hw10_master

November 22, 2019

1 Homework 10: Central Limit Theorem

Reading: * Why the mean matters

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 10 is due **Thursday, 11/14 at 11:59pm**. You will receive an early submission bonus point if you turn in your final submission by Wednesday, 11/13 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.

[ ]:
```
# Don't change this cell; just run it.

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)

from client.api.notebook import Notebook
ok = Notebook('hw10.ok')
```

1.1 1. The Bootstrap and The Normal Curve

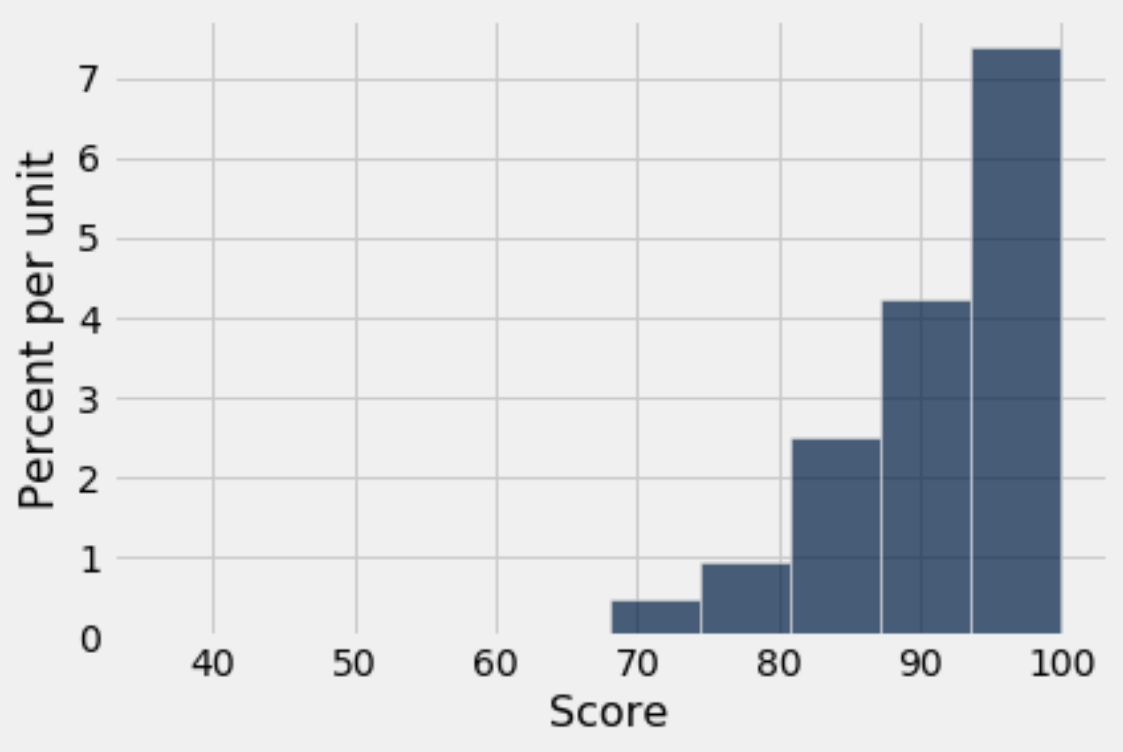
In this exercise, we will explore a dataset that includes the safety inspection scores for restaurants in the city of Austin, Texas. We will be interested in determining the average restaurant score for the city from a random sample of the scores; the average restaurant score is out of 100. We'll compare two methods for computing a confidence interval for that quantity: the bootstrap resampling method, and an approximation based on the Central Limit Theorem.

[2]:
```
# Just run this cell.
pop_restaurants = Table.read_table('restaurant_inspection_scores.csv').
 ↪drop('Facility ID','Process Description')
pop_restaurants
```

[2]:
```
Restaurant Name | Zip Code | Inspection Date | Score | Address
6M Grocery      | 78652    | 01/17/2014      | 90    | 805 W FM 1626 RD
AUSTIN, TX 78652
6M Grocery      | 78652    | 04/27/2015      | 93    | 805 W FM 1626 RD
AUSTIN, TX 78652
6M Grocery      | 78652    | 05/02/2016      | 88    | 805 W FM 1626 RD
AUSTIN, TX 78652
6M Grocery      | 78652    | 07/25/2014      | 100   | 805 W FM 1626 RD
AUSTIN, TX 78652
6M Grocery      | 78652    | 10/21/2015      | 87    | 805 W FM 1626 RD
AUSTIN, TX 78652
6M Grocery      | 78652    | 12/15/2014      | 93    | 805 W FM 1626 RD
AUSTIN, TX 78652
7 Eleven #36575 | 78660    | 01/25/2016      | 92    | 15829 N IH 35 SVRD NB
AUSTIN, TX 78660
7 Eleven #36575 | 78660    | 03/05/2015      | 86    | 15829 N IH 35 SVRD NB
AUSTIN, TX 78660
7 Eleven #36575 | 78660    | 03/14/2014      | 93    | 15829 N IH 35 SVRD NB
AUSTIN, TX 78660
7 Eleven #36575 | 78660    | 07/27/2015      | 97    | 15829 N IH 35 SVRD NB
AUSTIN, TX 78660
… (24357 rows omitted)
```

Run the cell below to plot a histogram of the scores from `pop_restaurants`.

[3]:
```
pop_restaurants.hist('Score')
```



This is the **population mean**:

[4]:
```
pop_mean = np.mean(pop_restaurants.column('Score'))
pop_mean
```

[4]:
```
91.40706693478886
```

Often it is impossible to find complete datasets like this. Imagine we instead had access only to a random sample of 100 restaurant inspections, called `restaurant_sample`. That table is created below. We are interested in using this sample to estimate the population mean.

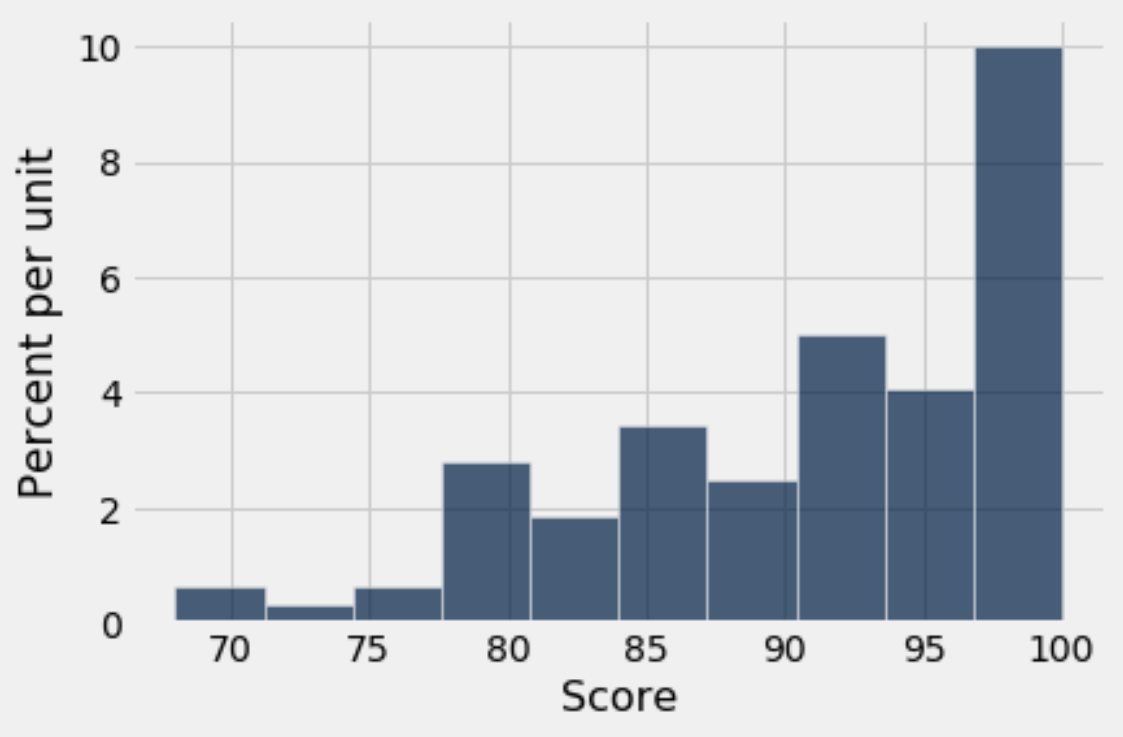
[5]:
```
restaurant_sample = pop_restaurants.sample(100, with_replacement=False)
restaurant_sample
```

[5]:
```
Restaurant Name                    | Zip Code | Inspection Date | Score |
Address
Snap Kitchen                       | 78759    | 12/30/2014      | 100   | 10001
RESEARCH BLVD NB Bunit 190
AUSTIN, TX 78759
(30. …
BJ's Restaurant & Brewhouse        | 78758    | 12/16/2014      | 94    | 10515
N MOPAC EXPY NB
AUSTIN, TX 78758
(30.39131, -97. …
```

```
McNeil High School                  | 78729    | 01/29/2015      | 100   | 5720
MC NEIL DR
AUSTIN, TX 78729
(30.447032, -97.732356)
Las Cimas Deli (Whole Foods Market) | 78746    | 07/20/2016      | 100   | 807
LAS CIMAS PKWY Bldg 2
AUSTIN, TX 78746
(30.290786, …
Hot Mama's Cafe                     | 78702    | 03/21/2014      | 97    | 2401
E 6TH ST Unit 1004
AUSTIN, TX 78702
(30.259413, - …
Workhorse                           | 78751    | 09/08/2016      | 89    | 100 E
NORTH LOOP BLVD Bldg B
AUSTIN, TX 78751
(30.3176 …
Baked By Amy's                      | 78729    | 01/06/2015      | 97    | 13265
N US 183 HWY NB Unit B
AUSTIN, TX 78729
(30.4623 …
Russian Bistro NaZdorovye           | 78701    | 12/08/2014      | 93    | 307 E
5TH ST
AUSTIN, TX 78701
(30.266283, -97.740149)
ABIA Schlotzsky's                   | 78719    | 11/18/2015      | 100   | 3600
PRESIDENTIAL BLVD
AUSTIN, TX 78719
(30.202654, -9 …
LW - Flintrock Falls Country Club   | 78738    | 01/02/2014      | 93    | 401
JACK NICKLAUS DR
LAKEWAY, TX 78738
(30.338765, -97 …
… (90 rows omitted)
```

Run the cell below to plot a histogram of the **sample** scores from `restaurant_sample`.

[6]:
```
restaurant_sample.hist('Score')
```



This is the **sample mean**:

[7]:
```
sample_mean = np.mean(restaurant_sample.column('Score'))
sample_mean
```

[7]:
```
90.92
```

Question 1 Complete the function `one_resampled_mean` below. It should take in an original table `data`, with a column `Score`, and return the mean score of one resample from `data`.

Remember to **call** your function and check the output before moving on to autograder tests.

```
BEGIN QUESTION
name: q1_1
manual: false
```

[17]:
```
def one_resampled_mean(data):
    resampled_data = data.sample() # SOLUTION
    return np.mean(resampled_data.column('Score')) # SOLUTION

# Check that your function works by calling it:
this_mean = one_resampled_mean(restaurant_sample) # SOLUTION
this_mean
```

[17]: 91.44

Question 2 Complete the function `bootstrap_scores` below. It should take no arguments. It should simulate drawing 5000 resamples from `restaurant_sample` and compute the mean restaurant score in each resample. It should return an array of those 5000 resample means.

```
BEGIN QUESTION
name: q1_2
manual: false
```

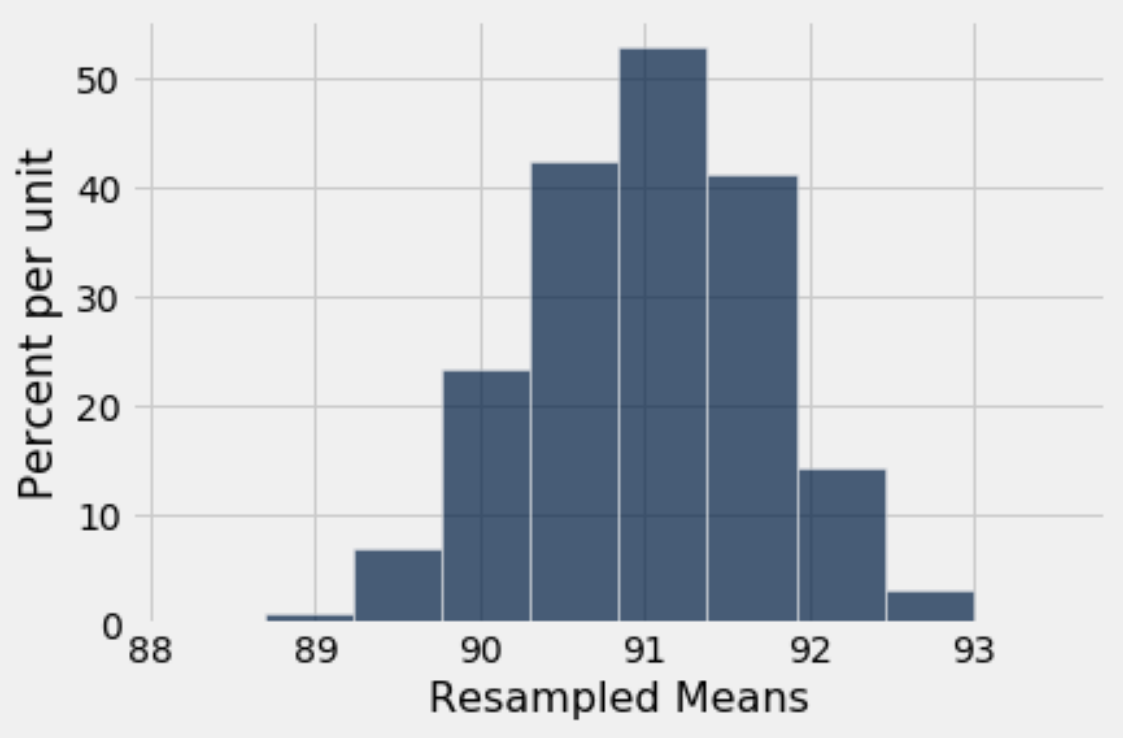
[11]:
```
def bootstrap_scores():
    resampled_means = make_array() #SOLUTION
    for i in range(5000):
        resampled_mean = one_resampled_mean(restaurant_sample) #SOLUTION
        resampled_means = np.append(resampled_means, resampled_mean) #SOLUTION
    return resampled_means #SOLUTION

resampled_means = bootstrap_scores()
resampled_means
```

[11]: array([89.56, 90.98, 91.07, …, 89.45, 90.69, 90.59])

Take a look at the histogram of the **resampled means**.

[15]:
```
Table().with_column('Resampled Means', resampled_means).hist()
```



Question 3 Compute a 95 percent confidence interval for the average restaurant score using the array `resampled_means`.

```
BEGIN QUESTION
name: q1_3
manual: false
```

[16]:
```
lower_bound = percentile(2.5,resampled_means) #SOLUTION
upper_bound = percentile(97.5, resampled_means) #SOLUTION
print("95% confidence interval for the average restaurant score, computed by␣
 ↪bootstrapping:\n(",lower_bound, ",", upper_bound, ")")
```

```
95% confidence interval for the average restaurant score, computed by
bootstrapping:
( 89.58 , 92.37 )
```

Question 4 What distribution is the histogram between question 2 and 3 displaying (that is, what data are plotted), and why does it have that shape?

```
BEGIN QUESTION
name: q1_4
manual: true
```

SOLUTION: The histogram is a distribution of the means of many samples from the restaurant scores, and the Central Limit Theorem tells us that a distribution of sample means of large random samples should be bell shaped.

Question 5 Does the distribution of the **sampled scores** look normally distributed? State "yes" or "no" and describe in one sentence why you should expect this result.

Hint: Remember that we are no longer talking about the resampled means!

```
BEGIN QUESTION
name: q1_5
manual: true
```

SOLUTION: No. The sampled scores are distributed much like the population scores, which are not normally distributed.

For the last question (Question 6), you'll need to recall two facts. 1. If a group of numbers has a normal distribution, around 95% of them lie within 2 standard deviations of their mean. 2. The Central Limit Theorem tells us the quantitative relationship between the following:
* the standard deviation of an array of numbers. * the standard deviation of an array of means of samples taken from those numbers.

Also recall the standard deviation of sample means:
$sd of sample means from many samples from a distribution =

$$\frac{\text{sd of the original distribution}}{\sqrt{\text{sample size}}}$$

$

Question 6 Calculate an interval around the `sample_mean` that covers approximately 95% of the numbers in the `resampled_means` array using the CLT. **You may use the following values to compute your result, but you should not perform additional resampling or reference the array `resampled_means`**

```
BEGIN QUESTION
name: q1_6
manual: false
```

[30]:
```
sample_mean = np.mean(restaurant_sample.column('Score'))
sample_sd = np.std(restaurant_sample.column('Score'))
sample_size = restaurant_sample.num_rows

sd_of_means = sample_sd / sample_size**0.5 # SOLUTION
lower_bound_normal = sample_mean - 2*sd_of_means # SOLUTION
upper_bound_normal = sample_mean + 2*sd_of_means # SOLUTION
print("95% confidence interval for the average restaurant score, computed by a␣
 ↪normal approximation:\n(",lower_bound_normal, ",", upper_bound_normal, ")")
```

```
95% confidence interval for the average restaurant score, computed by a normal
approximation:
( 92.62062580839152 , 94.95937419160849 )
```

This confidence interval should look very similar to the one you computed in **Question 3**.

1.2 2. Testing the Central Limit Theorem

To recap the properties we just saw: The Central Limit Theorem tells us that the probability distribution of the **sum** or **average** of a large random sample drawn with replacement will be roughly normal, *regardless of the distribution of the population from which the sample is drawn.*

That's a pretty big claim, but the theorem doesn't stop there. It further states that the standard deviation of this normal distribution is given by

$$\frac{\text{sd of the original distribution}}{\sqrt{\text{sample size}}}$$

In other words, suppose we start with *any distribution* that has standard deviation x, take a sample of size n (where n is a large number) from that distribution with replacement, and compute the **mean** of that sample. If we repeat this procedure many times, then those sample means will have a normal distribution with standard deviation $\frac{x}{\sqrt{n}}$.

That's an even bigger claim than the first one! The proof of the theorem is beyond the scope of this class, but in this exercise, we will be exploring some data to see the CLT in action.

Question 1. Define the function `one_statistic_prop_heads` which should return exactly one simulated statistic of the proportion of heads from n coin flips.

```
BEGIN QUESTION
name: q2_1
manual: false
```

[2]:
```
coin_proportions = make_array(.5, .5) # our coin is fair

def one_statistic_prop_heads(n):
    # BEGIN SOLUTION
    simulated_proportions = sample_proportions(n, coin_proportions)
    prop_heads = simulated_proportions.item(0)
    return prop_heads
    # END SOLUTION
```

Question 2. The CLT only applies when sample sizes are "sufficiently large." This isn't a very precise statement. Is 10 large? How about 50? The truth is that it depends both on the original population distribution and just how "normal" you want the result to look. Let's use a simulation to get a feel for how the distribution of the sample mean changes as sample size goes up.

Consider a coin flip. If we say `Heads` is 1 and `Tails` is 0, then there's a 50% chance of getting a 1 and a 50% chance of getting a 0, which definitely doesn't match our definition of a normal distribution. The average of several coin tosses, where Heads is 1 and Tails is 0, is equal to the proportion of heads in those coin tosses (which is equivalent to the mean value of the coin tosses), so the CLT should hold **true** if we compute the sample proportion of heads many times.

Write a function called `sample_size_n` that takes in a sample size n. It should return an array that contains 5000 sample proportions of heads, each from n coin flips.

```
BEGIN QUESTION
name: q2_2
manual: false
```

[5]:
```
def sample_size_n(n):
    coin_proportions = make_array(.5, .5) # our coin is fair
    heads_proportions = make_array()
    for i in np.arange(5000):
        prop_heads = one_statistic_prop_heads(n) #SOLUTION
        heads_proportions = np.append(heads_proportions, prop_heads) #SOLUTION
    return heads_proportions
```

 The code below will use the function you just defined to plot the empirical distribution of the sample mean for various sample sizes. Drag the slider or click on the number to the right to type in a sample size of your choice. The x- and y-scales are kept the same to facilitate comparisons. Notice the shape of the graph as the sample size increases and decreases.

[8]:
```
# Just run this cell
from ipywidgets import interact

def outer(f):
    def graph(x):
        bins = np.arange(-0.01,1.05,0.02)
        sample_props = f(x)
        Table().with_column('Sample Size: {}'.format(x), sample_props).
 ↪hist(bins=bins)
        plt.ylim(0, 30)
        print('Sample SD:', np.std(sample_props))
        plt.show()
    return graph

interact(outer(sample_size_n), x=(0, 400, 1), continuous_update=False);

# Min sample size is 0, max is 400
# The graph will refresh a few times when you drag the slider around
```

```
interactive(children=(IntSlider(value=200, description='x', max=400), Output()), _dom_classes=
```

You can see that even the means of samples of 10 items follow a roughly bell-shaped distribution. A sample of 50 items looks quite bell-shaped.

Question 3: In the plot for a sample size of 10, why are the bars spaced at intervals of .1, with gaps in between?

```
BEGIN QUESTION
name: q2_3
manual: true
```

SOLUTION: In a sample of 10 things, the only possible proportions are 0, 0.1, 0.2, etc. We used bins with width 0.01, so most bins (like [0.43, 0.44)) can't have any items in them.

 Now we will test the second claim of the CLT: That the SD of the sample mean is the SD of the original distribution, divided by the square root of the sample size.

We have imported the flight delay data and computed its standard deviation for you.

[9]:
```
united = Table.read_table('united_summer2015.csv')
united_std = np.std(united.column('Delay'))
united_std
```

[9]:
```
39.480199851609314
```

Question 4: Write a function called `empirical_sample_mean_sd` that takes a sample size `n` as its argument. The function should simulate 500 samples with replacement of size `n` from the flight delays dataset, and it should return the standard deviation of the **means of those 500 samples**.

Hint: This function will be similar to the `sample_size_n` function you wrote earlier.

```
BEGIN QUESTION
name: q2_4
manual: false
```

[10]:
```
def empirical_sample_mean_sd(n):
    sample_means = make_array()
    for i in np.arange(500):
        sample = united.sample(n).column('Delay') #SOLUTION
        sample_mean = np.mean(sample) #SOLUTION
        sample_means = np.append(sample_means, sample_mean) #SOLUTION
    return np.std(sample_means)

empirical_sample_mean_sd(10)
```

[10]:
```
11.89093796804945
```

Question 5: Now, write a function called `predict_sample_mean_sd` to find the predicted value of the standard deviation of means according to the relationship between the standard deviation of the sample mean and sample size that is discussed here in the textbook. It takes a sample size `n` (a number) as its argument. It returns the predicted value of the standard deviation of the mean delay time for samples of size `n` from the flight delays (represented in the table `united`).

```
BEGIN QUESTION
name: q2_5
manual: false
```

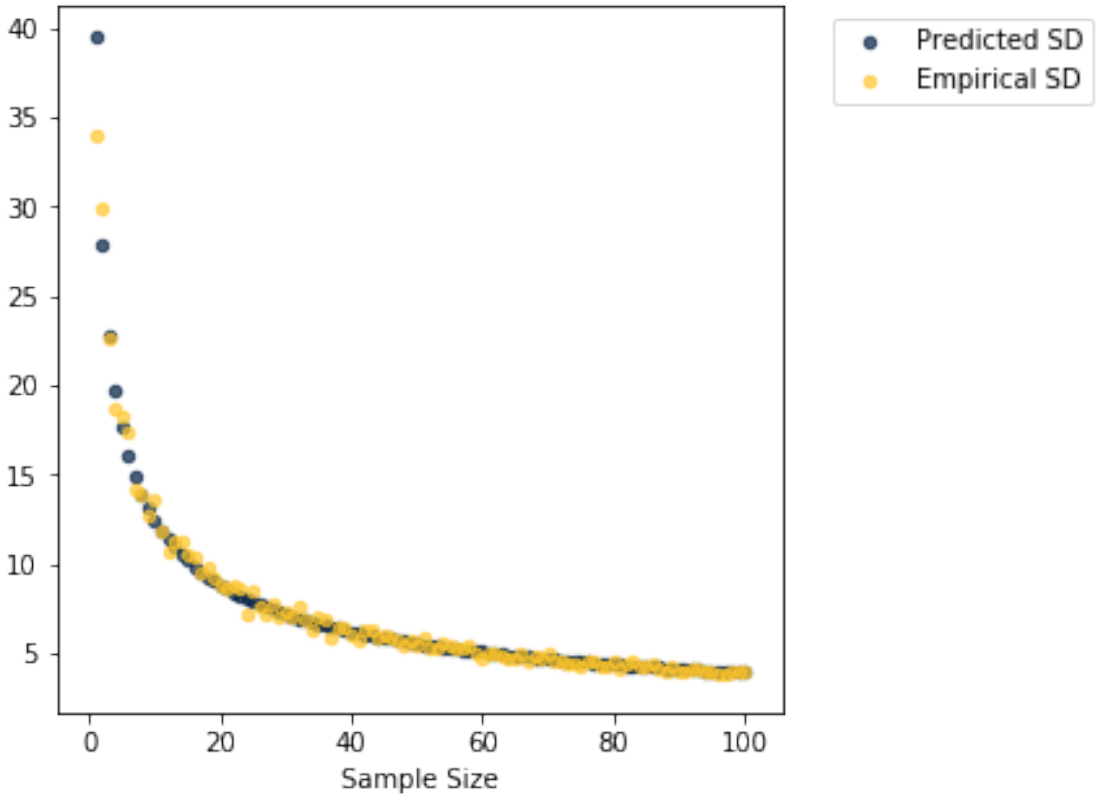
[13]:
```
def predict_sample_mean_sd(n):
    return united_std/(n**0.5) #SOLUTION

predict_sample_mean_sd(10)
```

[13]:
```
12.484735400972708
```

 The cell below will plot the predicted and empirical SDs for the delay data for various sample sizes.

[16]:
```
sd_table = Table().with_column('Sample Size', np.arange(1,101))
predicted = sd_table.apply(predict_sample_mean_sd, 'Sample Size')
empirical = sd_table.apply(empirical_sample_mean_sd, 'Sample Size')
sd_table = sd_table.with_columns('Predicted SD', predicted, 'Empirical SD',␣
↪empirical)
sd_table.scatter('Sample Size')
```



Question 6: Do our predicted and empirical values match? Why is this the case?

Hint: Are there any laws that we learned about in class that might help explain this?

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

SOLUTION: The law of averages tells us that the distribution of a large random sample should resemble the distribution from which it is drawn. If we compute 500 sample means, the distribution of those 500 means should be similar to the true distribution of the sample means. In particular, the standard deviation of the two distributions should be about the same. They won't generally be exactly the same, since the 500 sample means are generated randomly.

1.3 3. Polling and the Normal Distribution

Question 1 Michelle is a statistical consultant, and she works for a group that supports Proposition 68 (which would mandate labeling of all horizontal or vertical axes), called Yes on 68. They want to know how many Californians will vote for the proposition.

Michelle polls a uniform random sample of all California voters, and she finds that 210 of the 400 sampled voters will vote in favor of the proposition. Fill in the code below to form a table with 3 columns: the first two columns should be identical to `sample`. The third column should be named `Proportion` and have the proportion of total voters that chose each option.

```
BEGIN QUESTION
name: q3_1
manual: false
```

[2]:
```
sample = Table().with_columns(
    "Vote",  make_array("Yes", "No"),
    "Count", make_array(210,   190))
sample_size = sum(sample.column("Count")) #SOLUTION
sample_with_proportions = sample.with_column("Proportion", sample.
 ↪column("Count") / sample_size) #SOLUTION
sample_with_proportions
```

[2]:
```
Vote | Count | Proportion
Yes  | 210   | 0.525
No   | 190   | 0.475
```

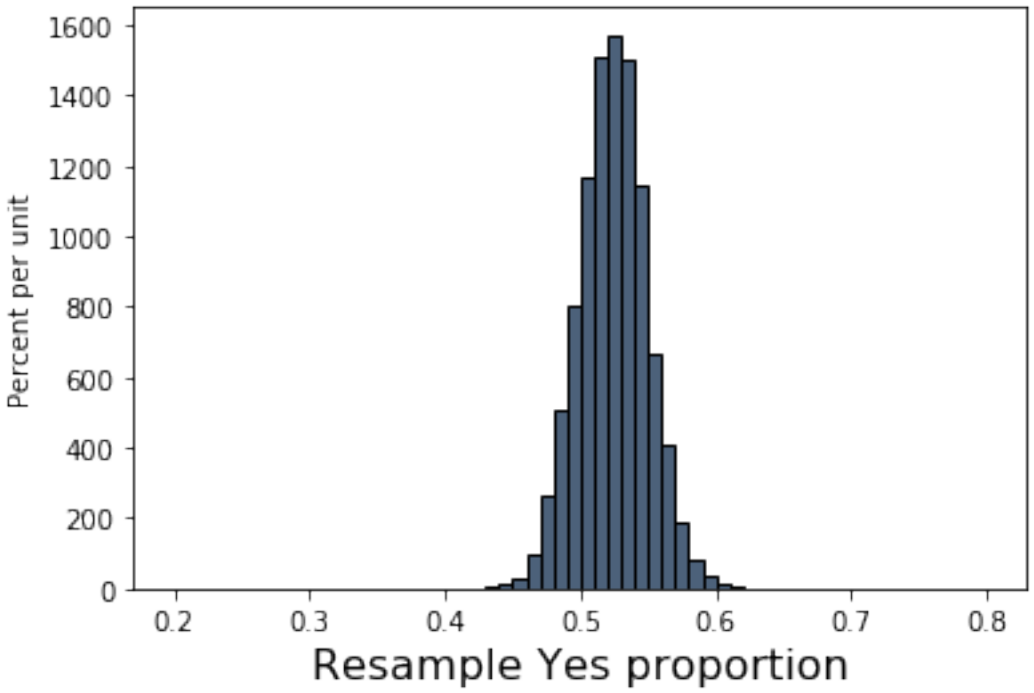
Question 2 Michelle then wants to use 10,000 bootstrap resamples to compute a confidence interval for the proportion of all California voters who will vote Yes.

Fill in the next cell to simulate an empirical distribution of Yes proportions. Use bootstrap resampling to simulate 10,000 election outcomes, and populate `resample_yes_proportions` with the yes proportion of each bootstrap resample. Then, visualize `resample_yes_proportions` with a

histogram. **You should see a bell shaped curve centered near the proportion of Yes in the original sample.**

```
BEGIN QUESTION
name: q3_2
manual: false
```

[4]:
```
resample_yes_proportions = make_array()
for i in np.arange(10000):
    resample = sample_proportions(sample_size, sample_with_proportions.
 ↪column("Proportion")) #SOLUTION
    resample_yes_proportions = np.append(resample_yes_proportions, resample.
 ↪item(0)) #SOLUTION
Table().with_column("Resample Yes proportion", resample_yes_proportions).
 ↪hist(bins=np.arange(.2, .8, .01))
```



Question 3 Why does the Central Limit Theorem (CLT) apply in this situation, and how does it explain the distribution we see above?

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

SOLUTION: If we think of Yes votes as 1 and No votes as 0, then the sample is a collection of numbers. A resample is sampled with replacement from that collection, so a resample mean is the

mean of a sample with replacement from some collection of numbers. The CLT therefore applies to the resample means: across resamples, they will have an approximately normal distribution. That's why the histogram above is bell-shaped.

 In a population whose members are 0 and 1, there is a simple formula for the standard deviation of that population:

$$\texttt{standard deviation} = \sqrt{(\text{proportion of 0s}) \times (\text{proportion of 1s})}$$

(Figuring out this formula, starting from the definition of the standard deviation, is an fun exercise for those who enjoy algebra.)

Question 4 Using only the CLT and the numbers of Yes and No voters in our sample of 400, compute (*algebraically*) a number `approximate_sd` that's the predicted standard deviation of the array `resample_yes_proportions` according to the Central Limit Theorem. **Do not access the data in `resample_yes_proportions` in any way.**

Remember that the standard deviation of the sample means can be computed from the population SD and the size of the sample. If we do not know the population SD, we can use the sample SD as a reasonable approximation in its place.

```
BEGIN QUESTION
name: q3_4
manual: false
```

[6]:
```
approximate_sd = ((210/400) * (190/400) / 400) ** 0.5 # SOLUTION
approximate_sd
```

[6]:
```
0.02496873044429772
```

Question 5 Compute the SD of the array `resample_yes_proportions` which will act as an approximation to the true SD of the possible sample proportions. This will help verify whether your answer to question 2 is approximately correct.

```
BEGIN QUESTION
name: q3_5
manual: false
```

[9]:
```
exact_sd = np.std(resample_yes_proportions) #SOLUTION
exact_sd
```

[9]:
```
0.02497825948575081
```

Question 6 **Again, without accessing `resample_yes_proportions` in any way**, compute an approximate 95% confidence interval for the proportion of Yes voters in California.

The cell below draws your interval as a red bar below the histogram of `resample_yes_proportions`; use that to verify that your answer looks right.

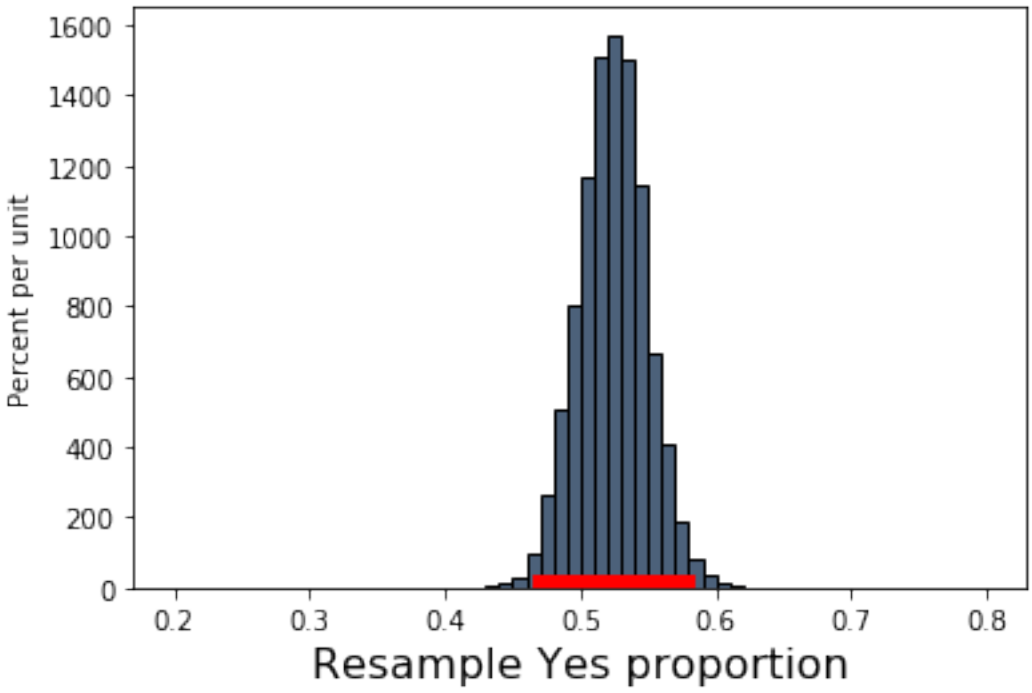
Hint: How many SDs corresponds to 95% of the distribution promised by the CLT? Recall the discussion in the textbook here.

```
BEGIN QUESTION
name: q3_6
manual: false
```

[12]:
```
lower_limit = 210/400 - 2*approximate_sd #SOLUTION
upper_limit = 210/400 + 2*approximate_sd #SOLUTION
print('lower:', lower_limit, 'upper:', upper_limit)
```

```
lower: 0.47506253911140456 upper: 0.5749374608885954
```

[16]:
```
# Run this cell to plot your confidence interval.
Table().with_column("Resample Yes proportion", resample_yes_proportions).
 ↪hist(bins=np.arange(.2, .8, .01))
plt.plot(make_array(lower_limit, upper_limit), make_array(0, 0), c='r', lw=10);
```



 Your confidence interval should overlap the number 0.5. That means we can't be very sure whether Proposition 68 is winning, even though the sample Yes proportion is a bit above 0.5.

The Yes on 68 campaign really needs to know whether they're winning. It's impossible to be absolutely sure without polling the whole population, but they'd be okay if the standard deviation of the sample mean were only 0.005. They ask Michelle to run a new poll with a sample size that's large enough to achieve that. (Polling is expensive, so the sample also shouldn't be bigger than necessary.)

Michelle consults Chapter 14 of your textbook. Instead of making the conservative assumption that the population standard deviation is 0.5 (coding Yes voters as 1 and No voters as 0), she decides to assume that it's equal to the standard deviation of the sample,

$$\sqrt{(\text{Yes proportion in the sample}) \times (\text{No proportion in the sample})}.$$

Under that assumption, Michelle decides that a sample of 9,975 would suffice.

Question 7 Does Michelle's sample size achieve the desired standard deviation of sample means? What SD would you achieve with a smaller sample size? A higher sample size? To explore this, first compute the SD of sample means obtained by using Michelle's sample size.

```
BEGIN QUESTION
name: q3_7
manual: false
```

[17]:
```
estimated_population_sd = np.sqrt((210/400)*(1 - 210/400)) #SOLUTION
michelle_sample_size = 9975 #SOLUTION
michelle_sample_mean_sd = estimated_population_sd/(michelle_sample_size**.5)␣
↪#SOLUTION
print("With Michelle's sample size, you would predict a sample mean SD of %f."␣
↪% michelle_sample_mean_sd)
```

```
With Michelle's sample size, you would predict a sample mean SD of 0.005000.
```

Question 8

Then, compute the SD of sample means that you would get from a smaller sample size. Ideally, you should pick a number that is significantly smaller, but any sample size smaller than Michelle's will do.

```
BEGIN QUESTION
name: q3_8
manual: false
```

[19]:
```
smaller_sample_size = 5000 #SOLUTION
smaller_sample_mean_sd = estimated_population_sd/(smaller_sample_size**.5)␣
↪#SOLUTION
print("With this smaller sample size, you would predict a sample mean SD of %f"␣
↪% smaller_sample_mean_sd)
```

```
With this smaller sample size, you would predict a sample mean SD of 0.007062
```

[20]:
```
# TEST
smaller_sample_size < michelle_sample_size
```

[20]:
```
True
```

Question 9

Finally, compute the SD of sample means that you would get from a larger sample size. Here, a number that is significantly larger would make any difference more obvious, but any sample size larger than Michelle's will do.

```
BEGIN QUESTION
name: q3_9
manual: false
```

[21]:
```
larger_sample_size = 15000 #SOLUTION
larger_sample_mean_sd = estimated_population_sd/(larger_sample_size**.5)␣
↪#SOLUTION
print("With this larger sample size, you would predict a sample mean SD of %f"␣
↪% larger_sample_mean_sd)
```

```
With this larger sample size, you would predict a sample mean SD of 0.004077
```

Question 10 Based off of this, was Michelle's sample size approximately the minimum sufficient sample, given her assumption that the sample SD is the same as the population SD? Assign `min_sufficient` to `True` if this 9975 was indeed approximately the minimum sufficient sample, and `False` if it wasn't.

```
BEGIN QUESTION
name: q3_10
manual: false
```

[24]:
```
min_sufficient = True #SOLUTION
min_sufficient
```

[24]:
```
True
```

1.4 4. Submission

Once you're finished, select "Save and Checkpoint" in the File menu and then execute the `submit` cell below. The result will contain a link that you can use to check that your assignment has been submitted successfully. If you submit more than once before the deadline, we will only grade your final submission. If you mistakenly submit the wrong one, you can head to okpy.org and flag the correct version. To do so, go to the website, click on this assignment, and find the version you would like to have graded. There should be an option to flag that submission for grading!

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

[ ]:
```
# 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.")
```