

Resampling and the Bootstrap

Welcome to lab 6!

In textbook section [9.3 \(https://www.inferentialthinking.com/chapters/09/3/empirical-distribution-of-a-statistic.html\)](https://www.inferentialthinking.com/chapters/09/3/empirical-distribution-of-a-statistic.html), we saw an example of *estimation*. The British Royal Air Force wanted to know how many warplanes the Germans had (some number N , a *population parameter*), and they needed to estimate that quantity knowing only a random sample of the planes' serial numbers (from 1 to N). For example, one estimate was twice the mean of the sample serial numbers.

We investigated the random variation in these estimates by simulating sampling from the population many times and computing estimates from each sample. In real life, if the RAF had known what the population looked like, they would have known N and would not have had any reason to think about random sampling. They didn't know what the population looked like, so they couldn't have run the simulations we did. So that was useful as an exercise in *understanding random variation* in an estimate, but not as a tool for practical data analysis.

Now we'll flip that idea on its head to make it practical. Given *just* a random sample of serial numbers, we'll estimate N , and then we'll use simulation to find out how accurate our estimate probably is, without ever looking at the whole population. This is an example of *statistical inference*.

As usual, **run the cell below** to prepare the lab and the automatic tests.

```
In [39]: # 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
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
import warnings
warnings.simplefilter('ignore', FutureWarning)

# These lines load the tests.
from client.api.assignment import load_assignment
tests = load_assignment('lab06.ok')
```

1. Preliminaries

Remember the setup: We (the RAF in World War II) want to know the number of warplanes fielded by the Germans. That number is N . The warplanes have serial numbers from 1 to N , so N is also equal to the largest serial number on any of the warplanes.

We only see a small number of serial numbers (assumed to be a random sample with replacement from among all the serial numbers), so we have to use estimation.

Question 1.1

Is N a population parameter or a statistic? If we compute a number using our random sample that's an estimate of N , is that a population parameter or a statistic?

Write your answer here, replacing this text.

Check your answer with a neighbor or a TA.

To make the situation realistic, we're going to hide the true number of warplanes from you. You'll have access only to this random sample:

```
In [40]: observations = Table.read_table("serial_numbers.csv")
num_observations = observations.num_rows
observations
```

Question 1.2

Define a function named `plot_serial_numbers` to make a histogram of any table of serial numbers. It should take one argument, a table like `observations` with one column called "serial number". It should make a histogram **using bars of width 1** ranging from **1 to 200**. It should return nothing. Then, call that function to make a histogram of `observations`.

```
In [41]: def plot_serial_numbers(numbers):
...
    # Assuming the lines above produce a histogram, this next
    # line may make your histograms look nicer. Feel free to
    # delete it if you want.
    plt.ylim(0, .25)

plot_serial_numbers(observations)
```

Question 1.3

What does each little bar in the histogram represent?

Write your answer here, replacing this text.

We saw that one way to estimate N was to take twice the mean of the serial numbers we see.

Question 1.4

Write a function that computes that statistic. It should take as its argument an array of serial numbers and return twice their mean. Call it `mean_based_estimator`. Use it to compute an estimate of N called `mean_based_estimate`.

```
In [42]: def mean_based_estimator(nums):  
        ...  
  
        mean_based_estimate = ...  
        mean_based_estimate
```

```
In [43]: _ = tests.grade('q1_4')
```

Question 1.5

We also estimated N using the biggest serial number in the sample. Compute it, giving it the name `max_estimate`.

```
In [44]: max_estimate = ...  
        max_estimate
```

```
In [45]: _ = tests.grade('q1_5')
```

Question 1.6

Look at the values of `max_estimate` and `mean_based_estimate` that we happened to get for our dataset. The value of `max_estimate` tells you something about `mean_based_estimate`. Can it be equal to N (at least if we round it to the nearest integer)? If not, is it definitely higher, definitely lower, or can we not tell? Can you make a statement like "`mean_based_estimate` is at least *[fill in a number]* away from N "?

Write your answer here, replacing this text.

Check your answer with a neighbor or a TA.

We can't just confidently proclaim that `max_estimate` or `mean_based_estimate` is equal to `N`. What if we're really far off? So we want to get a sense of the accuracy of our estimates.

In section 9.3, we ran a simulation like this:

```
In [13]: # ???
N = ...

# Attempts to simulate one sample from the population of all serial
# numbers, returning an array of the sampled serial numbers.
def simulate_observations():
    # You'll get an error message if you try to call this
    # function, because we didn't define N properly!
    serial_numbers = Table().with_column("serial number", np.arange(1,
N+1))
    return serial_numbers.sample(num_observations)

estimates = make_array()
for i in np.arange(5000):
    estimate = mean_based_estimator(simulate_observations())
    estimates = np.append(estimates, estimate)

Table().with_column("mean-based estimate", estimates).hist()
```

Since we don't know what the population looks like, we don't know `N`, and we can't run that simulation.

Question 1.7

Using the terminology you've learned, describe the kind of histogram that cell would have made if we had filled in `N`. If that histogram is an approximation to something, say what it approximates.

Write your answer here, replacing this text.

Check your answer with a neighbor or a TA.

2. Resampling

Instead, we'll use resampling. That is, we won't exactly simulate the observations the RAF would have really seen. Rather we sample from our sample, or "resample."

Why does that make any sense?

When we tried to estimate N , we would have liked to use the whole population. Since we had only a sample, we used that to estimate N instead.

This time, we would like to use the population of serial numbers to *run a simulation* about estimates of N . But we still only have our sample. We use our sample in place of the population to run the simulation.

So there is a simple analogy between estimating N and simulating the variability of estimates.

computing N from the population	
:	
computing an estimate of N from a sample	
as	
simulating the distribution of estimates of N using samples from the population	
:	
simulating an (approximate) distribution of estimates of N using resamples from a sample	

Question 2.1

Write a function called `simulate_resample`. It should generate a resample from the observed serial numbers in `observations` and return that resample. (The resample should be a table like `observations`.) It should take no arguments.

```
In [15]: def simulate_resample():  
        ...
```

Let's make one resample.

```
In [16]: # This is a little magic to make sure that you see the same results  
        # we did.  
        np.random.seed(123)  
  
        one_resample = simulate_resample()  
        one_resample
```

Later, we'll use many resamples at once to see what estimates typically look like. We don't often pay attention to single resamples, so it's easy to misunderstand them. Let's examine some individual resamples before we start using them.

Question 2.2

Make a histogram of your resample using the plotting function you defined earlier in this lab, **and** a separate histogram of the original observations.

```
In [17]: ...  
...
```

Question 2.3

Which of the following are true:

1. In the plot of the resample, there are no bars at locations that weren't there in the plot of the original observations.
2. In the plot of the original observations, there are no bars at locations that weren't there in the plot of the resample.
3. The resample has exactly one copy of each serial number.
4. The sample has exactly one copy of each serial number.

Write your answer here, replacing this text. Discuss your answers with a neighbor or TA.

Question 2.4

Create 2 more resamples. For each one, plot it, compute the max- and mean-based estimates using that resample, and find those estimates on the horizontal axis of the plot.

```
In [21]: resample_0 = ...  
...  
mean_based_estimate_0 = ...  
max_based_estimate_0 = ...  
print("Mean-based estimate for resample 0:", mean_based_estimate_0)  
print("Max-based estimate for resample 0:", max_based_estimate_0)  
  
resample_1 = ...  
...  
mean_based_estimate_1 = ...  
max_based_estimate_1 = ...  
print("Mean-based estimate for resample 1:", mean_based_estimate_1)  
print("Max-based estimate for resample 1:", max_based_estimate_1)
```

You may find that the max-based estimates from the resamples are both exactly 135. You will probably find that the two mean-based estimates do differ from the sample mean-based estimate (and from each other).

Question 2.5

Using the probability theory you've learned, compute the exact chance that a max-based estimate from *one* resample is 135. Using your intuition, explain why a mean-based estimate from a resample is less often exactly equal to the mean-based estimate from the original sample.

Write your answer here, replacing this text.

Discuss your answers with a neighbor or TA. If you have difficulty with the probability calculation, work with someone or ask for help; don't stay stuck on it for too long.

3. Simulating with resampling

Since resampling from a sample looks just like sampling from a population, the code should look almost the same. That means we can write a function that simulates either sampling from a population or resampling from a sample. If we pass it a population as its argument, it will do the former; if we pass it a sample, it will do the latter.

Question 3.1

Write a function called `simulate_estimates`. It should take 4 arguments:

1. A table from which the data should be sampled. The table will have 1 column named "serial number".
2. The size of each from that table, an integer. (For example, to do resampling, we would pass for this argument the number of rows in the table.)
3. A function that computes a statistic of a sample. This argument is a *function* that takes an array of serial numbers as its argument and returns a number.
4. The number of replications to perform.

It should simulate many samples with replacement from the given table. (The number of samples is the 4th argument.) For each of those samples, it should compute the statistic on that sample, and it should return an array containing each of those statistics. The code below provides an example use of your function and describes how you can verify that you've written it correctly.

```
In [30]: def simulate_estimates(original_table, sample_size, statistic, num_repli
cations):
    # Our implementation of this function took 5 short lines of code.
    ...

    # This should generate an empirical histogram of twice-mean estimates
    # of N from samples of size 50 if N is 1000. This should be a bell-shap
    ed
    # curve centered at 1000 with most of its mass in [800, 1200]. To verif
    y your
    # answer, make sure that's what you see!
    example_estimates = simulate_estimates(
        Table().with_column("serial number", np.arange(1, 1000+1)),
        50,
        mean_based_estimator,
        10000)
    Table().with_column("mean-based estimate",
        example_estimates).hist(bins=np.arange(0, 1500, 25))
```

Question 3.2

Was the example in the previous cell performing a bootstrap simulation (i.e. resampling from a sample) or an ordinary simulation (sampling from some population) of the kind we saw in chapter 9? What was the sample or population?

Write your answer here, replacing this text.

Now we can go back to the sample we actually observed (the table observations) and estimate how much our mean-based estimate of N would have varied from sample to sample.

Question 3.3

Using the bootstrap and the sample observations, simulate the approximate distribution of *mean-based estimates* of N . Use 5,000 replications. To visualize the simulated estimates, **make a histogram** of them. We suggest using bins of width around 4.

```
In [32]: bootstrap_estimates = ...
    ...
```

Question 3.4

Compute an interval that covers the middle 95% of the bootstrap estimates. Verify that your interval looks like it covers 95% of the area in the histogram above.


```
In [34]: left_end = ...
         right_end = ...
         print("Middle 95% of bootstrap estimates: [{:f}, {:f}]"
               .format(left_end,
                       right_end))
```

Question 3.5

Your mean-based estimate of N should have been around 122. Given the above calculations, is it likely that N is exactly 122? Quantify the amount of error in the estimate by making a statement like this:

"Assuming the population looks similar to the sample, the *difference* between N and mean-based estimates of N from samples of size 17 is typically in the range [A NUMBER, ANOTHER NUMBER]."

Write your answer here, replacing this text.

Question 3.6

N was actually 150! Write code that simulates the sampling and bootstrapping process again, as follows:

1. Generate a new set of random observations the RAF might have seen, following the procedure laid out at the start of this lab.
2. Compute an estimate of N from these new observations, using `mean_based_estimator`.
3. Using only the new observations, compute 5,000 bootstrap estimates of N .
4. Plot these bootstrap estimates and compute an interval covering the middle 95%.

```
In [38]: population = Table().with_column("serial number", np.arange(1, 150+1))

         new_observations = ...
         new_mean_based_estimate = ...
         new_bootstrap_estimates = ...
         ...
         new_left_end = ...
         new_right_end = ...

         print("Middle 95% of bootstrap estimates: [{:f}, {:f}]"
               .format(new_left_end, new_right_end))
```

Question 3.7

Does the interval covering the middle 95% of the new bootstrap estimates include N ? If you ran that cell many times, would it always include N ?

Write your answer here, replacing this text.

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

```
In [47]: # Run this cell to submit your work *after* you have passed all of the test cells.
# It's ok to run this cell multiple times. Only your final submission will be scored.

!TZ=America/Los_Angeles jupyter nbconvert --output=".lab06_$(date +%m%d_%H%M)_submission.html" lab06.ipynb && echo "Submitted successfully!"
```