
simulate_tvds_under_null()
draw_red_line(tvd_from_eligible_population(with_proportions.column("Proporti
```

```
/opt/tljh/user/lib/python3.9/site-packages/datascience/tables.py:5865: User
Warning: FixedFormatter should only be used together with FixedLocator
  axis.set_xticklabels(ticks, rotation='vertical')
```



Question 8

Do we have evidence for or against the null hypothesis, or not much evidence either way? Discuss with a neighbor or a TA.

Answer: Since all of the test statistics are below 0.05, we reject the null hypothesis. Also, we can see that the 20,000 test statistics are below the TVD between Swain's actual jury panel and the eligible population, which is about 0.18. Therefore, we have evidence against the null hypothesis.