Resampling and the Bootstrap

Welcome to lab 6!

In class (Lecture 16), 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*.

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
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
import warnings
warnings.simplefilter('ignore', FutureWarning)
```

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?

`N` is the population parameter because it is the number describing the whole population.

Check your answer with a neighbor or Kenneth.

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 [3]:
```
observations = Table.read_table("~/DS_113_S23/Labs/Lab_6/serial_numbers.csv'
num_observations = observations.num_rows
observations
```

Out[3]:

serial number
47
42
57
79
26
23
36
64
83
135

... (7 rows omitted)

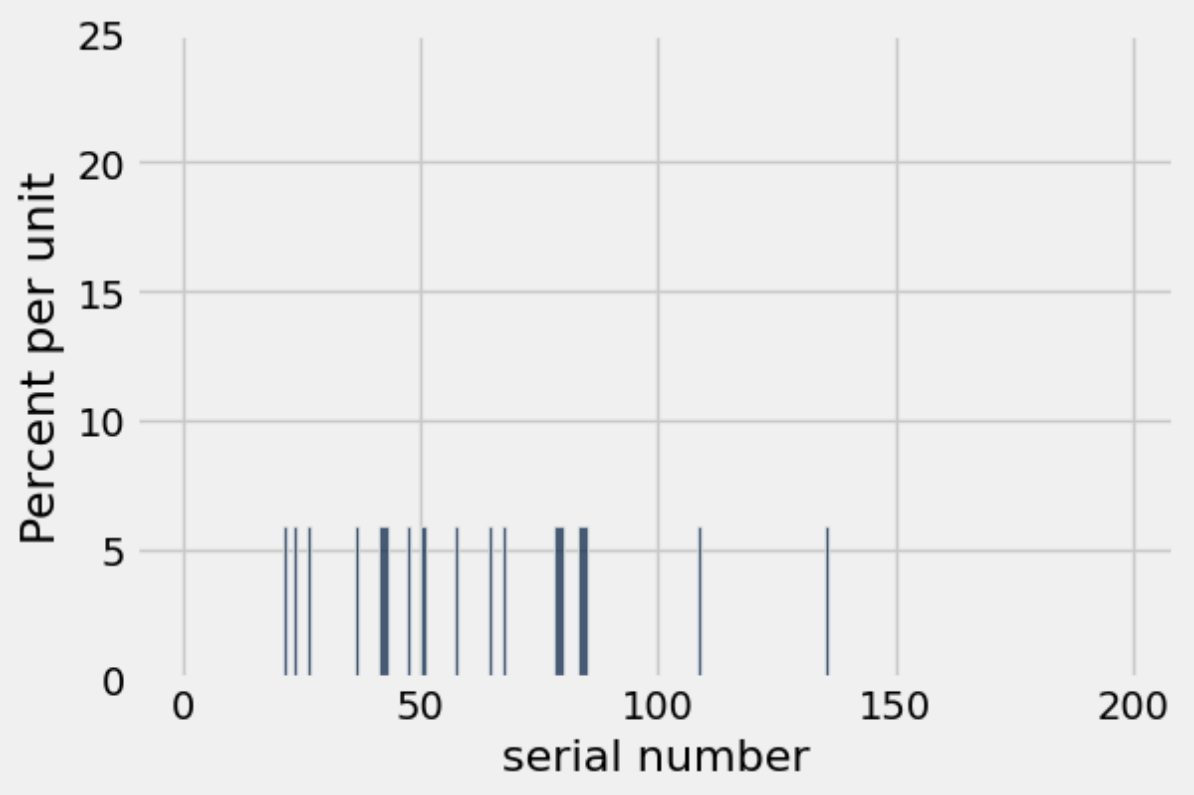
Question 1.2

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 [4]:
```
def plot_serial_numbers(numbers):
    '''take a table and make a histogram'''
    numbers.hist(bins=np.arange(0,200,1))

    # 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?

Each bar represents there is a plane for that serial code.

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 [5]:
```
def mean_based_estimator(nums):
    '''take an array and return twice their mean'''
    mean = np.average(nums)
    return 2*mean

mean_based_estimate = mean_based_estimator(observations[0])
mean_based_estimate
```

Out[5]:
```
122.47058823529412
```

Question 1.5

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

In [6]:
```
max_estimate = max(observations[0])
max_estimate
```

Out[6]: 135

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`"?

As we look at the histogram, most of the data are crowded in between 20 to 80. We can expect that the `mean_based_estimate` would be smaller than the `max_estimate`. Therefore, "`mean_based_estimate` is at least *12.53* away from `N`.

Check your answer with a neighbor or Kenneth.

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 Lecture 16, we ran a simulation like this:

In [7]:
```
# ???
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()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In [7], line 14
     12 estimates = make_array()
     13 for i in np.arange(5000):
---> 14     estimate = mean_based_estimator(simulate_observations())
     15     estimates = np.append(estimates, estimate)
     17 Table().with_column("mean-based estimate", estimates).hist()

Cell In [7], line 9, in simulate_observations()
      6 def simulate_observations():
      7     # You'll get an error message if you try to call this
      8     # function, because we didn't define N properly!
----> 9     serial_numbers = Table().with_column("serial number", np.arange
(1, N+1))
     10     return serial_numbers.sample(num_observations)

TypeError: unsupported operand type(s) for +: 'ellipsis' and 'int'
```

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.

The histogram will be a bell-shaped histogram with most of the data in centered around `N`.

Check your answer with a neighbor or Kenneth.

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.

$$\text{computing } N \text{ from the population}$$

$$:$$

$$\text{computing an estimate of } N \text{ from a sample}$$

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

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 [8]:
```
def simulate_resample():
    resample = observations.sample()
    return resample
```

Let's make one resample.

In [9]:
```
# 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
```

Out[9]:

serial number
108
57
57
36
41
42
47
50
135
47

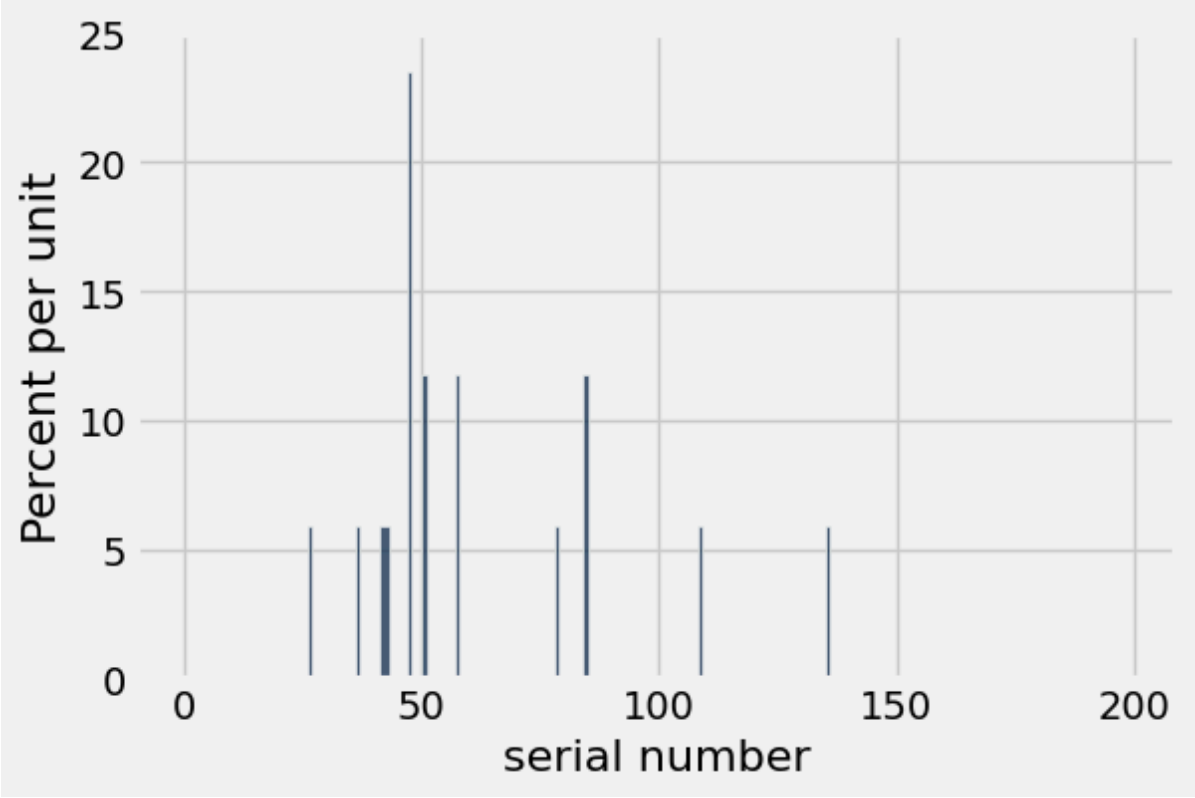
... (7 rows omitted)

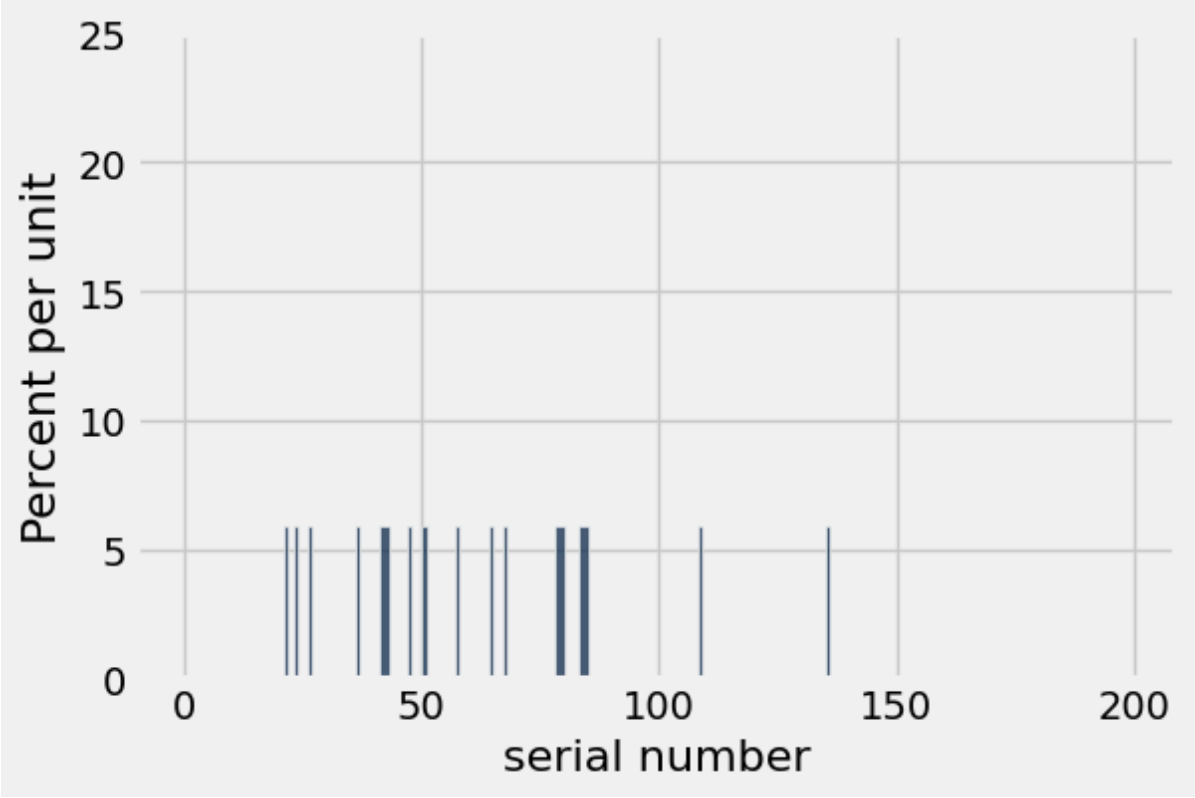
Later, we'll use many resamples at once to see what estimates typically look like. We don't often pay attention to single resamples, so it's easy to misunderstand them. Let's examine some individual resamples before we start using them.

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 [10]:
```
plot_serial_numbers(one_resample)
plot_serial_numbers(observations)
```





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.

1. True
2. False
3. False
4. True

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 [12]:
```
resample_0 = simulate_resample()

mean_based_estimate_0 = mean_based_estimator(resample_0[0])
max_based_estimate_0 = max(resample_0[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 = simulate_resample()

mean_based_estimate_1 = mean_based_estimator(resample_1[0])
max_based_estimate_1 = max(resample_1[0])
print("Mean-based estimate for resample 1:", mean_based_estimate_1)
print("Max-based estimate for resample 1:", max_based_estimate_1)
```

```
Mean-based estimate for resample 0: 126.352941176
Max-based estimate for resample 0: 135
Mean-based estimate for resample 1: 128.352941176
Max-based estimate for resample 1: 135
```

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

See if you can 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.

The resample has some serial numbers that are repeated. Therefore, the mean of the resample would be different with the mean of the original sample.

Discuss your answers with a neighbor or Kenneth. 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

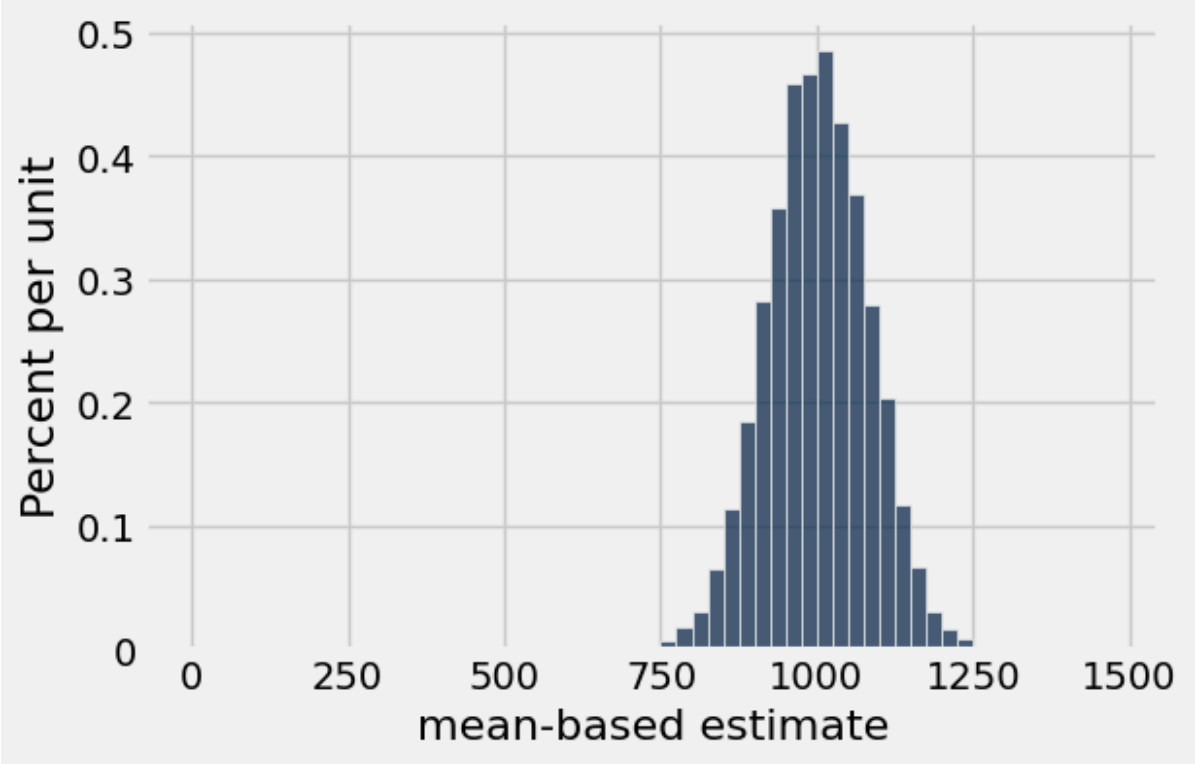
Write a function called `simulate_estimates`. It should take 4 arguments:

1. A table from which the data should be sampled. The table will have 1 column named `"serial number"`.
2. The size of each sample 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 a framework for your function and is followed by code that you can use to verify that you've written it correctly.

In [13]:
```
def simulate_estimates(original_table, sample_size, statistic, num_replicati
    '''simulate many samples, compute the statistic, and return an array wit
    # Our implementation of this function took 5 short lines of code.
    stat_array = make_array()
    for i in np.arange(num_replications):
        resample = original_table.sample(sample_size)
        stat = statistic(resample["serial number"])
        stat_array = np.append(stat_array, stat)
    return stat_array

# 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-shaped
# curve centered at 1000 with most of its mass in [800, 1200].  To verify yo
# 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.a
```



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 Lecture 16? What was the sample or population?

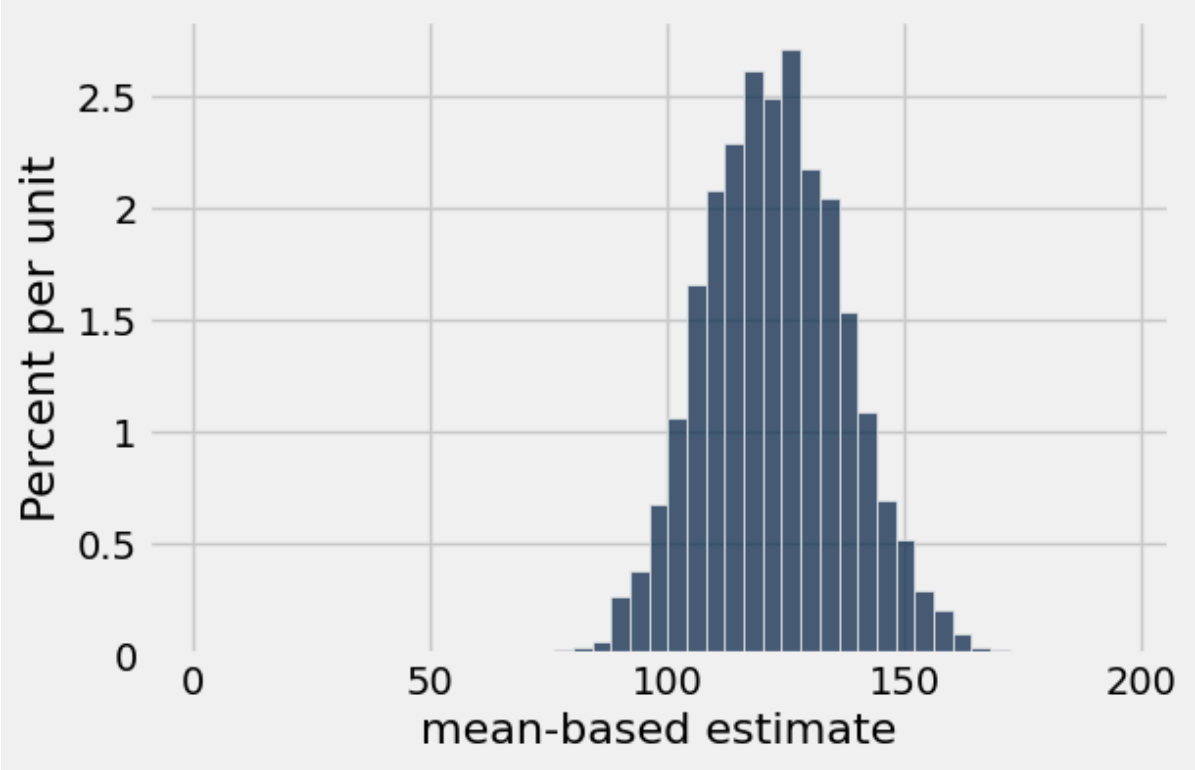
The example is an ordinary simulation. The population was the serial numbers from 1 to 1000.

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 [14]:
```
bootstrap_estimates = simulate_estimates(observations,
                                         num_observations,
                                         mean_based_estimator,
                                         5000)
Table().with_column("mean-based estimate", bootstrap_estimates).hist(bins=np
```



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 [15]:
```
left_end = percentile(2.5, bootstrap_estimates)
right_end = percentile(97.5, bootstrap_estimates)
print("Middle 95% of bootstrap estimates: [{:f}, {:f}]".format(left_end, rig
```

```
Middle 95% of bootstrap estimates: [94.823529, 152.000000]
```

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]."

In [30]:
```
left_end-122
```

Out[30]:
```
-27.17647058823529
```

In [31]:
```
right_end-122
```

Out[31]:
```
30.0
```

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 [-27.176, 30]

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%.

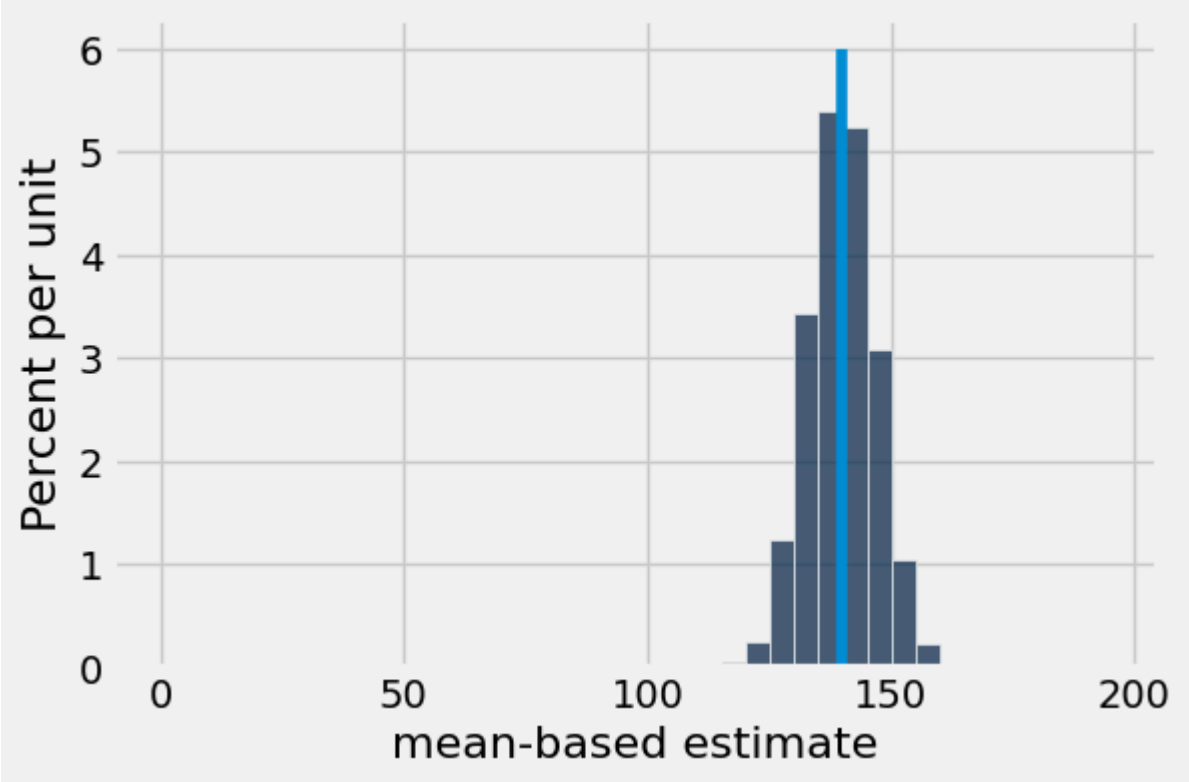
In [29]:
```
population = Table().with_column("serial number", np.arange(1, 150+1))

new_observations = population.sample()
new_mean_based_estimate = mean_based_estimator(new_observations["serial numb
new_bootstrap_estimates = simulate_estimates(new_observations,
                                             new_observations.num_rows,
                                             mean_based_estimator,
                                             5000)

new_left_end = percentile(2.5, new_bootstrap_estimates)
new_right_end = percentile(97.5, new_bootstrap_estimates)

print("Middle 95% of bootstrap estimates: [{:f}, {:f}]".format(new_left_end,
Table().with_column("mean-based estimate", new_bootstrap_estimates).hist(bir
_ = plt.plot([new_mean_based_estimate, new_mean_based_estimate], [0, 0.06])
```

```
Middle 95% of bootstrap estimates: [126.346667, 152.800000]
```



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`?

The estimates would include 150, which is `N`. However, it will not always include `N` in the interval of the middle 95%.