hw09

April 10, 2020

1 Homework 9: 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 9 is due **Thursday, 4/9 at 11:59pm**.

Start early so that you can come to office hours if you're stuck. Late work will not be accepted as per the course policies.

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.

[1]:
```
# 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)
```

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.

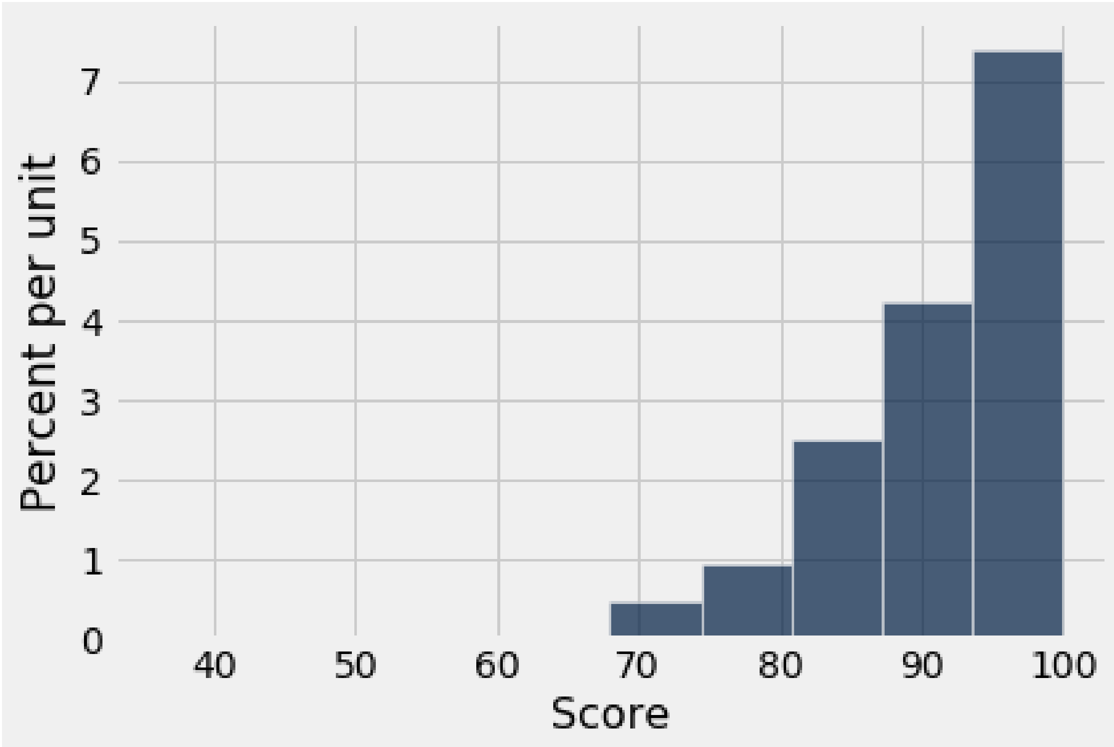
[2]:
```
# Just run this cell.
pop_restaurants = Table.read_table('restaurant_inspection_scores.csv').drop(5,6)
```

```
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)
```

Question 1.1. Plot a histogram of the scores in the cell below.

[3]:
```
# Write your code here.
pop_restaurants.hist('Score')
```



This is the **population mean**:

[4]:
```
pop_mean = np.mean(pop_restaurants.column(3))
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.

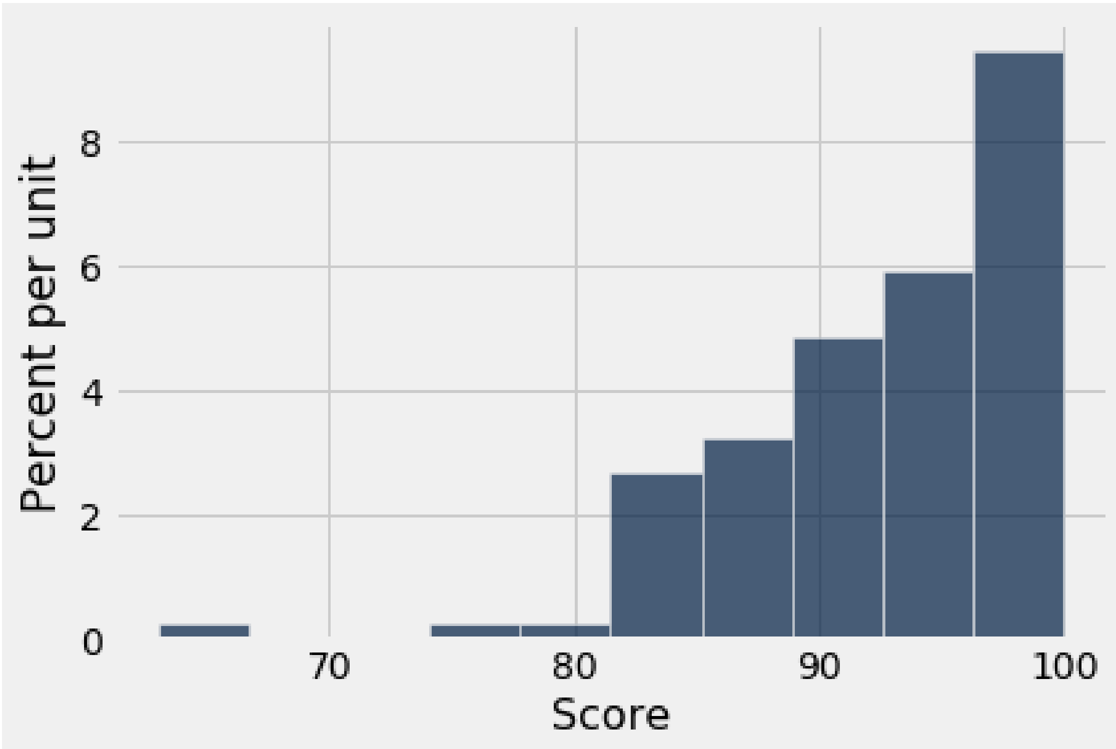
[5]:
```
restaurant_sample = pop_restaurants.sample(100, with_replacement=False)
restaurant_sample
```

[5]:
```
Restaurant Name             | Zip Code | Inspection Date | Score | Address
Bertha Sadler Means YWLA    | 78723    | 05/05/2015      | 100   | 6401 NORTH
HAMPTON DR
AUSTIN, TX 78723
(30.314467, -97.6 ...
My Fit Foods                | 78753    | 07/14/2015      | 84    | 500 CANYON
RIDGE DR Bldg H
AUSTIN, TX 78753
(30.403174,  ...
Jimmy John's # 1293         | 78753    | 05/04/2016      | 89    | 1100 CENTER
RIDGE DR Unit 300
AUSTIN, TX 78753
(30.41772 ...
```

```
Project Transitions, Inc  | 78756    | 01/08/2015     | 100   | 5606 ROOSEVELT
AVE
AUSTIN, TX 78756
(30.327671, -97.73208)
Papalote Taco House       | 78729    | 01/07/2016     | 95    | 13219 N US 183
HWY NB Unit 100
AUSTIN, TX 78729
(30.4619 ...
Tokyo Gardens Catering LLC | 78756    | 08/04/2016     | 96    | 5808 BURNET RD
AUSTIN, TX 78756
(30.334115, -97.739948)
HEB  Deli & Bakery        | 78758    | 01/19/2016     | 92    | 9414 N LAMAR
BLVD
AUSTIN, TX 78758
(30.364881, -97.695995)
Austin Shell              | 78702    | 08/28/2015     | 94    | 701 N IH 35
SVRD NB
AUSTIN, TX 78702
(30.266764, -97.733784)
Dollar General Store #4031 | 78752    | 05/16/2014     | 91    | 6929 AIRPORT
BLVD
AUSTIN, TX 78752
(30.336574, -97.718265)
Chicken Express           | 78726    | 07/23/2015     | 100   | 8300 N FM 620
RD Bldg H
AUSTIN, TX 78726
(30.45313, -97. ...
... (90 rows omitted)
```

Question 1.2. Plot a histogram of the **sample** scores in the cell below.

[6]:
```
# Write your code here:
restaurant_sample.hist('Score')
```



This is the **sample mean**:

[7]:
```
sample_mean = np.mean(restaurant_sample.column(3))
sample_mean
```

[7]:
```
92.67
```

Question 1.3. Complete the function `bootstrap_scores` below. It should take no arguments. It should simulate drawing 5000 resamples from `restaurant_sample` and computing the mean restaurant score in each resample. It should return an array of those 5000 resample means.

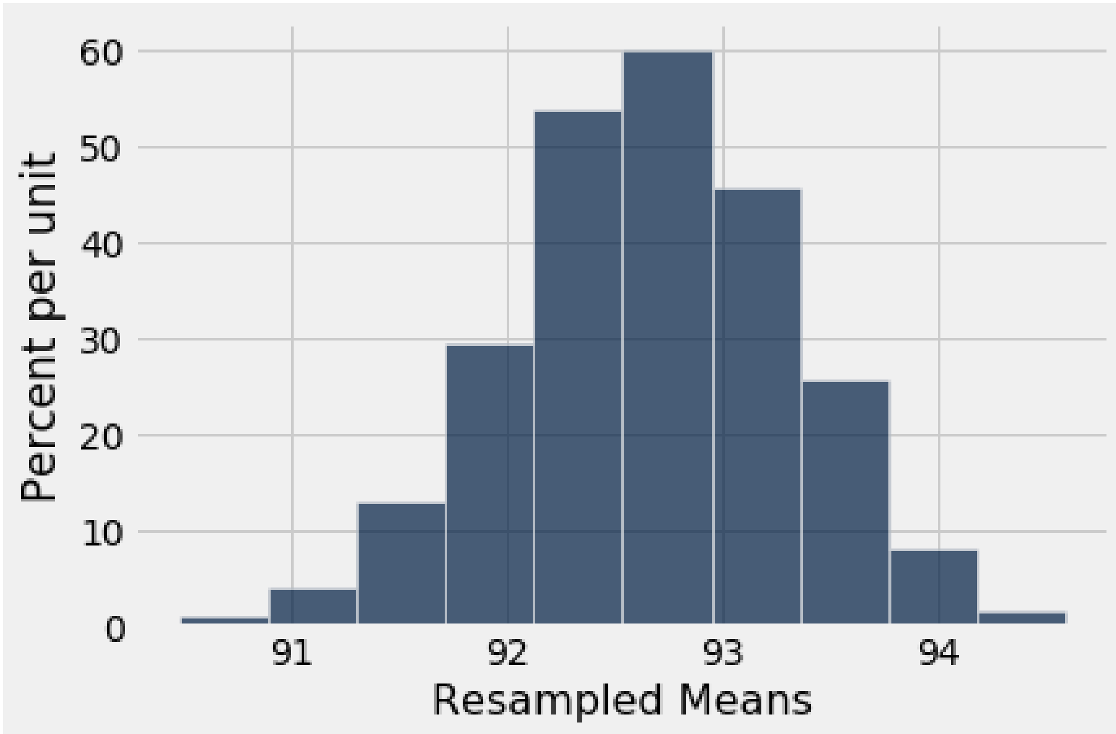
[8]:
```
def bootstrap_scores():
    resampled_means = make_array ()
    for i in range(5000):
        resampled_mean = np.mean(restaurant_sample.sample().column(3))
        resampled_means = np.append(resampled_means,resampled_mean)
    return resampled_means

resampled_means = bootstrap_scores()
resampled_means
```

[8]:
```
array([91.72, 93.11, 92.31, ..., 93.09, 92.34, 92.12])
```

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

[9]:
```
Table().with_column('Resampled Means', resampled_means).hist()
```



Question 1.4. Compute a 95 percent confidence interval for the average restaurant score using the array `resampled_means`.

[10]:
```
lower_bound = percentile (2.5, resampled_means)
upper_bound = percentile (97.5, resampled_means)
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:
( 91.36 , 93.88 )
```

Question 1.5. Does the distribution of the resampled mean scores look normally distributed? State "yes" or "no" and describe in one sentence why you would expect that result.

Yes, because the central limit theorem says that the probability disrbution of the average of a random sample drawn with replacement will be roughly normal **Question 1.6.** 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! *No. The distribution of the sampled scores resemble those of the population yet the population scores are not normally distributed*

For the last question, 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.

Question 1.7. Without referencing the array `resampled_means` or performing any new simulations, calculate an interval around the `sample_mean` that covers approximately 95% of the numbers in the `resampled_means` array. **You may use the following values to compute your result, but you should not perform additional resampling** - think about how you can use the CLT to accomplish this.

[11]:
```
sample_mean = np.mean(restaurant_sample.column(3))
sample_sd = np.std(restaurant_sample.column(3))
sample_size = restaurant_sample.num_rows

lower_bound_normal = sample_mean - 2*(sample_sd/sample_size ** 0.5)
upper_bound_normal = sample_mean + 2*(sample_sd/sample_size ** 0.5)
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:
( 91.39499254904138 , 93.94500745095863 )
```

This confidence interval should look very similar to the one you computed in **Question 1.4**.

1.2 2. Testing the Central Limit Theorem

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

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

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 2.1. 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.

[12]:
```
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):
        simulated_proportions = sample_proportions (n, coin_proportions)
        prop_heads = simulated_proportions.item (0)
        heads_proportions = np.append(heads_proportions, prop_heads)
    return heads_proportions


sample_size_n(5)
```

[12]:
```
array([0.6, 0.8, 0.6, ..., 0.4, 0.8, 0.2])
```

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.

[13]:
```
# 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 2.2. In the plot for a sample size of 10, why are the bars spaced at intervals of .1, with gaps in between?

There are only 10 items in the sample hence the increments can only be .1

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.

[14]:
```
united = Table.read_table('united_summer2015.csv')
united_std = np.std(united.column('Delay'))
united_std
```

[14]:
```
39.480199851609314
```

Question 2.3. 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.

[15]:
```
def empirical_sample_mean_sd(n):
    sample_means = make_array()
    for i in np.arange(500):
        sample = united.sample (n).column('Delay')
        sample_mean = np.mean(sample)
        sample_means = np.append (sample_means, sample_mean)
    return np.std(sample_means)

empirical_sample_mean_sd(10)
```

[15]:
```
12.264000815394624
```

Question 2.4. 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`).

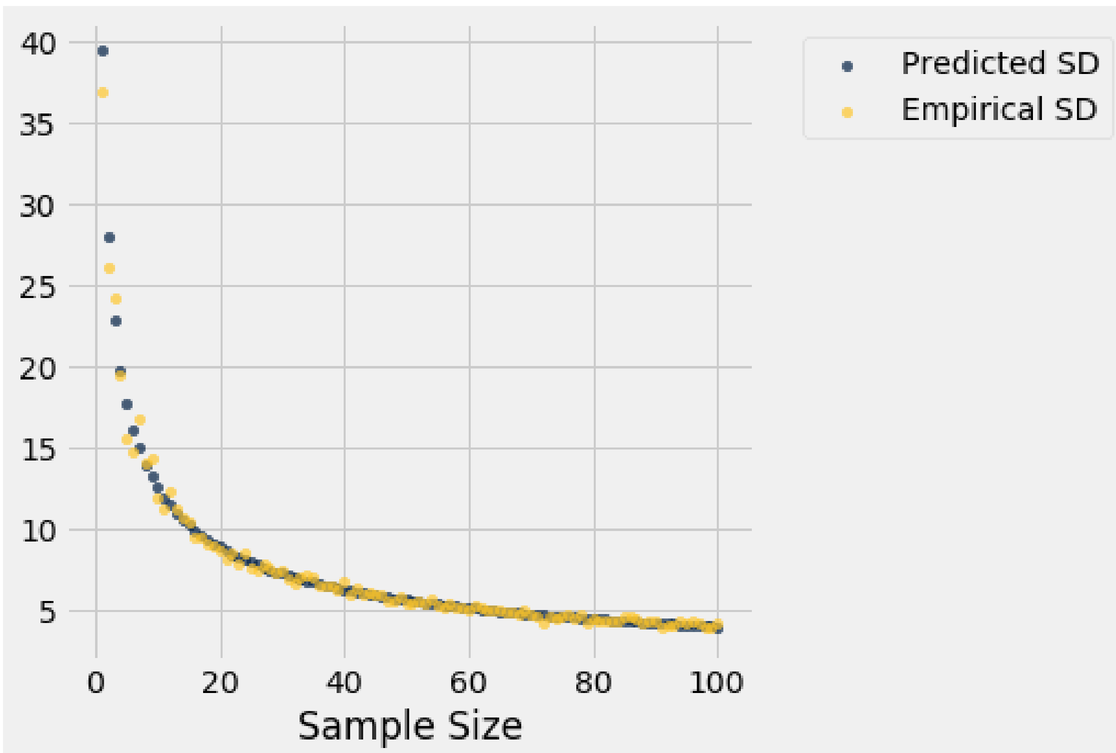
[16]:
```
def predict_sample_mean_sd(n):
    return united_std / (n) ** 0.5

predict_sample_mean_sd(10)
```

[16]:
```
12.484735400972708
```

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

[17]:
```
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 2.5. 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?

Yes they closely resemble because the sample size is large and according to the law of averages, the sample distribution should resemble the true distribution of the sample means.

1.3 3. Polling and the Normal Distribution

Question 3.1. 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.

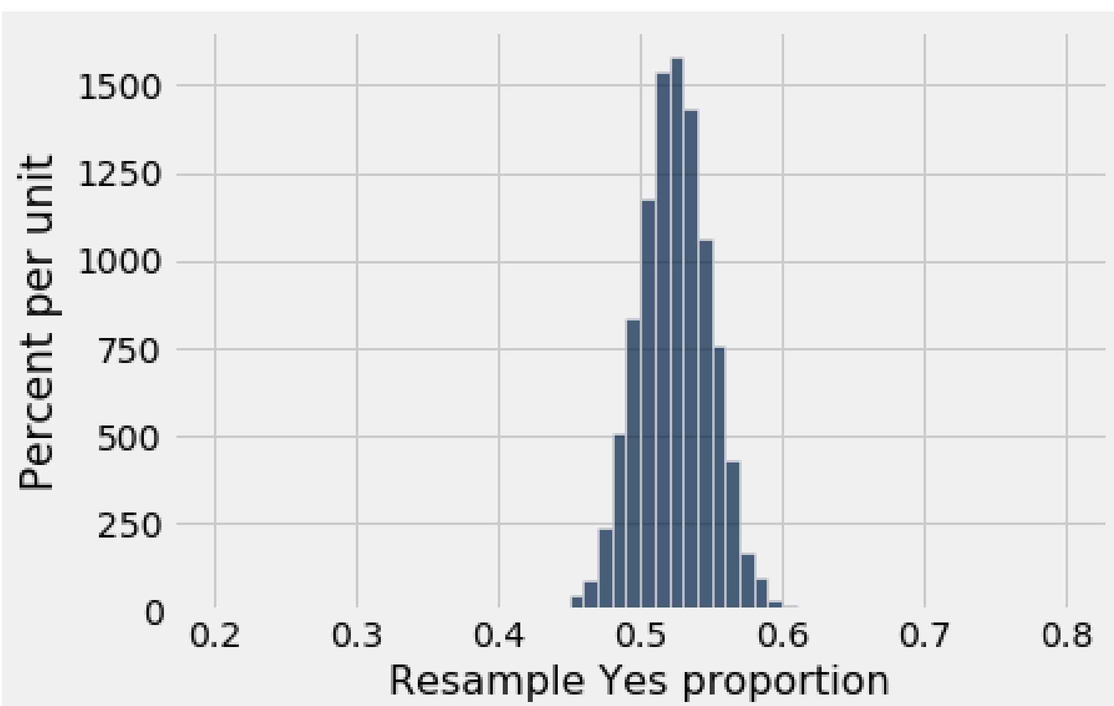
Michelle polls a uniform random sample of all California voters, and she finds that 210 of the 400 sampled voters will vote in favor of the proposition. Fill in the code below to form a table with 3 columns: the first two columns should be identical to `sample`. The third column should be named `Proportion`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

[18]:
```
sample = Table().with_columns(
    "Vote",  make_array("Yes", "No"),
    "Count", make_array(210,   190))
sample_size = 400
sample_with_proportions = sample.with_columns ('Proportion',␣
 ↪make_array(210,190)/sample_size)
sample_with_proportions
```

[18]:
```
Vote | Count | Proportion
Yes  | 210   | 0.525
No   | 190   | 0.475
```

Question 3.2. She 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 with 10,000 resamples. In other words, 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.

[19]:
```
resample_yes_proportions = make_array()
for i in np.arange(10000):
    resample = sample_proportions (400, sample_with_proportions.
 ↪column('Proportion'))
    resample_yes_proportions = np.append (resample_yes_proportions, resample.
 ↪item(0))
Table().with_column("Resample Yes proportion", resample_yes_proportions).
 ↪hist(bins=np.arange(.2, .8, .01))
```



Question 3.3. Why does the Central Limit Theorem (CLT) apply in this situation, and how does it explain the distribution we see above?

The resample is sampled with replacement from the sample. Therefore, a resample mean would be the mean of a sample with replacement. Because of this, the CLT would apply to resample means as well and the distribution would be approximately normal.

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 3.4. 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 a predicted standard deviation of the sample means can be computed from the population SD and the size of the sample.

Also remember that if we do not know the population SD, we can use the sample SD as a reasonable approximation in its place.

[21]:
```
approximate_sd = ((210/400) * (190/400) / 400) ** 0.5
approximate_sd
```

[21]:
```
0.024968730444297772
```

Question 3.5. 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 3.2 is approximately correct.

[22]:
```
exact_sd = np.std (resample_yes_proportions)
exact_sd
```

[22]:
```
0.024993116391068577
```

Question 3.6. Again, without accessing `resample_yes_proportions` in any way, compute an approximate 95% confidence interval for the proportion of Yes voters in California using the Central Limit Theorem and `approximate_sd`.

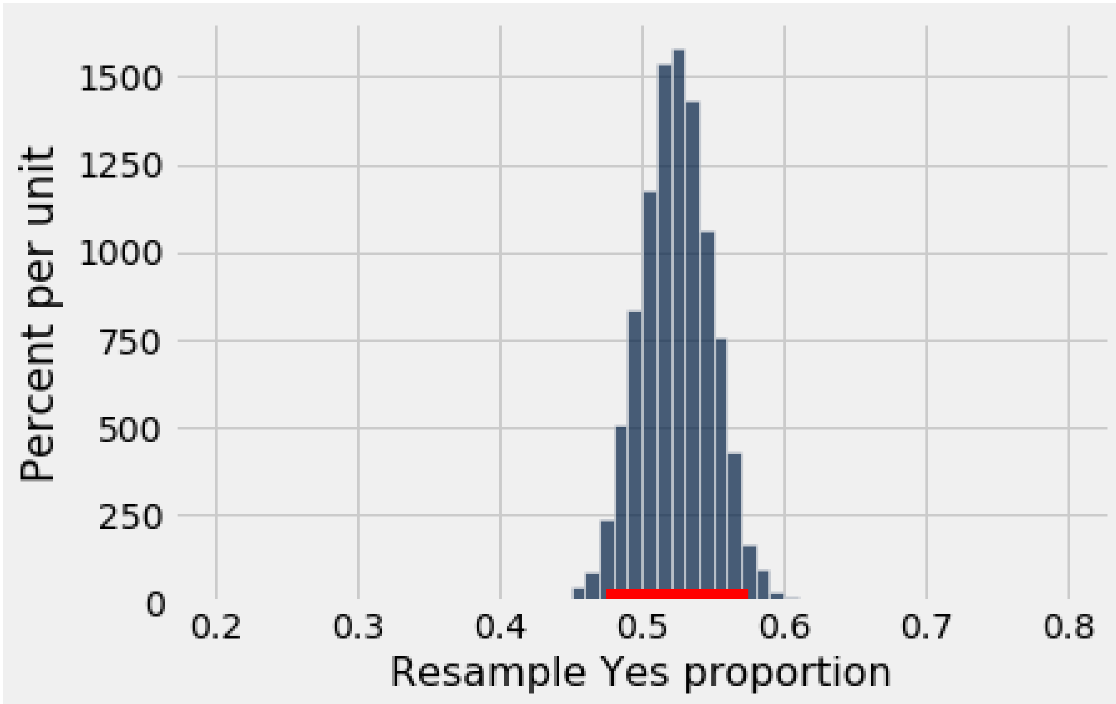
The cell below draws your interval as a red bar below the histogram of `resample_yes_proportions`; use that to verify that your answer looks right.

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

[23]:
```
lower_limit = 210/400 - 2 * approximate_sd
upper_limit = 210/400 + 2 * approximate_sd
print('lower:', lower_limit, 'upper:', upper_limit)
```

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

[24]:
```
# 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);
```



1500
1250
1000
750
500
250
0
Percent per unit
0.2
0.3
0.4
0.5
0.6
0.7
0.8
Resample Yes proportion

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 3.7. 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.

[26]:
```
estimated_population_sd = exact_sd * (400) ** (.5)
michelle_sample_size = 9975
michelle_sample_mean_sd = np.average(estimated_population_sd /␣
 ↪michelle_sample_size ** 0.5)
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 0.005005.
```

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.

[28]:
```
smaller_sample_size = 500
smaller_sample_mean_sd = np.average (estimated_population_sd / 500 **0.5)
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 0.022355
```

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.

[30]:
```
larger_sample_size = 50000
larger_sample_mean_sd = np.average (estimated_population_sd / 50000 **0.5)
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 0.002235
```

Question 3.8. 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.

[31]:
```
min_sufficient = True
min_sufficient
```

[31]:
```
True
```

1.4 4. Submission

Once you're finished, submit your assignment as a .ipynb (Jupyter Notebook) and .pdf (download as .html, then print to save as a .pdf) on the class Canvas site.

[ ]:
```
```