Lab 8: How Many Piano Tuners in New York City?

Due Date: Wednesday April 22, 2020 at 8:00PM EDT

Late labs, even by one minute, will be graded as zero. Improperly formatted labs also count as late. Note that the course academic honesty policy applies to every assignment, including this one. Please remeber to submit this lab on Jupyterhub and Gradescope.

In this lab, we will get practice with estimation. In particular, we will use resampling to make guesses about the size of a population. We will learn to:

1. compare test statistics
2. use the boostrap method
3. generate confidence intervals

We will guide you through the code step-by-step to put these concepts into practice. We encourage you to contact us about any questions.

Collaboration Policy

Data science is a collaborative activity. While you may talk with others about the labs, we ask that you **write your solutions individually**.

In [1]:
```
# import some packages
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# change some settings

pd.set_option('display.max_rows', 10)
pd.set_option('display.max_columns', 8)

%matplotlib inline
plt.rcParams['figure.figsize'] = (10,8)

# add some helpful functions

from helper_functions import *
```

Sometimes we have to estimate unknown quantities about a population. If we cannot count the population, then the size of the population is an unknown quantity. So we need to make guesses based on samples. For example, how many piano tuners work in New York City?

Suppose we label the piano tuners in New York City from `1` to `N`. That means every piano tuner in New York City would have a unique number, starting from 1 and counting up sequentially until every tuner had a number, with the final tuner getting the number `N`, which would also be the total number of tuners.

How can we estimate the unknown quantity is `N`? Maybe we could try to make some deductions to make our estimate.

- Roughly how many people live in New York City? — 8,000,000
- Does every person own a piano? — no
- Can we assume that families own pianos, not individuals? — yes
- How large is the average family? — 5 people
- So how many families are there in NYC? — 1,600,000
- Does every family own a piano? — no… perhaps one in ten does
- So how many pianos are there in NYC? — 160,000
- How often per year do pianos need to be tuned? — once per year
- How many piano tunings can one piano tuner do? — let's say 4 per day, so if there's 200 working days in a year, that's 800 per year
- So how many piano tuners could NYC support?— 160,000/800 = 200 piano tuners

See this article (https://medium.com/@2michaeltaylor/how-many-piano-tuners-are-there-in-new-york-city-f421695c0653) for more information. Since we have been studying hypothesis testing, we want to take a different, more statistical approach. We will try to generate evidence for some guesses and against other guesses.

Question 1: Comparing Test Statistics

Sometimes we can make guesses by simulating many samples from a population. For each sample, we can generate an estimate. Other times, we lack the information needed for simulating samples from a population. Instead of looking to the population for more information, we can focus on the sample. We will try to make guesses by re-sampling from the sample.

We want to estimate the number of piano tuners in New York City. The piano tuners have labels from 1 to `N`. So `N` is the largest label. Imagine we have a sample of tuners, with their labels. In our sample, we only see a small number of labels (because our sample is only a small portion of the population). We can also assume that the sample came at random from the population.

Question 1a

Is `N` a parameter or a statistic?

(A) Parameter
(B) Statistic

In [2]:
```
q1a_answer = 'A'
```

Question 1b

Suppose we use our random sample to compute a number $\widehat{N}$ that estimates `N` (the total number of piano tuners). Is $\widehat{N}$ a parameter or a statistic?

(A) Parameter
(B) Statistic

In [3]:
```
q1b_answer = 'B'
```

Question 1c

We actually have a (simulated) sample that we can load the data from the file `piano_tuners.csv` . Let's load it.

In [4]:
```
csv_fpath = os.path.expanduser("~/shared/piano_tuners.csv")  # filepath
 of our data
observations = pd.read_csv(csv_fpath)
number_of_observations = len(observations)

observations
```

Out[4]:

	label
0	47
1	42
2	57
3	79
4	26
...	...
12	67
13	108
14	84
15	50
16	78

17 rows × 1 columns

Note that we have 17 observations and the maximum value in the `label` column is 135.

Let's try to calculate `N` using the maximum value in the `label` column. In order to do that, first we'll need a function that returns the maximum value from a column. Write a function called `max_based_estimator` that inputs an array of numbers and outputs the maximum value. You may find the `np.max` function helpful.

In [5]:
```
def max_based_estimator(array_of_numbers):
    max = np.max(array_of_numbers)
    return np.max(array_of_numbers)

max_based_estimator(observations["label"])
```

Out[5]: 135

Question 1d

Note that the average of the numbers $1, \ldots, N$ is

$$\frac{1}{N}(1 + 2 + \ldots + N - 1 + N) = \frac{(N+1)}{2}$$

So twice the average of the numbers $1, \ldots, N$ approximately equals N.

We could also try to estimate `N` by doubling the average of the values in the `label` column. Write a function called `mean_based_estimator` that inputs an array of numbers, calculates their average, and then returns a number that is double that average. It may be helpful to use the `np` library to calculate the mean.

In [6]:
```
def mean_based_estimator(array_of_numbers):
    avg = np.mean(array_of_numbers)*2
    return avg

mean_based_estimator(observations["label"])
```

Out[6]: 122.47058823529412

Question 1e

Before applying these statistics to the data, we should try some examples. Suppose the population consisted of labels `[1,2,3,4]`, and we randomly sample from this population to obtain the sample `[1,3]`. Here we've created a Python list for you, `my_sample`, containing `[1,3]`. Use this list and the two functions you wrote above to compute:

1. A test statistic for this sample based on the mean of the sample (call it `example_mean_based_estimate`) and
2. A test statistic for this sample based on the max of the sample (call it `example_max_based_estimate`).

When you have both of these computed, use `print()` to display the two test statistic values

In [8]:
```
my_sample = [1,3]

example_mean_based_estimate = mean_based_estimator(my_sample)
example_max_based_estimate = max_based_estimator(my_sample)

print(example_mean_based_estimate)
print(example_max_based_estimate)
```

```
4.0
3
```

Question 1f

In question 1e the mean-based estimate is greater than the max-based estimate. Will that always the case, for any sample of piano tuners? If yes, set `q1f_answer` to be `None`, otherwise set it to be a sample of piano tuners (so, a list of `int`s) for which the mean-based estimate is ***not*** greater than the max-based estimate, and use the `print()` function to show the result of computing `mean_based_estimator()` and `max_based_estimator()` on the sample you construct.

In [ ]:
```
q1f_answer = [1, 2, 4, 50]

example_mean_based_estimate = mean_based_estimator(q1f_answer)
example_max_based_estimate = max_based_estimator(q1f_answer)

print(example_mean_based_estimate)
print(example_max_based_estimate)
```

Question 2: Resampling

We would like to have access to a census of the piano tuners in New York City. Since we lack a census, we would like to go out into the population and take another sample. Unfortunately, we just have the one sample we have already loaded. So instead we can resample from our original sample.

Question 2a

In the first cell of this notebook, we imported all functions from `helper_functions.py`, so that we now have a helper function called `sample_with_replacement()`, which creates a bootstrapped sample from its input with replacement.

Take a look below for an example of how this function works (you can run it a few times to see the different outputs it produces) when we run it on our original sample of piano tuners (which we loaded into a DataFrame called `observations` above):

In [ ]:
```
resample = sample_with_replacement(number_of_observations, observations["label"])
resample
```

Which of the following statements are true?

1. The resample can contain numbers that are not in the original sample.
2. The original sample can contain numbers that are not in the resample.
3. The resample has either zero, one, or more than one copy of each number from 1 to `N`.
4. The original sample has exactly one copy of each number from 1 to `N`.

Assign `true_statements` to a list of the number(s) corresponding to correct statements. For example if you think that statements 1, 2, and 4 are true, define `true_statements` to be `[1,2,4]`.

In [ ]:
```
true_statements = [2, 3]
```

Question 2b

Now let's write a function to generate different resamples.

Since the process of resampling from a sample looks just like sampling from a population, the code should look almost the same. That means we can write a ***single*** block of code that will sample from a population ***or*** resample from a sample, depending on what we're trying to do.

In [ ]:
```
data = observations["label"] # The column in our DataFrame with the obse
rved piano tuner labels

sample_size = number_of_observations # The original sample size and the
 resample size we will use now

statistic = mean_based_estimator # We'll be using the mean based estimat
or

number_of_replications = 1000 # The number of times we will resample

# Below is a for loop to do this.
# Make sure you can follow the code!
estimates = []

for replication in range(number_of_replications):

    new_data = sample_with_replacement(sample_size, data)

    estimates.append(statistic(new_data))

estimates[:10] # Show just the first 10 estimates, so we can check that
 it worked
```

Above we used the `sample_with_replacement` function from Question 2a inside the for loop. We generated 1000 resamples and for each resample, we computed the mean-based statistic and then stored this mean-based statistic in the list `estimates`.

Now we want you to write a ***function*** (rather than just a code cell) that we can call to generate these estimates for a given set of parameters passed as arguments to the function. This function will make it convenient to do different simulations.

Using the code we wrote above (if you want!), write a function called `bootstrap_estimates`. It should have four inputs

1. `data`: The data for resampling
2. `sample_size`: The size of the resamples
3. `statistic`: A function that computes a statistic from an array of numbers
4. `number_replications`: The number of replications

It should output a list of estimates calculated from the statistics on each resample, just like the list `estimates` above.

Simply put, convert the cell above this one into a reusable function.

In [ ]:
```
def bootstrap_estimates(data, sample_size, statistic, number_of_replicat
ions):
    estimates = []
    for replication in range(number_of_replications):
        # Your code here
        new_data = sample_with_replacement(sample_size, data)
        estimates.append(statistic(new_data))

    return estimates
```

Question 2c

Now use your function `bootstrap_estimates` to generate estimates from 1000 resamples. Use all the inputs we have defined for you here (`data`, `sample_size`, `statistic`, and `number_of_replications`) as arguments to your function and assign the output of the function to a variable called `bootstrap_mean_based_estimates`.

Importantly, note that we're setting `statistic` to be `mean_based_estimator`, so that the statistics we end up with will be the mean-based estimates of `N`.

In [ ]:
```
### Inputs we define
data = observations["label"] #the data we want to use

sample_size = number_of_observations # the sample size for the resample

statistic = mean_based_estimator # the estimator we want to use

number_of_replications = 1000 #the number of replications we want to do

# Your code here
bootstrap_mean_based_estimates = bootstrap_estimates(data, sample_size,
statistic, number_of_replications)

bootstrap_mean_based_estimates
```

We have provided code that plots a histogram to visualize the simulated estimates. Just run the cell below to see it.

In [ ]:
```
plt.hist(bootstrap_mean_based_estimates, density = True, rwidth = 0.97,
label = "Estimates")
plt.title(r"Mean Based Estimate of N")
plt.legend()
plt.show()
```

Question 2d

Now, repeat Question 2c using the max-based estimate of `N` instead of the mean-based estimate, again printing the first 10 estimates to make sure that the computation worked.

In [ ]:
```
### Inputs we define
data = observations["label"]

sample_size = number_of_observations

number_of_replications = 1000

#your code
statistic = max_based_estimator

bootstrap_max_based_estimates = bootstrap_estimates(data, sample_size, s
tatistic, number_of_replications)

#print(bootstrap_max_based_estimates[:10])
bootstrap_max_based_estimates[:10]
```

As before, have provided code that plots a histogram to visualize the simulated estimates.

In [ ]:
```
plt.hist(bootstrap_max_based_estimates, density = True, rwidth = 0.97, l
abel = "Estimates")
plt.title(r"Max Based Estimate of N")
plt.legend()
plt.show()
```

Question 2e

Can the max-based statistic generate an estimate greater than 135 on a resample?

(A) Yes
(B) No

In [ ]:
```
q2e_answer = 'B'
```

3. Computing Confidence Intervals

Suppose we scour Yelp (https://www.yelp.com/search?find_desc=piano+tuner&find_loc=New+York%2C+NY&ns=1) to determine the number of piano tuners in New York City.

We find that

$$N = 150$$

which we'll assume is the truth, meaning the actual number of piano tuners in the city is 150. Knowing the value of N, we want to evaluate estimates in the bootstrap method. Specifically, we want to see how well our bootstrap methods actually work now that we know the truth.

Question 3a

We want to compute an interval that covers 90% of the mean based bootstrap estimates between the 5th percentile and the 95th percentile. Use the helper function `calculate_percentile` to compute the 5th percentile and the 95th percentile. Call the 5th percentile `left_end` and the 95th percentile `right_end`, and use the `print()` function to display the value of both at the end of the code cell.

Importantly the function `calculate_percentile` takes two inputs in the following order:

1. An array of numbers
2. A percentile (specifically, an `int` between 0 and 100)

and outputs the number that corresponds to that percentile in that list.

In [ ]:
```
left_end = calculate_percentile(bootstrap_mean_based_estimates, 5)
right_end = calculate_percentile(bootstrap_mean_based_estimates, 95)
print(left_end)
print(right_end)
```

Verify that your interval appears to cover 90% of the total area in the histogram by running the cell below. The yellow bar is the area demarcated by your 5th to 95 percentile.

Also, note that the red dot on the histogram is the value of `N`.

In [ ]:
```
plt.hist(bootstrap_mean_based_estimates, density = True, rwidth = 0.97,
label = r"Estimates")

plt.hlines(y=0, xmin=left_end, xmax=right_end, color='yellow', lw=7, zor
der=1)
plt.scatter(150, 0, color='red', s=50, zorder=2)

plt.title("Mean Based Estimates")
plt.legend()
plt.show()
```

Question 3b

Imagine we did not know the actual truth `N`. In this case, based on the bootstrap from one sample, we generated a [5th percentile, 95th percentile] confidence interval.

If we had another random sample and did a bootstrap on that sample to generate a [1st percentile, 99th percentile] confidence interval, would that confidence interval, on average, be more or less likely to contain `N`?

(A) More likely
(B) Less likely
(C) Impossible to tell

In [ ]:
```
q3b_answer = 'A'
```

Question 3c

Suppose we have 100 samples of size 17 from the population. We apply bootstrap resampling with 1000 replications for each sample. From the 1000 resamples we can compute the 5th percentile and 95th percentile of the mean based statistic. The 5th percentile and 95th percentile determine a confidence interval for the sample.

Among the 100 confidence intervals (from our 100 samples of size 17), approximately how many should we expect to contain `N`?

(A) 5%
(B) 95%
(C) 10%
(D) 90%

In [ ]:
```
q3c_answer = 'D'
```

Congratulations, you completed lab 8! Please remember to submit it on both Jupyter Hub and Gradescope!