# hw09_master

November 14, 2019

## 1 Homework 9: Resampling and the Bootstrap

**Reading**: * Estimation

Please complete this notebook by filling in the cells provided. Before you begin, execute the following cell to load the provided tests. Each time you start your server, you will need to execute this cell again to load the tests.

Homework 9 is due **Thursday, 11/7 at 11:59pm**. You will receive an early submission bonus point if you turn in your final submission by Wednesday, 11/6 at 11:59pm. Start early so that you can come to office hours if you're stuck. Check the website for the office hours schedule. Late work will not be accepted as per the policies of this course.

Directly sharing answers is not okay, but discussing problems with the course staff or with other students is encouraged. Refer to the policies page to learn more about how to learn cooperatively.

For all problems that you must write our explanations and sentences for, you **must** provide your answer in the designated space. Moreover, throughout this homework and all future ones, please be sure to not re-assign variables throughout the notebook! For example, if you use `max_temperature` in your answer to one question, do not reassign it later on.

As usual, **run the cell below** to import modules and autograder tests.

[ ]:
```
# 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.notebook import Notebook
ok = Notebook('hw09.ok')
```

```
_ = ok.submit()
```

## 1.1 1. Preliminaries

The British Royal Air Force wanted to know how many warplanes the Germans had (some number `N`, which is a *parameter*), and they needed to estimate that quantity knowing only a random sample of the planes' serial numbers (from 1 to `N`). We know that the German's warplanes are labeled consecutively from 1 to `N`, so `N` would be the total number of warplanes they have.

We normally investigate the random variation among our estimates by simulating a sampling procedure from the population many times and computing estimates from each sample that we generate. 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. However, they didn't know what the population looked like, so they couldn't have run the simulations that we normally do.

Simulating a sampling procedure many times was a useful exercise in *understanding random variation* for an estimate, but it's not as useful as a tool for practical data analysis.

Let's flip that sampling 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*.

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 use our random sample to compute a number that is an estimate of `N`, is that a population parameter or a statistic?

Set `N` and `N_estimate` to either the string `"parameter"` or `"statistic"` to indicate whether each value is a parameter or a statistic.

```
BEGIN QUESTION
name: q1_1
```

[6]:
```
N = "parameter" # SOLUTION
N_estimate = "statistic" # SOLUTION
```

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:

[12]:
```
observations = Table.read_table("serial_numbers.csv")
num_observations = observations.num_rows
observations
```

[12]:
| serial number |
|---|
| 47 |
| 42 |
| 57 |
| 79 |
| 26 |
| 23 |
| 36 |
| 64 |
| 83 |
| 135 |

… (7 rows omitted)

**Question 1.2** The average of the sample is about half of `N`. So one way to estimate `N` is to take twice the mean of the serial numbers we see. Write a function that computes that statistic. It should take as its argument an array of serial numbers and return twice their mean. Call the function `mean_based_estimator`.

After that, use it to compute an estimate of `N` called `mean_based_estimate`.

```
BEGIN QUESTION
name: q1_2
```

[13]:
```
def mean_based_estimator(nums):
    return 2*np.average(nums) # SOLUTION

mean_based_estimate = mean_based_estimator(observations.column(0)) # SOLUTION
mean_based_estimate
```

[13]: 122.47058823529412

**Question 1.3** We can also estimate `N` by using the biggest serial number in the sample. Compute this value and give it the name `max_estimate`.

```
BEGIN QUESTION
name: q1_3
```

[17]:
```
max_estimate = max(observations.column(0)) # SOLUTION
max_estimate
```

[17]: 135

**Question 1.4** Let's take a look at the values of `max_estimate` and `mean_based_estimate` that we got for our dataset. Which of these values is closer to the true population maximum `N`? Based off of our estimators, can we give a lower bound for what `N` must be? In other words, is there a value that `N` must be greater than or equal to?

BEGIN QUESTION
name: q1_4
manual: true

**SOLUTION:** Based off our data, `max_esimate` is closer to the true population maximum. `max_estimate` can never be more than `N`, so `N` is at least 135.

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

## 1.2 2. Resampling

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

Why does that make any sense?

When we try to find the value of a population parameter, we ideally would like to use the whole population. However, we often only have access to one sample and we must use that to estimate the parameter instead.

Here, we would like to use the population of serial numbers to draw more samples and run a simulation about estimates of `N`. But we still only have our sample. So, we **use our sample in place of the population** to run the simulation. We resample from our original sample with replacement as many times as there are elements in the original sample. This resampling technique is called *bootstrapping*.

Note that in order for bootstrapping to work well, you must start with a large, random sample. Then the Law of Averages says that with high probability, your sample is representative of the population.

**Question 2.1** Write a function called `simulate_resample`. The function should take no arguments and generate a resample from the observed serial numbers in `observations`. The resample should be a table with the same column names as `observations`.

BEGIN QUESTION
name: q2_1

[21]:
```
def simulate_resample():
    return observations.sample(observations.num_rows, with_replacement = True)␣
↪#SOLUTION
```

We'll use many resamples at once to see what estimates typically look like. However, we don't often pay attention to single resamples, so it's easy to misunderstand them. Let's first answer some questions about our resample.

**Question 2.2** Which of the following statements are true?

1. The resample can contain serial numbers that are not in the original sample.
2. The original sample can contain serial numbers that are not in the resample.

3. The resample has either zero, one, or more than one copy of each serial number.
4. The original sample has exactly one copy of each serial number.

Assign `true_statements` to an array of the number(s) corresponding to correct statements.

```
BEGIN QUESTION
name: q2_2
```

[26]:
```
true_statements = make_array(2, 3) #SOLUTION
```

Now let's write a function to do many resamples at once.

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 the process of either sampling from a population or resampling from a sample. If we pass in population as its argument, it will do the former; if we pass in a sample, it will do the latter.

**Question 2.3** Write a function called `sample_estimates`. It should take 4 arguments: 1. `serial_num_tbl`: A table from which the data should be sampled. The table will have one column named `serial number`. 2. `sample_size`: The size of each sample from that table, an integer. 3. `statistic`: A *function* that takes in an array of serial numbers as its argument and computes a statistic from the array (i.e. returns a calculated number). 4. `num_replications`: The number of simulations to perform.

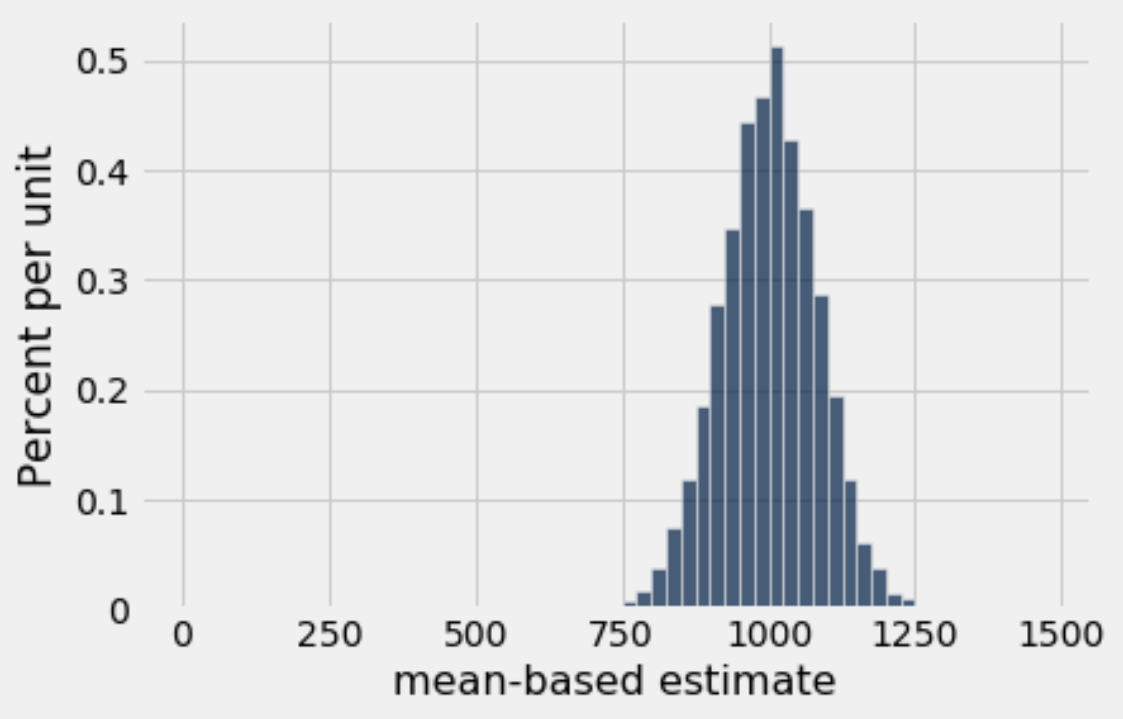
The function should simulate many samples **with replacement** from the given table. For each of those samples, it should compute the statistic on that sample. Then it should **return an array** containing each of those statistics. The code below provides an example use of your function and describes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

```
BEGIN QUESTION
name: q2_3
```

[30]:
```
def sample_estimates(serial_num_tbl, sample_size, statistic, num_replications):
    # BEGIN SOLUTION
    stats = make_array()
    for i in np.arange(num_replications):
        s = statistic(serial_num_tbl.sample(sample_size).column("serial␣
 ↪number"))
        stats = np.append(stats, s)
    return stats
    # END SOLUTION

# Don't change the code below this comment!
# This should generate an empirical histogram of twice-mean-based 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 your
# answer, make sure that's what you see!
population = Table().with_column("serial number", np.arange(1, 1000+1))
example_estimates = sample_estimates(
    population,
```

```
    50,
    mean_based_estimator,
    10000)
Table().with_column("mean-based estimate", example_estimates).hist(bins=np.
 ↪arange(0, 1500, 25))
```




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.

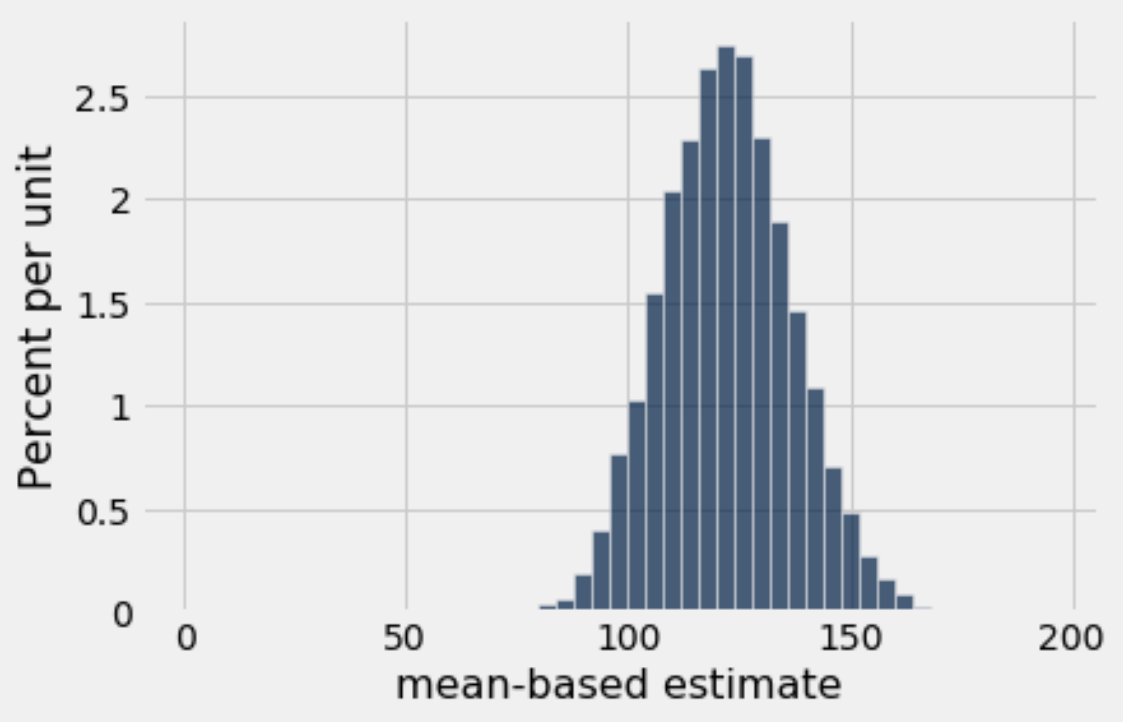
**Question 2.4** Using the bootstrap and the sample `observations`, simulate the approximate distribution of *mean-based estimates* of `N`. Use 10,000 replications and save the estimates in an array called `bootstrap_mean_based_estimates`.

We have provided code that plots a histogram, allowing you to visualize the simulated estimates.

`BEGIN QUESTION`
`name: q2_4`

[35]:
```
bootstrap_mean_based_estimates = sample_estimates(observations,␣
 ↪num_observations, mean_based_estimator, 10000) # SOLUTION

# Don't change the code below! This plots bootstrap_mean_based_estimates.
Table().with_column("mean-based estimate", bootstrap_mean_based_estimates).
 ↪hist(bins=np.arange(0, 200, 4))
```



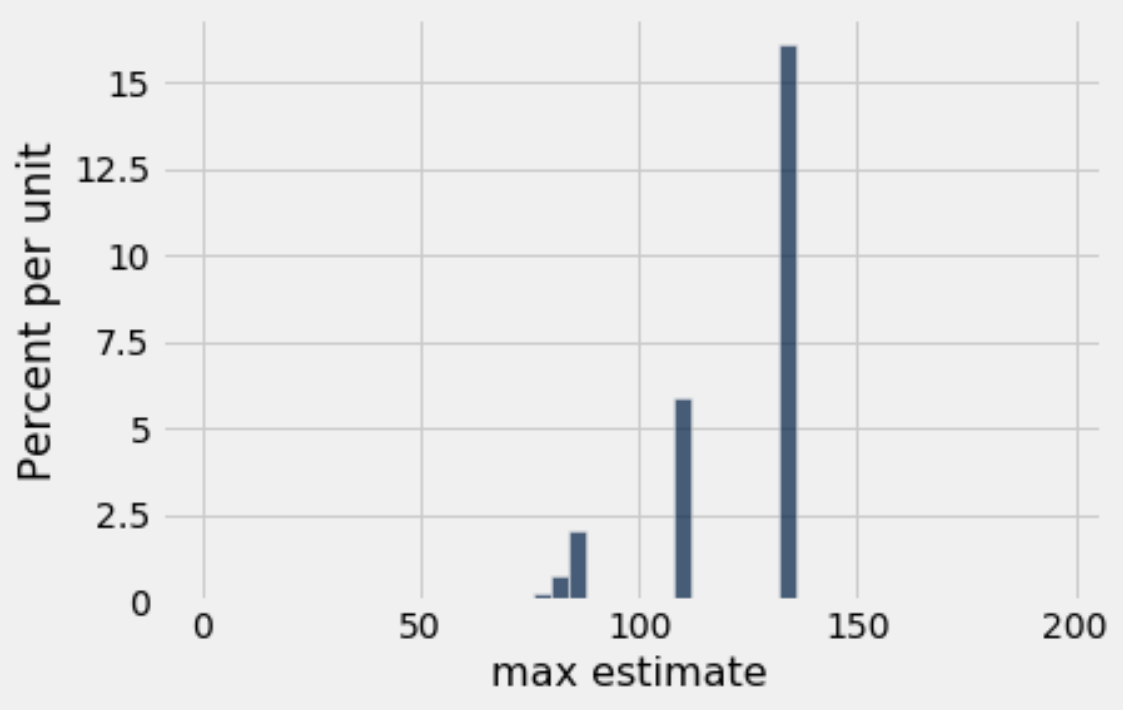

**Question 2.5** Using the bootstrap and the sample `observations`, simulate the approximate distribution of *max estimates* of `N`. Use 10,000 replications and save the estimates in an array called `bootstrap_max_estimates`.

We have provided code that plots a histogram, allowing you to visualize the simulated estimates.

`BEGIN QUESTION`
`name: q2_5`

[38]:
```
bootstrap_max_estimates = sample_estimates(observations, num_observations, max,␣
↪10000) #SOLUTION

# Don't change the code below! This plots bootstrap_max_estimates.
Table().with_column("max estimate", bootstrap_max_estimates).hist(bins=np.
↪arange(0, 200, 4))
```




**Question 2.6** `N` was actually 150! Compare the histograms of estimates you generated in 2.4 and 2.5 and answer the following questions:

1. How does the distribution of values for the mean-based estimates differ from the max estimates? Do both distributions contain the true max value?
2. Which estimator is more dependent on the original random sample? Why so?

```
BEGIN QUESTION
name: q2_6
manual: true
```

**SOLUTION:** The distribution of values for the mean-based estimates is bell-shaped and centered around 125. It has a wide range of possible values. The distribution of the max estimates is a lot sparser - it has very few bars because there is only a small number of possible maximum values in the original sample. Only the mean-based estimate distribute contains the true max value. The max-based estimator depends largely on the random sample we received while the mean-based estimator still provides a more accurate range even if the sample is not as representative of the population. This is because the max-based estimator is limited to the values in the sample while the mean-based estimator is not.

## 1.3 3. Computing intervals

**Question 3.1** Compute an interval that covers the middle 95% of the mean-based bootstrap estimates. Assign your values to `left_end_1` and `right_end_1`.

*Hint:* Use the `percentile` function! Read up on its documentation here.

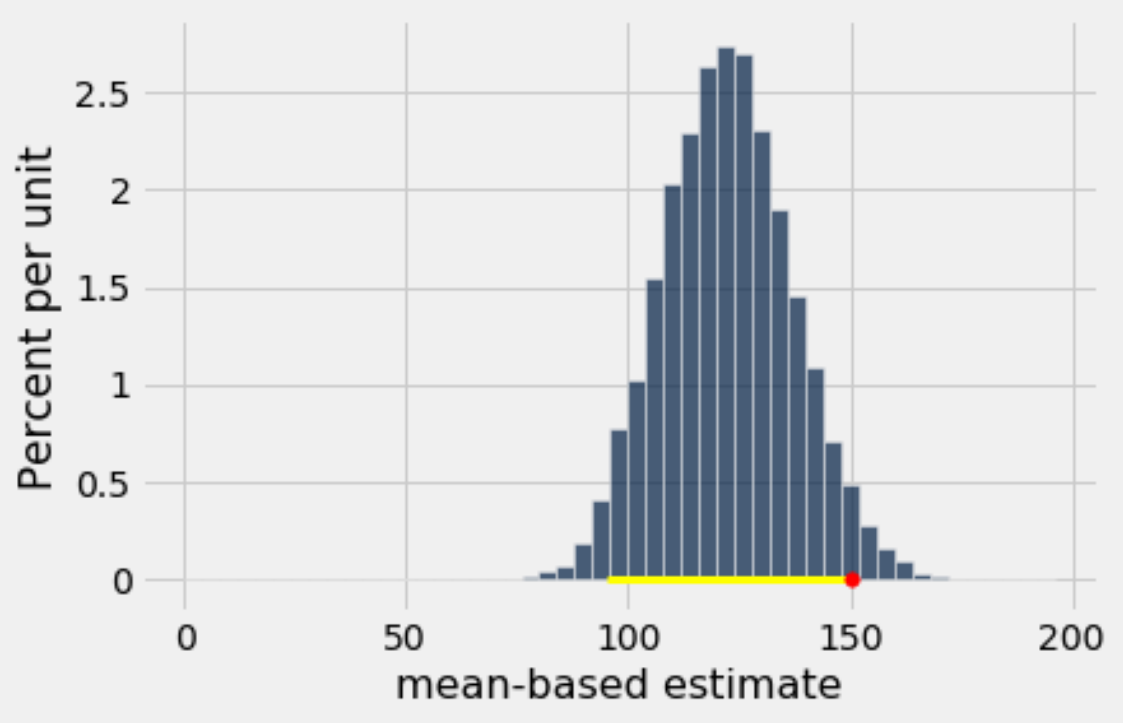
Verify that your interval looks like it covers 95% of the area in the histogram. The red dot on the histogram is the value of the parameter (150).

```
BEGIN QUESTION
name: q3_1
```

[41]:
```
left_end_1 = percentile(2.5, bootstrap_mean_based_estimates) #SOLUTION
right_end_1 = percentile(97.5, bootstrap_mean_based_estimates) #SOLUTION
print("Middle 95% of bootstrap estimates: [{:f}, {:f}]".format(left_end_1,␣
 ↪right_end_1))

# Don't change the code below! It draws your interval and N on the histogram of␣
 ↪mean-based estimates.
Table().with_column("mean-based estimate", bootstrap_mean_based_estimates).
 ↪hist(bins=np.arange(0, 200, 4))
plt.plot(make_array(left_end_1, right_end_1), make_array(0, 0), color='yellow',␣
 ↪lw=3, zorder=1)
plt.scatter(150, 0, color='red', s=30, zorder=2);
```

```
Middle 95% of bootstrap estimates: [95.058824, 151.529412]
```




**Question 3.2** 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 by sampling from the `population` table we have created for you below. Use the sample size `num_observations`.
2. Compute an estimate of `N` from these new observations, using `mean_based_estimator`.
3. Using only the new observations, compute 10,000 bootstrap estimates of `N`.
4. Plot these bootstrap estimates and compute an interval covering the middle 95%.

```
BEGIN QUESTION
name: q3_2
```

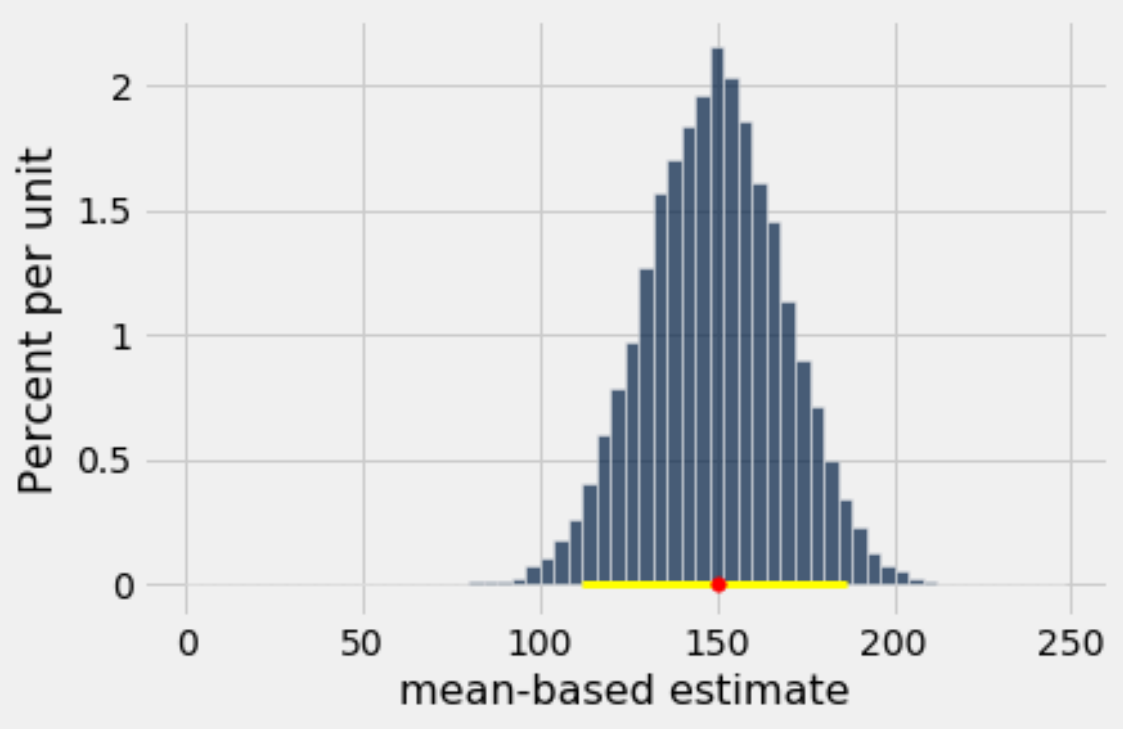
[45]:
```
population = Table().with_column("serial number", np.arange(1, 150+1))

new_observations = population.sample(num_observations) # SOLUTION
new_mean_based_estimate = mean_based_estimator(new_observations.column("serial␣
 ↪number")) # SOLUTION
new_bootstrap_estimates = sample_estimates(new_observations, num_observations,␣
 ↪mean_based_estimator, 10000) # SOLUTION
Table().with_column("mean-based estimate", new_bootstrap_estimates).
 ↪hist(bins=np.arange(0, 252, 4))
new_left_end = percentile(2.5, new_bootstrap_estimates) # SOLUTION
new_right_end = percentile(97.5, new_bootstrap_estimates) # SOLUTION

# Don't change code below this line!
print("New mean-based estimate: {:f}".format(new_mean_based_estimate))
print("Middle 95% of bootstrap estimates: [{:f}, {:f}]".format(new_left_end,␣
 ↪new_right_end))

plt.plot(make_array(new_left_end, new_right_end), make_array(0, 0),␣
 ↪color='yellow', lw=3, zorder=1)
plt.scatter(150, 0, color='red', s=30, zorder=2);
```

```
New mean-based estimate: 148.823529
Middle 95% of bootstrap estimates: [111.529412, 186.470588]
```




**Question 3.3** Does the interval covering the middle 95% of the new bootstrap estimates include `N`? If you ran that cell 100 times and generated 100 intervals, how many of those intervals would you expect to include `N`?

```
BEGIN QUESTION
name: q3_3
manual: true
```

**SOLUTION:** When we ran this, it did. We'd expect about 95 of the 100 intervals to include `N`. Note that this *process* generates an interval that captures the parameter 95% of the time. Each interval, however, is fixed and either includes the parameter or doesn't.

Congratulations, you're done with lab 8! Be sure to - **run all the tests** (the next cell has a shortcut for that), - **Save and Checkpoint** from the `File` menu, - **run the last cell to submit your work**, - and ask one of the staff members to check you off.

[31]:
```
# For your convenience, you can run this cell to run all the tests at once!
import os
print("Running all tests...")
_ = [ok.grade(q[:-3]) for q in os.listdir("tests") if q.startswith('q') and␣
 ↪len(q) <= 10]
print("Finished running all tests.")
```

```
Running all tests…
Finished running all tests.
```

[ ]:
```
_ = ok.submit()
```