lab10

May 3, 2022

Name: Allan Gongora

Section: 0131

1 Lab 10: Resampling and the Bootstrap

Welcome to Lab 10!

The British Royal Air Force wanted to know how many warplanes the Germans had (some number `N`, which is a *population 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 amongst 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*.

Please complete this notebook by filling in the cells provided. Before you begin, execute the following cell to load the provided tests.

[1]:
```
pip install gofer-grader
```

```
Collecting gofer-grader
  Using cached gofer_grader-1.1.0-py3-none-any.whl (9.9 kB)
Requirement already satisfied: tornado in /opt/conda/lib/python3.7/site-packages
(from gofer-grader) (6.1)
Requirement already satisfied: jinja2 in /opt/conda/lib/python3.7/site-packages
(from gofer-grader) (3.0.3)
Requirement already satisfied: pygments in /opt/conda/lib/python3.7/site-
packages (from gofer-grader) (2.11.2)
Requirement already satisfied: MarkupSafe>=2.0 in /opt/conda/lib/python3.7/site-
```

```
packages (from jinja2->gofer-grader) (2.0.1)
Installing collected packages: gofer-grader
Successfully installed gofer-grader-1.1.0
Note: you may need to restart the kernel to use updated packages.
```

[2]:
```
# 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', UserWarning)

# Don't change this cell; just run it.
from gofer.ok import check
```

Recommended Reading: * Randomness * Sampling and Empirical Distributions

1) For all problems that you must write explanations and sentences for, you **must** provide your answer in the designated space. This can include:
   A) Sentence reponses to questions that ask for an explanation
   B) Numeric responses to multiple choice questions
   C) Programming code
2) Moreover, throughout this lab 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. Otherwise, you will fail tests that you thought you were passing previously!

Once you're finished, select "Save and Checkpoint" in the File menu. Your name and course section number should be in the first and last cell of the assignment. Be sure you have run all cells with code and that the output from that is showing. Then click "Print Preview" in the File menu. Print a copy from there in pdf format. (This means you right click and choose print and choose "save as pdf" from your printer options.) You will need to submit the pdf in Canvas by the deadline.

The gopher grader output and/or output from your coding are essential to helping your instructor grade your work correctly and in a timely manner.

Files submitted that are missing the required output will lose some to all points so double check your pdf before submitting.

1.1 1. Preliminaries

The setup: We want to know the number of elements in the population. That number is `N`. Each element is numbered from 1 to `N`.

We only see a small number of elements (assumed to be a uniform random sample with replacement from among all the elements), 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 param because "number of elements in the population". Population parameter.

Check your answer by posting on the discussion forum.

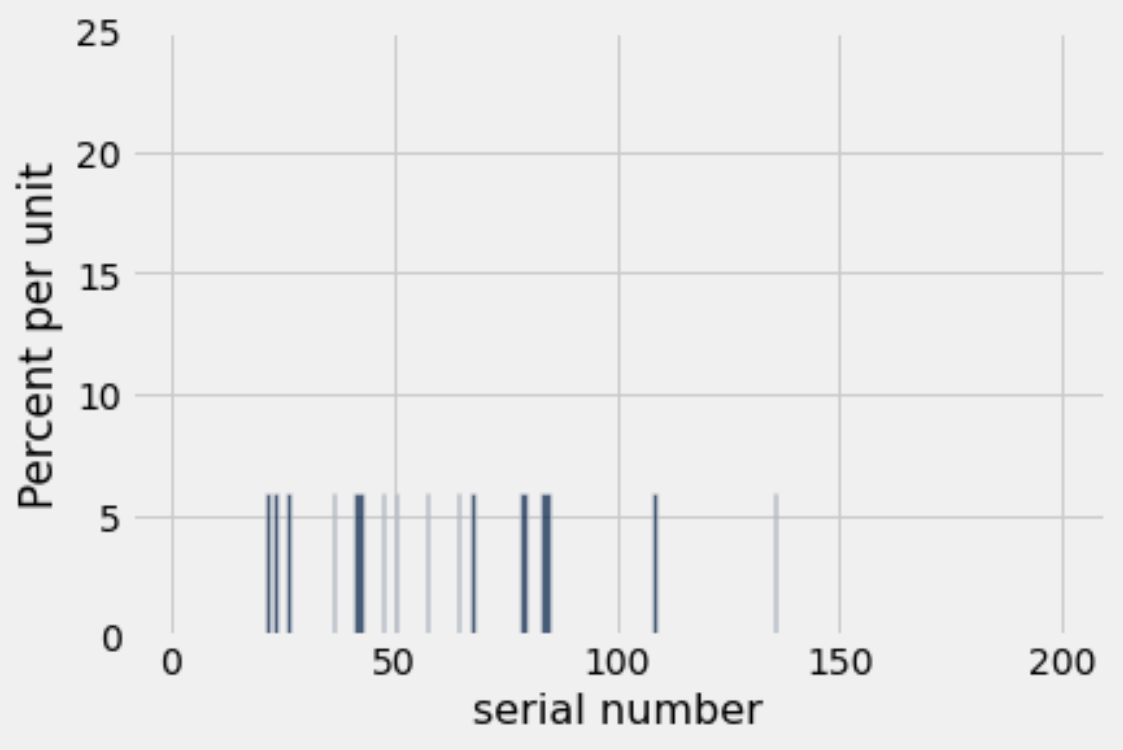
To make the situation realistic, we're going to hide the true number of elements from you. You'll have access only to this random sample:

[3]:
```
observations = Table.read_table("serial_numbers.csv")
num_observations = observations.num_rows
observations
```

[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 plot a histogram of the values in the column **using bins of width 1** ranging from **1 to 200** but return nothing. Then, call that function to make a histogram of `observations`.

[9]:
```
def plot_serial_numbers(numbers):
    numbers.hist("serial number", bins=range(1, 201))

    # 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 By looking at the histogram, what can we say about `N` immediately? (Hint: What is the relationship between `N` and the largest serial number in `observations`?) What does each little bar in the histogram represent? Why are all the bars the same height?

[10]:
```
observations[0].max()
```

[10]: 135

N is at least 135. Each bar is showing that that number appears in the data. The bars are the same height because they all appear the same number of times, once.

Question 1.4 One way to estimate `N` is to take twice the mean of the serial numbers we observe. 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`.

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

[14]:
```
def mean_based_estimator(nums):
    return nums.mean() * 2

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

[14]: 122.47058823529412

[15]:
```
check('tests/q1_4.py')
```

[15]:
```
<gofer.ok.OKTestsResult at 0x7fdd367a0a90>
```

Question 1.5 We can also estimate `N` using the biggest serial number in the sample. Compute it, giving it the name `max_estimate`.

[19]:
```
max_estimate = observations[0].max()
max_estimate
```

[19]:
```
135
```

[20]:
```
check('tests/q1_5.py')
```

[20]:
```
<gofer.ok.OKTestsResult at 0x7fdd369551d0>
```

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`. For these specific values, is it possible for our value of `mean_based_estimate` to 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 the value of our "`mean_based_estimate` is at least *[fill in a number]* away from `N`"?

Means_based_estimate can't be correct because it is smaller than max_estimate. We can't tell. If we round, mean_based_estimate is at least 12 away from n.

Check your answer by posting on the discussion forum.

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.

1.2 2. Resampling

To do this, we'll use resampling. That is, we won't exactly simulate new observations. Rather, we sample from our current sample, or "resample" the data.

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.

[22]:
```
def simulate_resample():
    return observations.sample()
```

Let's make one resample.

[23]:
```
# This line 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
```

[23]:
```
serial number
108
57
57
36
41
42
47
50
135
47
… (7 rows omitted)
```

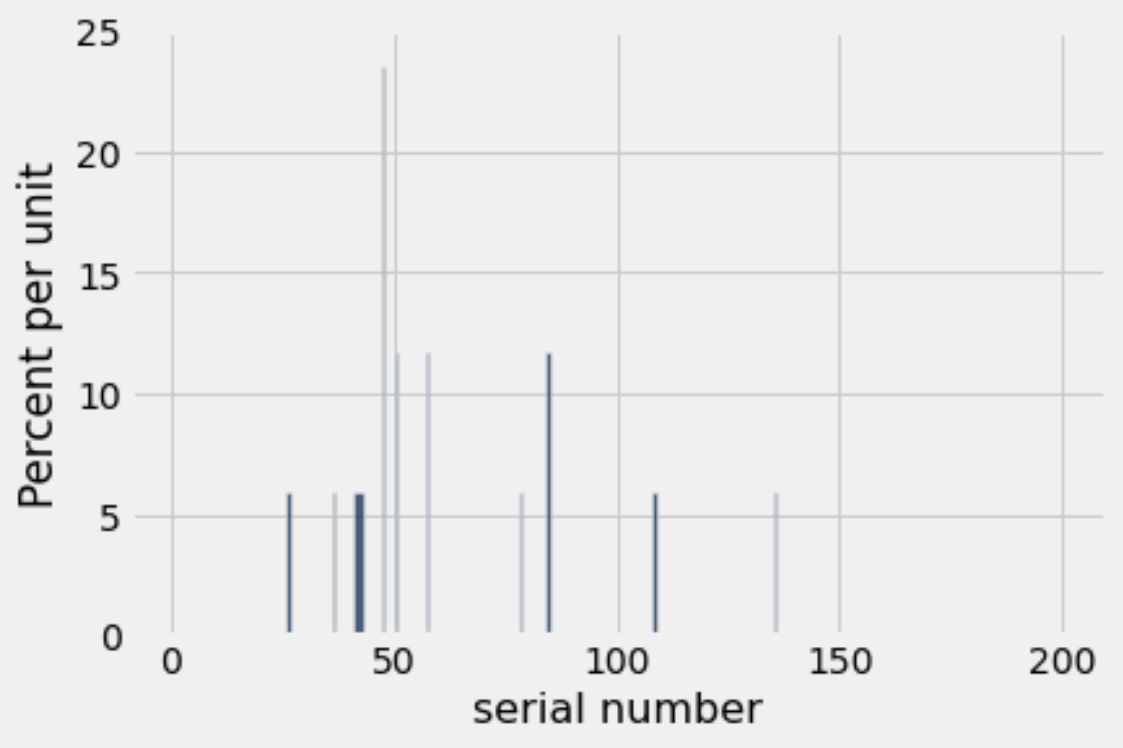
[24]:
```
check('tests/q2_1.py')
```

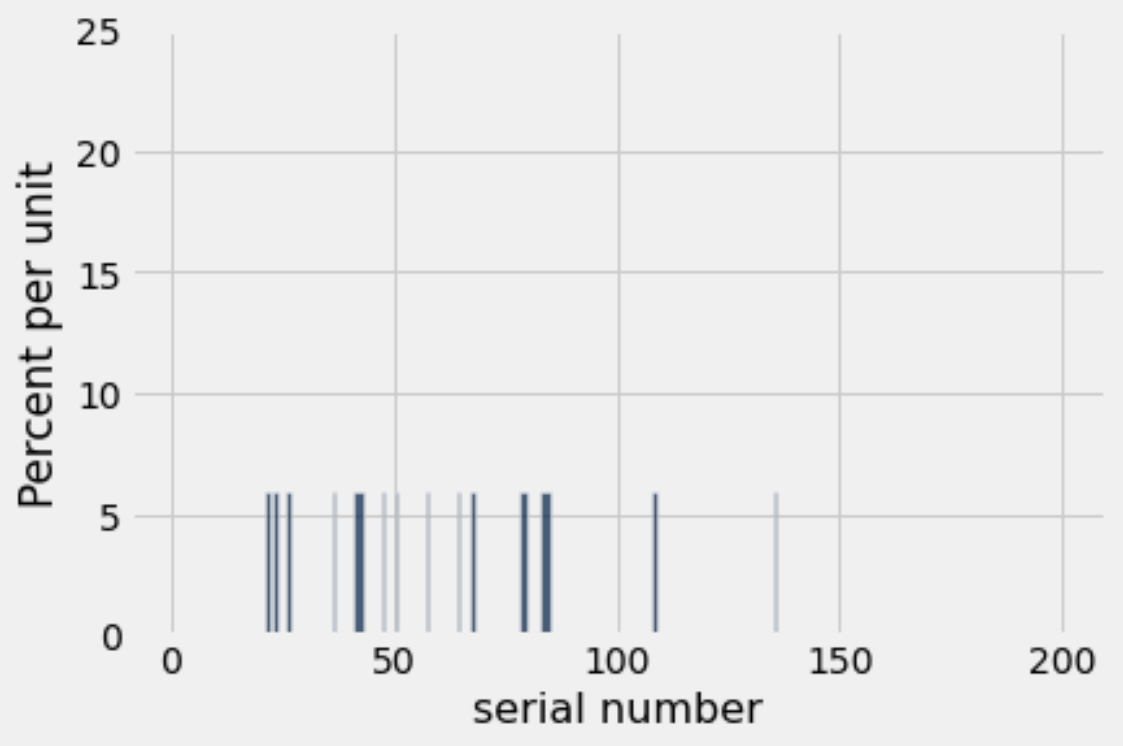
[24]:
```
<gofer.ok.OKTestsResult at 0x7fdd35a4f850>
```

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 In preparation for answering the next question, generate a histogram of your resample using the plotting function you defined earlier in this lab, **and** generate a separate histogram of the original observations.

[25]:
```
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.

Assign `true_statements` to a list of the correct statements.

[28]:
```
true_statements = [1, 4]
```

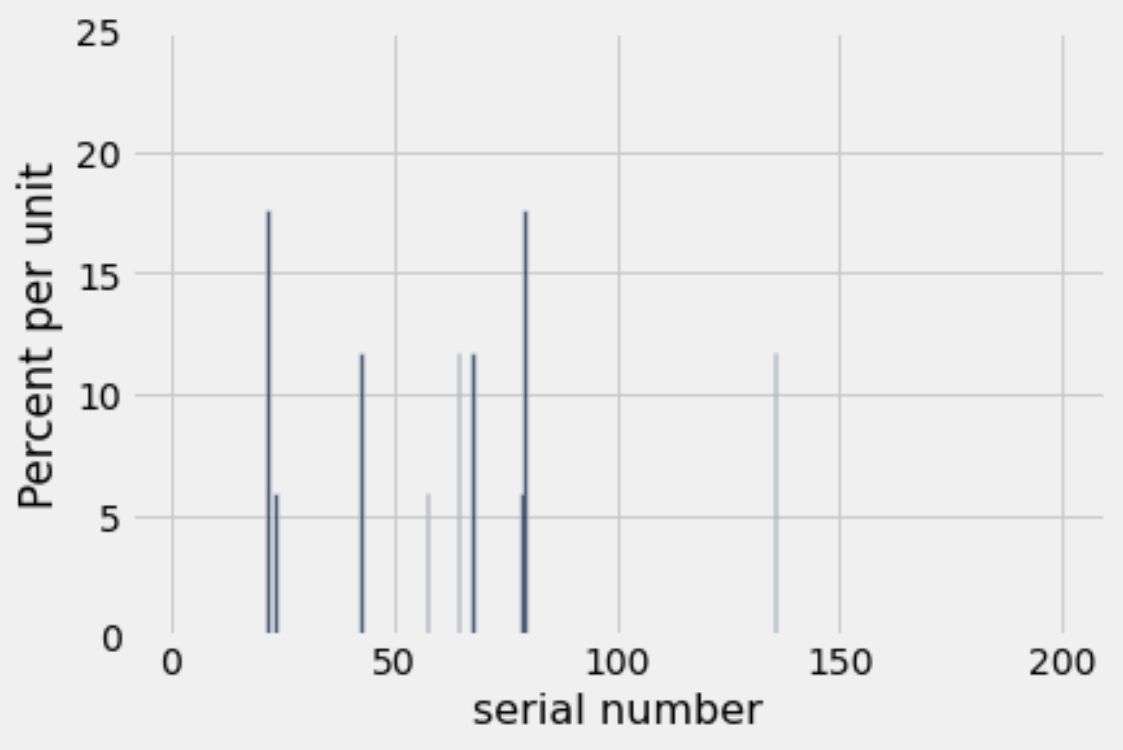
[29]:
```
check('tests/q2_3.py')
```

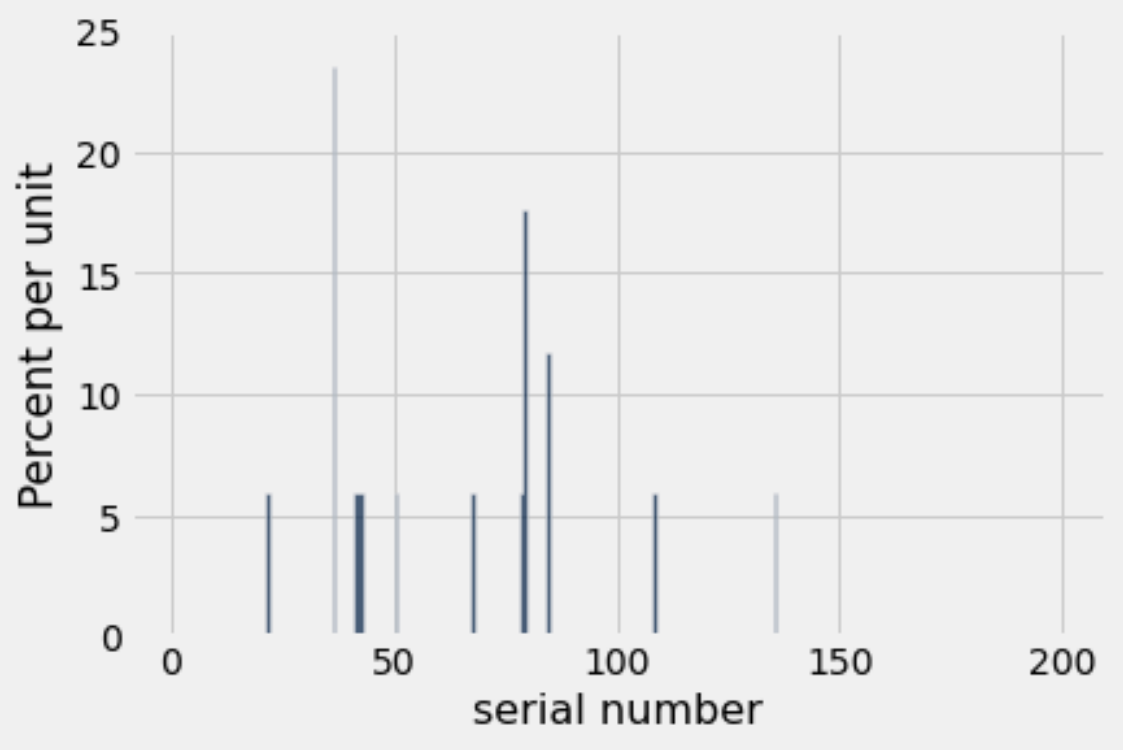
[29]:
```
<gofer.ok.OKTestsResult at 0x7fdd367a0510>
```

Question 2.4 Create two more resamples using the function `simulate_resample` from above. For each resampled data, plot it and compute its max- and mean-based estimates.

[31]:
```
resample_0 = simulate_resample()
plot_serial_numbers(resample_0)
mean_based_estimate_0 = resample_0[0].mean()
max_based_estimate_0 = resample_0[0].max()
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()
plot_serial_numbers(resample_1)
mean_based_estimate_1 = resample_1[0].mean()
max_based_estimate_1 = resample_1[0].max()
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: 63.1764705882353
Max-based estimate for resample 0: 135
Mean-based estimate for resample 1: 64.17647058823529
Max-based estimate for resample 1: 135
```


25
20
15
10
5
0
Percent per unit
0
50
100
150
200
serial number


25
20
15
10
5
0
Percent per unit
0
50
100
150
200
serial number

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 probability that 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 as compared to a max-based estimate.

As a refresher, here are some rules of probability that may be helpful:

- When all outcomes are equally likely:

$$\text{P(event happens)} = \frac{\text{number of outcomes that make event happen}}{\text{number of all outcomes}}$$

- When an event can happen in 2 ways: P(event) = P(event happening first way) + P(event happening second way)
- When 2 events must both happen: P(2 events both happen) = P(one event happens) $*$ P(other event happens, given the first one happened)
- When an event doesn't happen: P(event doesn't happen) = 1 − P(event does happen)
- P(at least one success) $= 1 - $ P(no successes)

1 - (16/17)**17 ~ 64.32%

Max based matching relies on randomly selecting 135 at least once in 17 trys. Mean based estimate matching would mean selecting each serial number exactly once which is less likely.

Discuss your answers on the edX discussion forums. If you have difficulty with the probability calculation, ask for help; don't stay stuck on it for too long.

1.3 3. Simulating With Resampling

Note: *The last part of this lab is difficult to check automatically, so it will not be graded. We strongly suggest that you try to complete it. We will release solutions to this lab so that you can compare to them.*

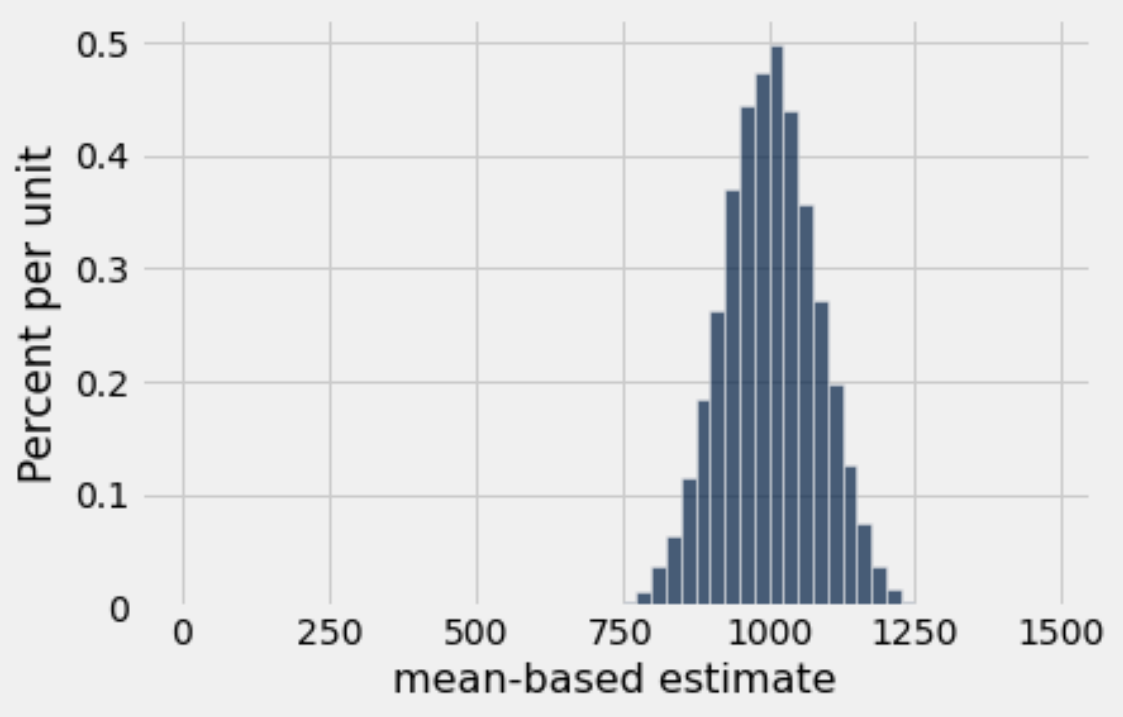
Since resampling from a sample looks just like sampling from a population, the code should look almost the same. That means we can write a function that simulates the process of either sampling from a population or resampling from a sample. If we pass in population as its argument, it will do the former;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

Question 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.

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.

[44]:
```
def simulate_estimates(original_table, sample_size, statistic,␣
 ↪num_replications):
    # Our implementation of this function took 5 short lines of code.
    stats = []

    for i in range(num_replications):
        stats.append(statistic(original_table.sample(sample_size)[0]))

    return np.array(stats)
# 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 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))
```



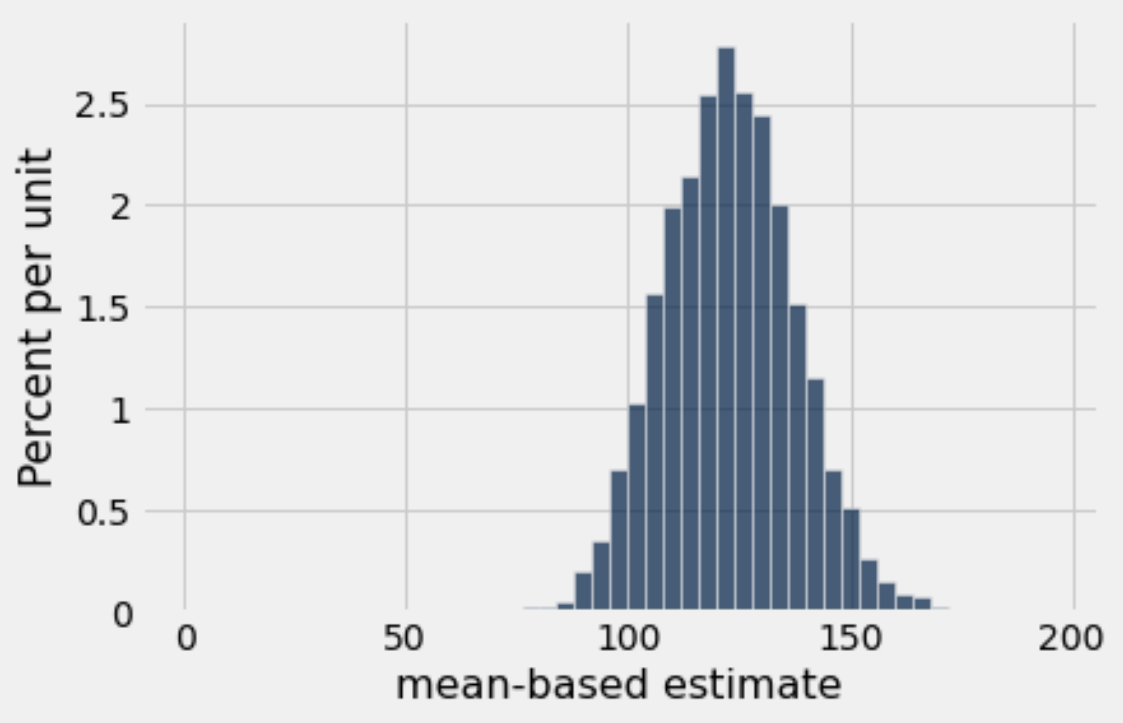
Now we can go back to the sample we actually observed (the table `observations`) and estimate

how much our mean-based estimate of N would have varied from sample to sample.

Question 3.2 Using the bootstrap and the sample `observations`, simulate the approximate distribution of *mean-based estimates* of N. Use 5,000 replications.

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

[42]:
```
bootstrap_estimates = simulate_estimates(observations, observations.num_rows,␣
↪mean_based_estimator, 5000)
Table().with_column("mean-based estimate", bootstrap_estimates).hist(bins=np.
↪arange(0, 200, 4))
```



Question 3.3 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.

[43]:
```
left_end = percentile(5, bootstrap_estimates)
right_end = percentile(95, bootstrap_estimates)
print("Middle 95% of bootstrap estimates: [{:f}, {:f}]".format(left_end,␣
↪right_end))
```

```
Middle 95% of bootstrap estimates: [99.176471, 147.411765]
```

Question 3.4 Your mean-based estimate of N should have been around 122. Given the above calculations, is it likely that N is exactly 122? If not, what is the typical range of values of the mean-based estimates of N for samples of size 17?

No. Probably 100-150.

Check your solutions with someone on the edX discussion forums.

Question 3.5 `N` was actually 150! Write code that simulates the sampling and bootstrapping process again, as follows:

1. Generate a new set of random observations by sampling from the population table we have created for you below.
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%.

[61]:
```
population = Table().with_column("serial number", np.arange(1, 150+1))

new_observations = population.sample()
new_bootstrap_estimates = simulate_estimates(new_observations, observations.
 ↪num_rows, mean_based_estimator, 5000)
new_mean_based_estimate = new_bootstrap_estimates.mean()

new_left_end = percentile(5, new_bootstrap_estimates)
new_right_end = percentile(95, new_bootstrap_estimates)

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))
```

```
New mean-based estimate: 160.850212
Middle 95% of bootstrap estimates: [125.764706, 194.823529]
```

Question 3.6 Does the interval covering the middle 95% of the new bootstrap estimates include `N`? If you ran that cell many times, what is the probability that it will include `N`?

Yes. Not actually sure that soujds like a hard question to answer.

Check your solutions with someone on the edX discussion forums.

1.4 4. Submission

Once you're finished, select "Save and Checkpoint" in the File menu. Your name and course section number should be in the first and last cell of the assignment. Be sure you have run all cells with code and that the output from that is showing.

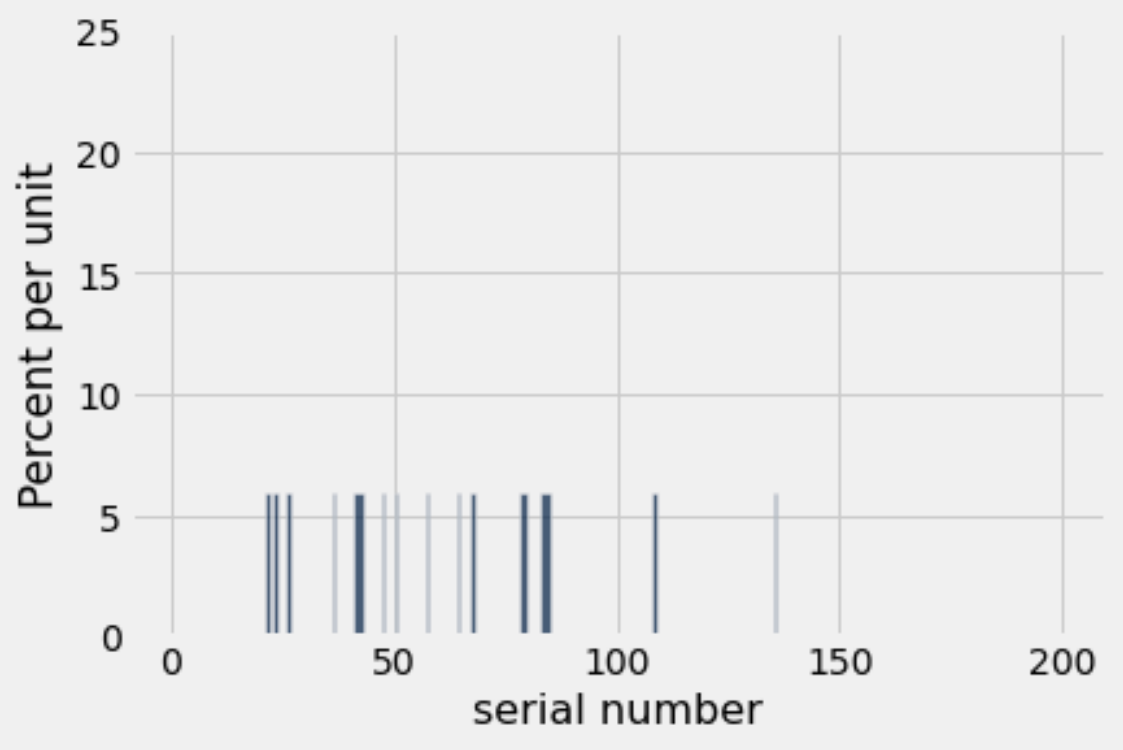
Double check that you have completed all of the free response questions as the auto-grader does NOT check that and YOU are responsible for knowing those questions are there and completing them as part of the grade for this lab. When ready, click "Print Preview" in the File menu. Print a copy from there in pdf format. (This means you right click and choose print and choose "save as pdf" from your printer options.) You will need to submit the pdf in Canvas by the deadline.

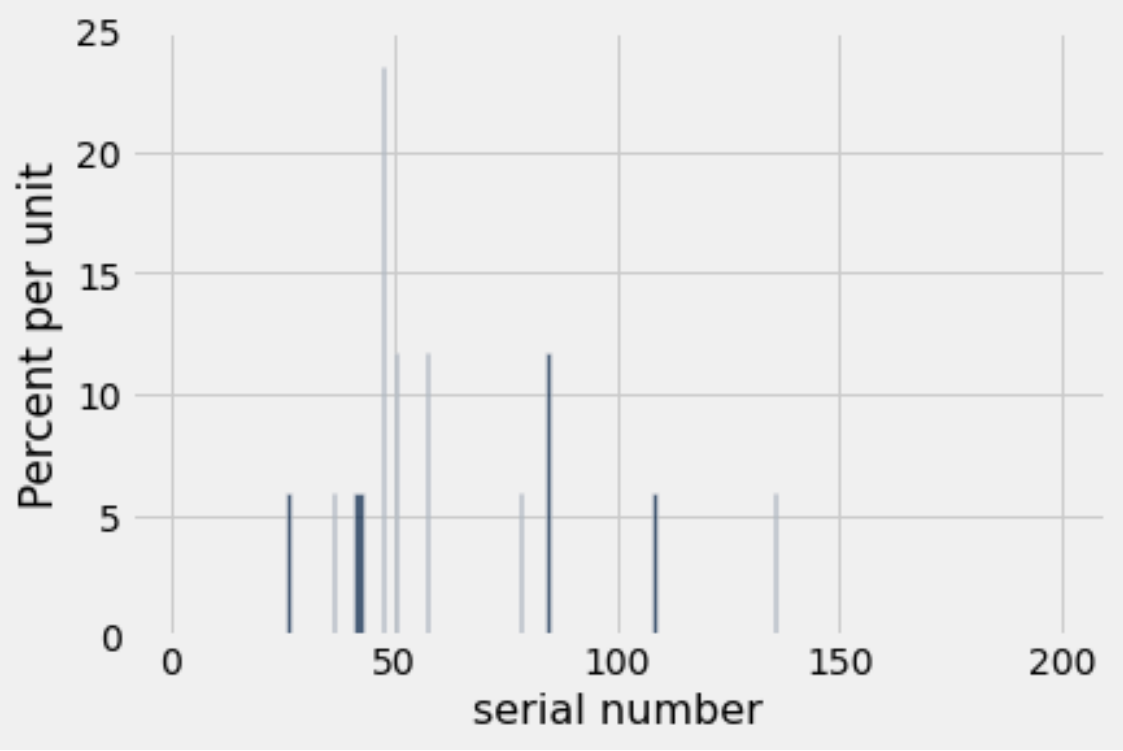
The gopher grader output and/or output from your coding are essential to helping your instructor grade your work correctly and in a timely manner.

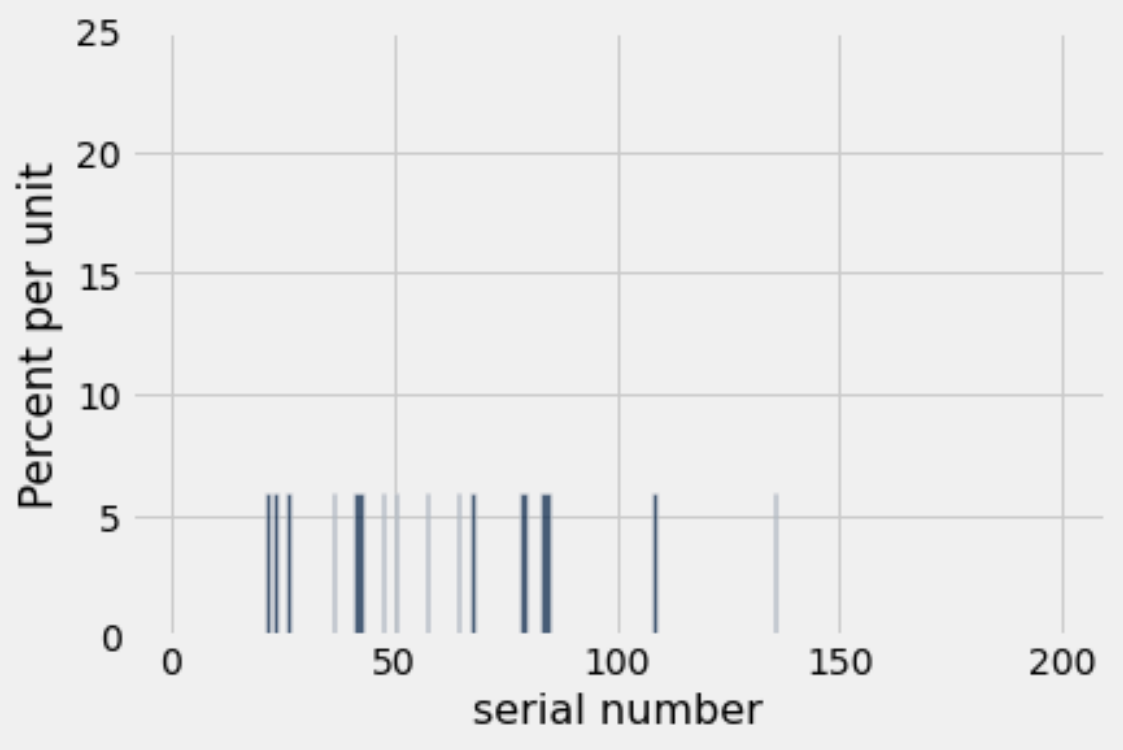
Files submitted that are missing the required output will lose some to all points so double check your pdf before submitting.

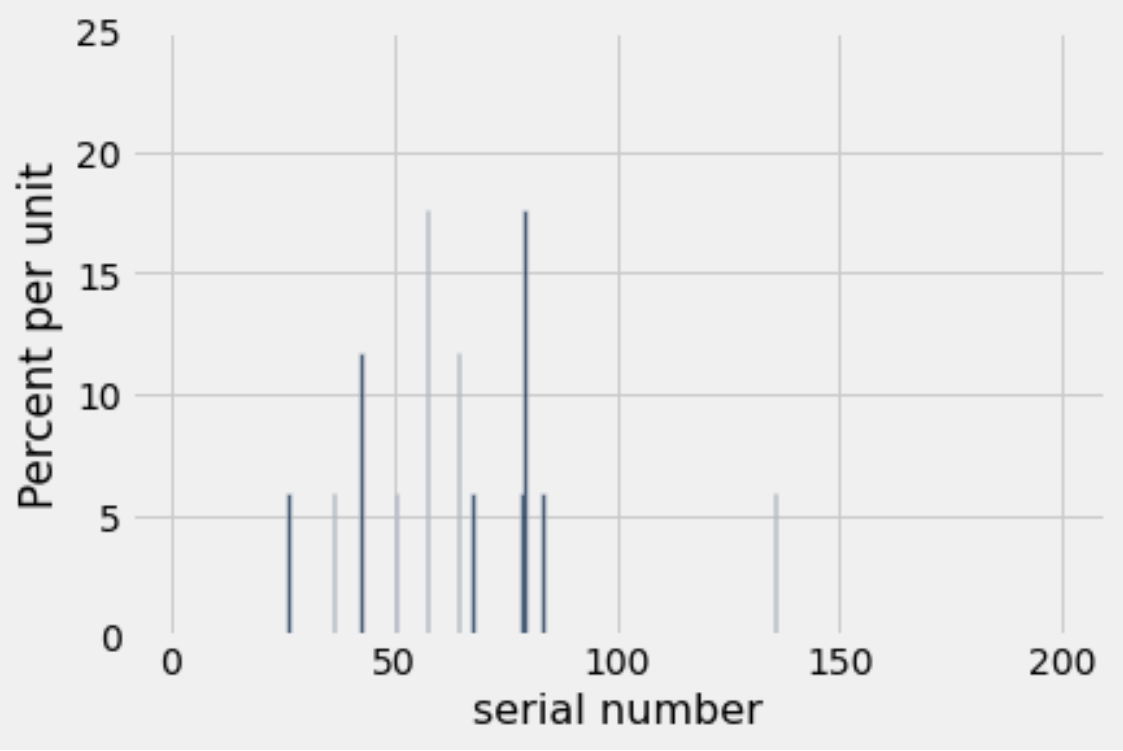
[66]:
```
# For your convenience, you can run this cell to run all the tests at once!
import glob
from gofer.ok import grade_notebook
if not globals().get('__GOFER_GRADER__', False):
    display(grade_notebook('lab10.ipynb', sorted(glob.glob('tests/q*.py'))))
```

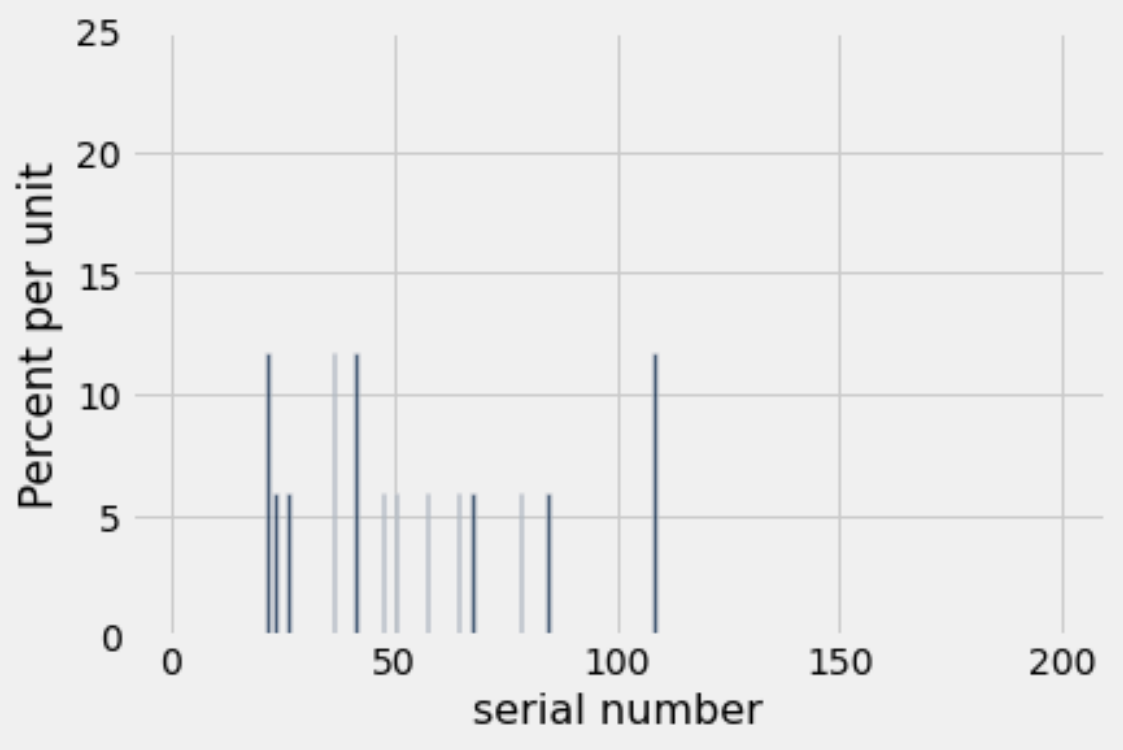
```
Mean-based estimate for resample 0: 64.41176470588235
Max-based estimate for resample 0: 135
Mean-based estimate for resample 1: 53.411764705882355
Max-based estimate for resample 1: 108
Middle 95% of bootstrap estimates: [99.294118, 147.058824]
New mean-based estimate: 136.524212
Middle 95% of bootstrap estimates: [104.000000, 171.294118]
['tests/q1_4.py', 'tests/q1_5.py', 'tests/q2_1.py', 'tests/q2_3.py']
Question 1:

<gofer.ok.OKTestsResult at 0x7fdd34b26050>

Question 2:

<gofer.ok.OKTestsResult at 0x7fdd34ade450>

Question 3:

<gofer.ok.OKTestsResult at 0x7fdd34e03850>

Question 4:

<gofer.ok.OKTestsResult at 0x7fdd34a32ad0>

1.0
```

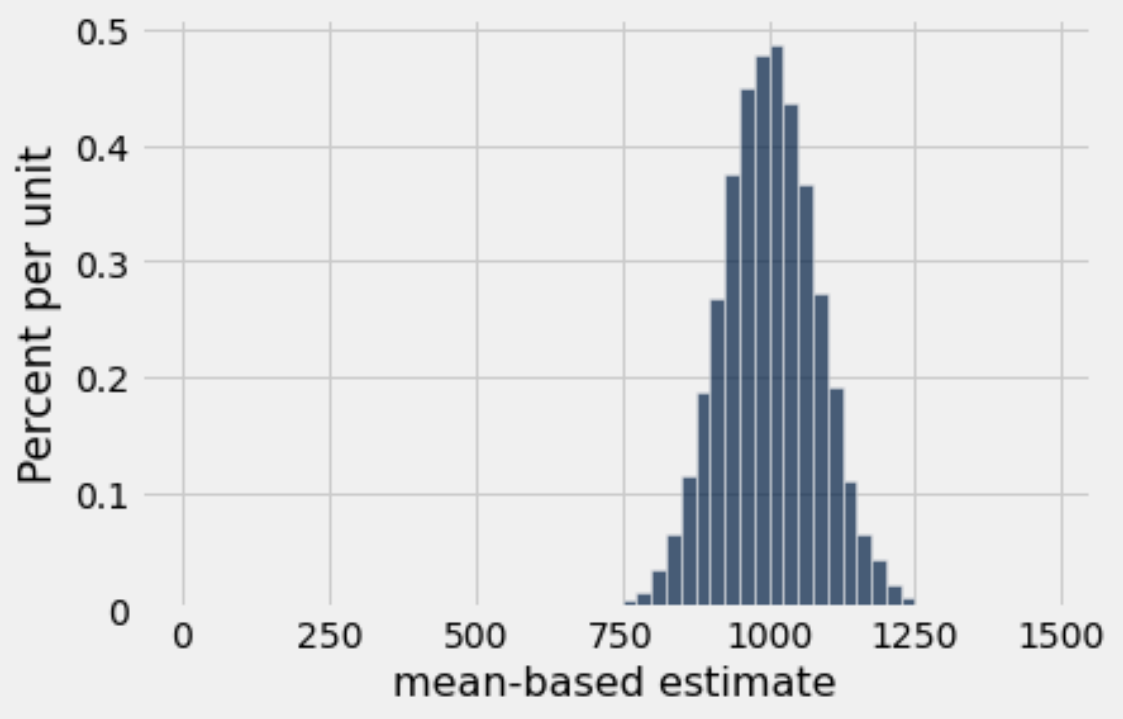

25
20
15
10
5
0
Percent per unit
0
50
100
150
200
serial number

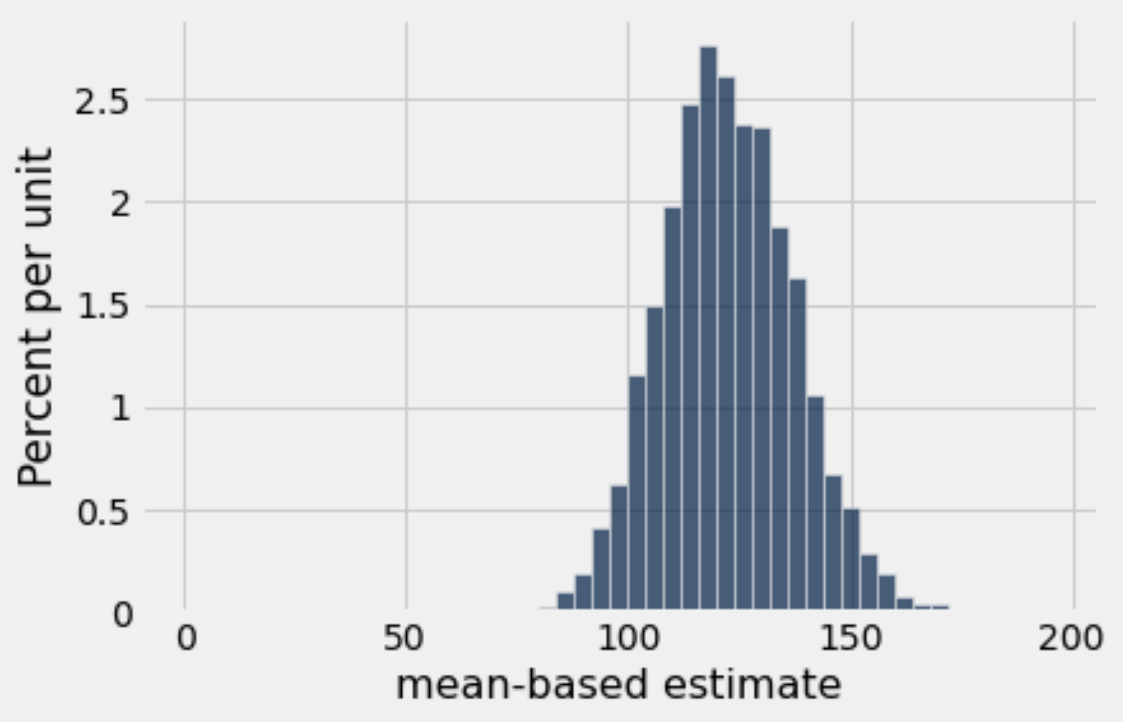

25
20
15
10
5
0
Percent per unit
0
50
100
150
200
serial number

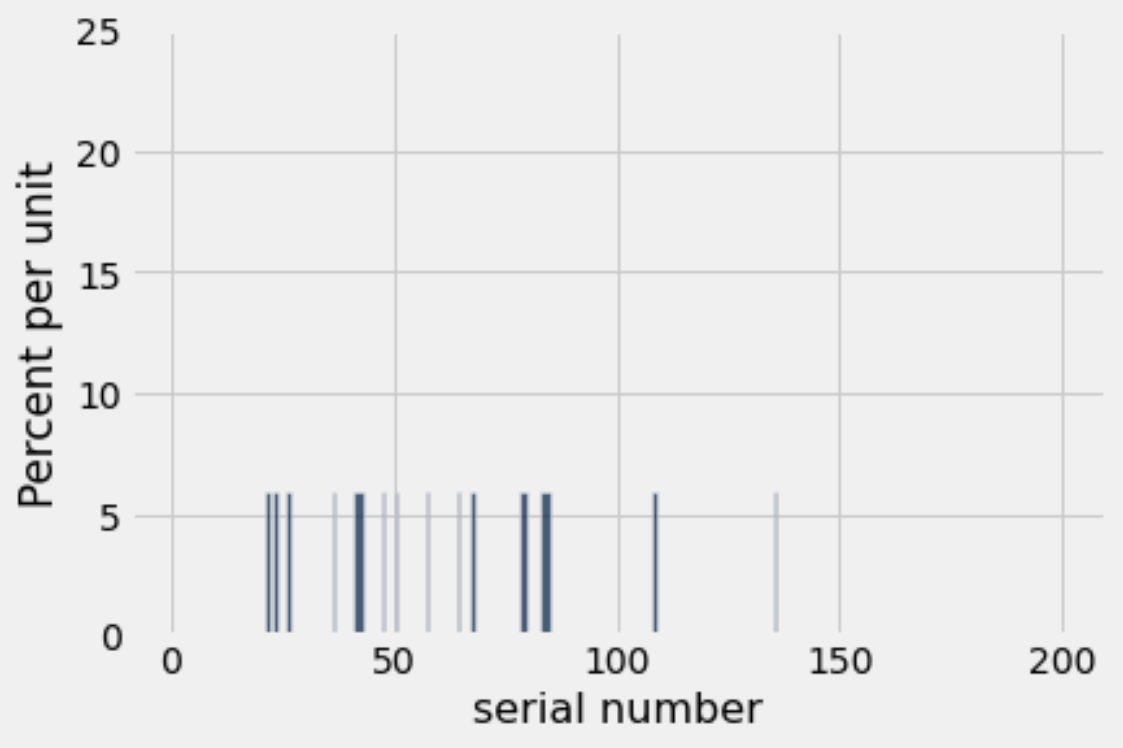

25
20
15
10
5
0
Percent per unit
0
50
100
150
200
serial number

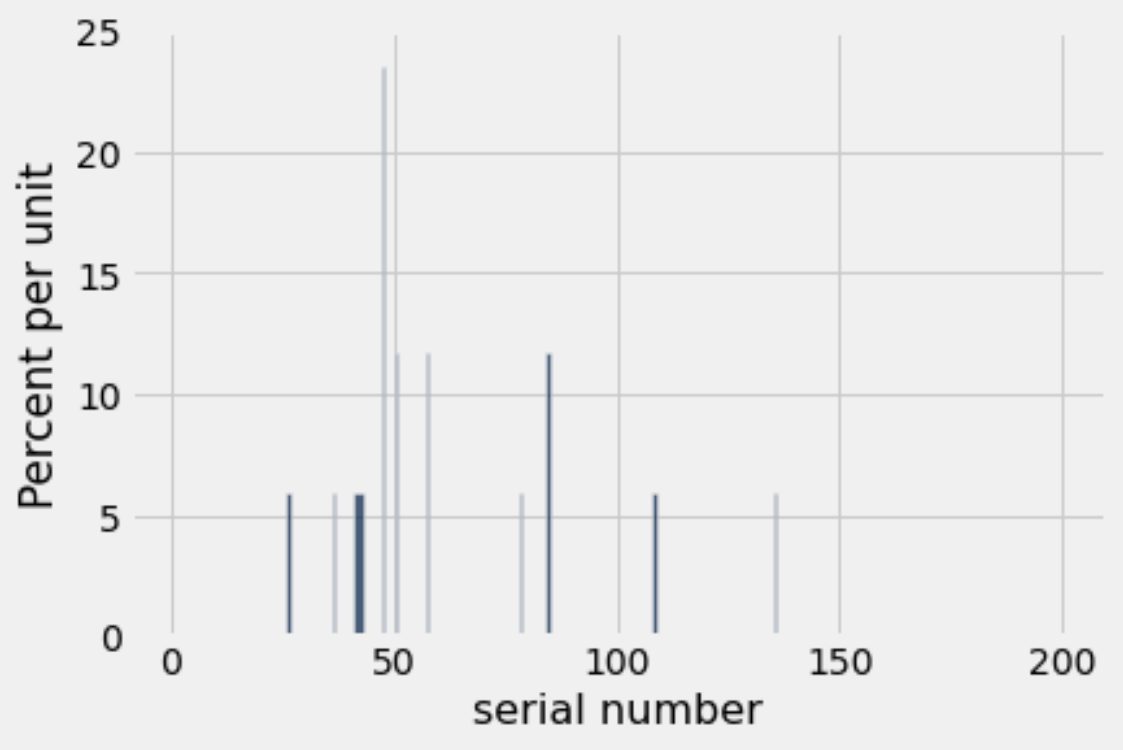

25
20
15
10
5
0
Percent per unit
0
50
100
150
200
serial number

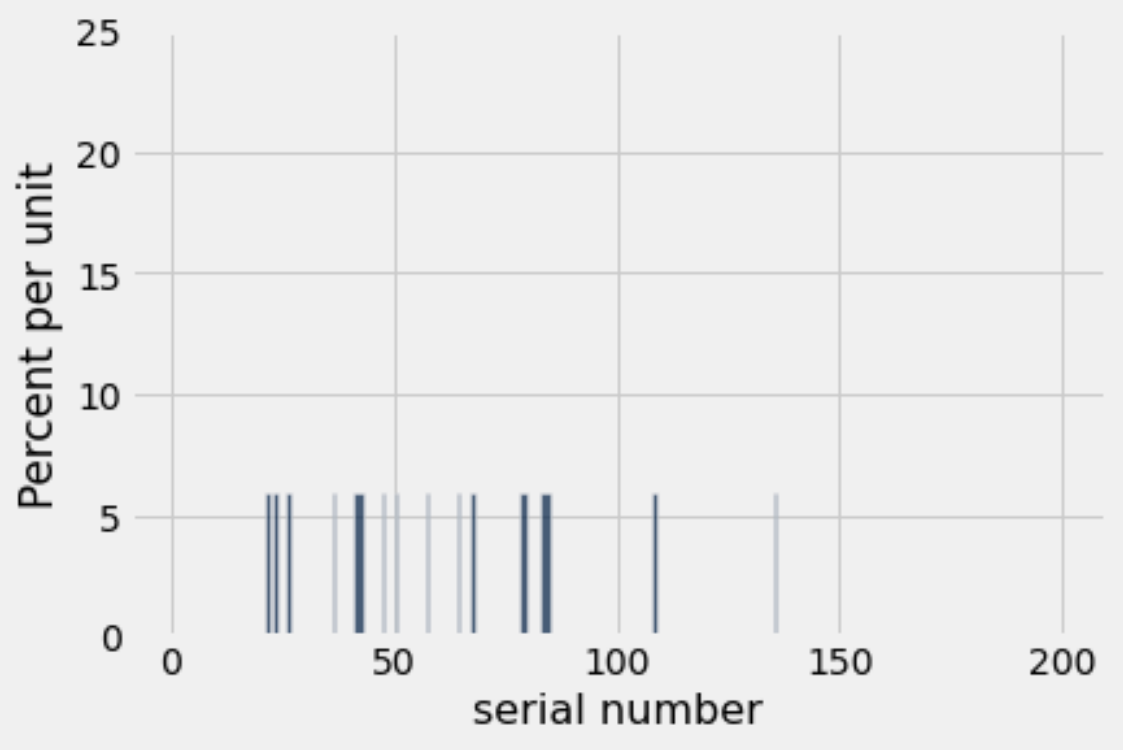

25
20
15
10
5
0
Percent per unit
0
50
100
150
200
serial number

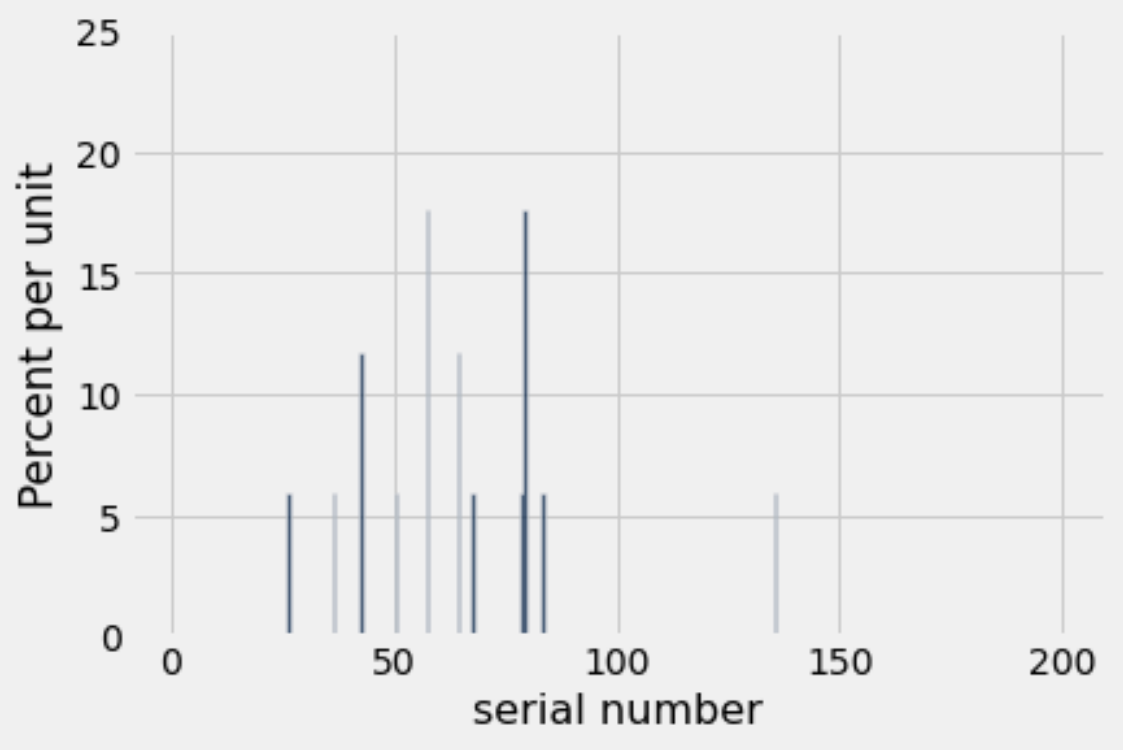

0.5
0.4
0.3
0.2
0.1
0
Percent per unit
0
250
500
750
1000
1250
1500
mean-based estimate

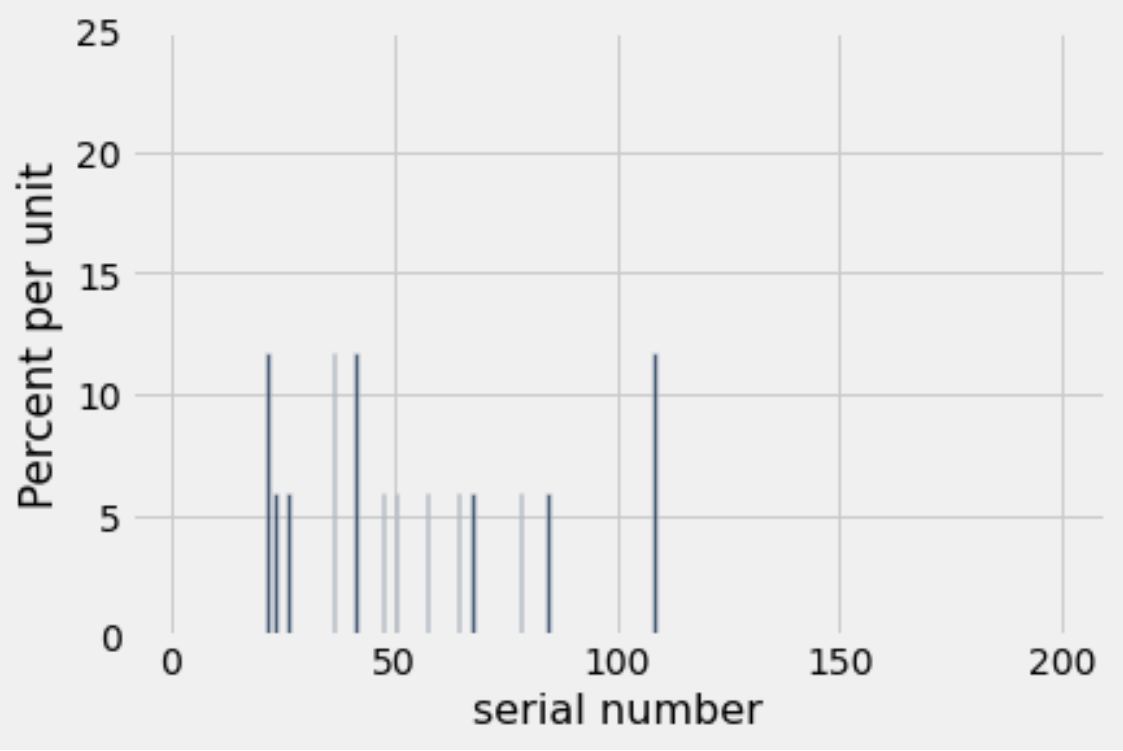

Percent per unit
2.5
2
1.5
1
0.5
0
0
50
100
150
200
mean-based estimate

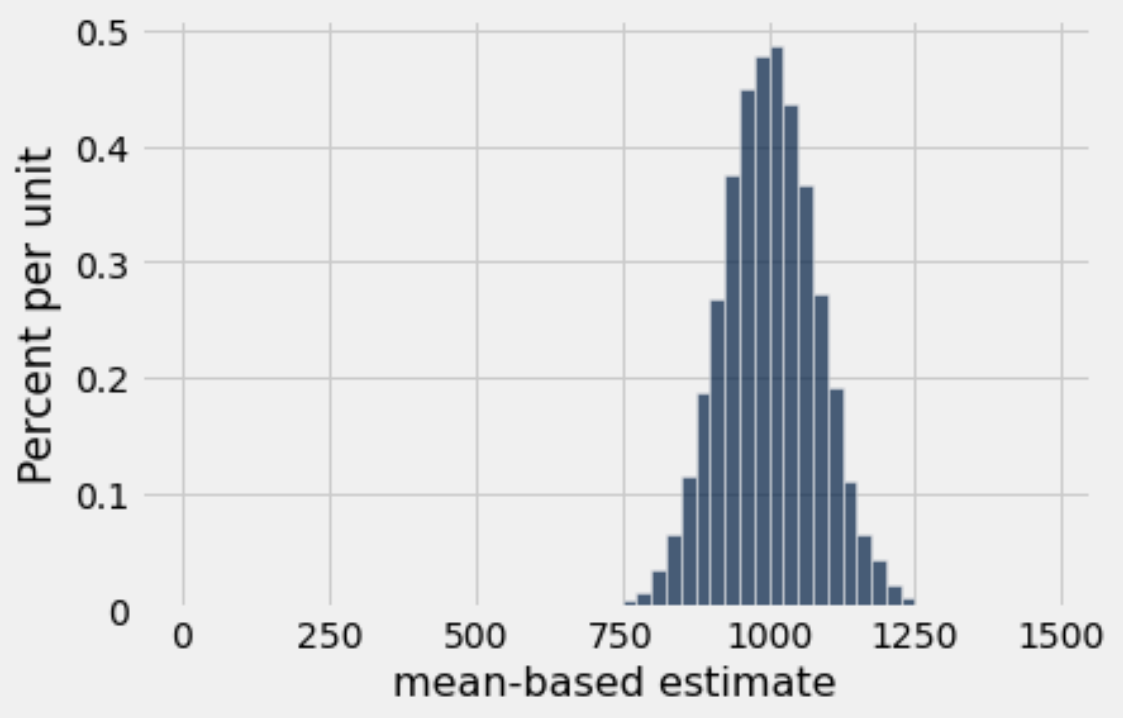

Percent per unit
25
20
15
10
5
0
0
50
100
150
200
serial number

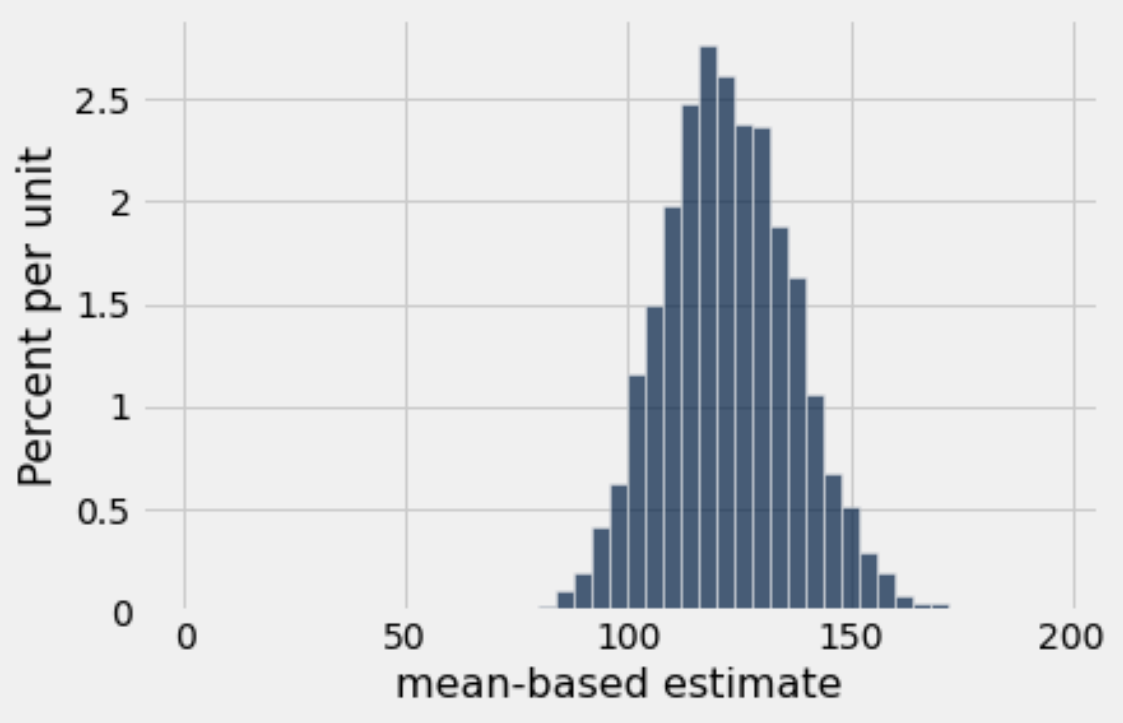

25
20
15
10
5
0
Percent per unit
0
50
100
150
200
serial number


25
20
15
10
5
0
Percent per unit
0
50
100
150
200
serial number


25
20
15
10
5
0
Percent per unit
0
50
100
150
200
serial number


25
20
15
10
5
0
Percent per unit
0
50
100
150
200
serial number





Name: Allan Gongora

Section: 0131