Central Limit Theorem and Confidence Intervals - Introduction

Introduction

In this section, you'll be introduced to inferential statistics. You'll learn about sampling, the central limit theorem, and the T-distribution.

Distributions and Sampling

In this section, we're returning to statistics to broaden and deepen our understanding of distributions and sampling.

Sampling

We'll start by providing an introduction to the idea of ***Sampling*** - selecting a subset of a population to survey. We'll then start to introduce some statistics related to sampling by explaining and showing how to calculate the standard error.

The Central Limit Theorem

Once we understand a bit about sampling, we'll explore how we can use it by digging deep into one of the coolest and most important concepts in inferential statistics--the ***Central Limit Theorem***! We'll start by learning about how the Central Limit Theorem works, and explore how we can use it in a way that allows us to treat non-normal distributions as normal distributions, and provides a way for us to estimate parameters about a population.

The T-Distribution

Finally, we'll end this section by learning about how we can use the ***T-Distribution*** for dealing with samples that are smaller, and that have an unknown standard deviation. We'll explore how the T-Distribution works, learn about *degrees of freedom*, and then see how we can calculate confidence intervals using our newfound knowledge of the T-Distribution.

Summary

While some of this material may seem a little dry, a deep understanding of and intuition for distributions and sampling will be important in your career as a data scientist. This knowledge will help you avoid making mistakes in your EDA (exploratory data analysis), feature selection, and modeling work, which could lead to faulty predictions from your models.

Introduction to Sampling

Introduction

Rarely, if ever, are we able to completely survey a population of interest. Similarly, we will often deal with missing data. Whatever it may be, whether estimating asthma rates, fish populations, daily temperatures, material volumes, risk, manufacturing defects or any other measurement of unknown or large scale quantities, we are unlikely to have complete information of the system in question. As a result, we do our best by taking samples and using these to estimate the corresponding measurements for the complete population, from which we took the sample. These estimates of population parameters are known as **point estimates**. Interestingly, point estimates of specific parameters of a population have predictable behaviors, in that the point estimates themselves will form specific probability distributions. For example, we may want to know information about the age of a population. One parameter we might want to estimate is the mean age of the population. Once we take a sample, we can take the mean age of that sample and that would become the point estimate for the mean age of the entire population. If we continue to take more samples from the population, the mean age of each of these samples will begin to form a normal distribution! This intriguing fact lets us apply some logic and calculate confidence intervals surrounding our point estimates so that we not only have a best guess for the parameter, but also can have a range to describe various levels of certainty for our estimates. Ideally, these ranges will be small, indicating that we have a high degree of confidence that the parameter is very close to our estimate.

Objectives

You will be able to:

- Describe how samples are able to allow data scientists to gain insights to a population

Let's start by importing a dataset to use for demonstration. In this case, we'll use a datafile concerning individuals who were on board the Titanic. We'll use this as our entire population and start to observe how the point estimates from various samples of this population behave.

In [1]:

```
import pandas as pd
import numpy as np
df = pd.read_csv('titanic.csv')
print(len(df))
df.head()
```

```
891
```

Out[1]:

	Unnamed: 0	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

If we take a look at the population mean age we have:

In [2]:

```
df.Age.mean()
```

Out[2]:

```
29.69911764705882
```

Let's see what happens when we take a sample in order to estimate this population parameter. (Again remember, this is called a point estimate!)

In [3]:

```
sample = df.sample(n=50, random_state=22) #Take a sample of 50 people
sample.Age.mean() #Calculate the sample mean
```

Out[3]:

```
27.79268292682927
```

It's not a bad estimate, even though it's not exact. From here we can start to ask many questions related to how confident we are in this estimate. A first simple approach is to quantify our estimate. We'll first look at the percent error:

In [4]:

```
err = np.abs(sample.Age.mean() - df.Age.mean())
per_err = err / df.Age.mean()
print(per_err)
```

```
0.06419162827951391
```

As it stands, our estimate is close but about 6% off of the actual figure. We might start to wonder whether this is a *normal* or expected error for our sample to be off. Can we say that a sample of 50 from a population of roughly 900 will always produce a point estimate this accurate? To simulate this, let's repeat this process of taking a sample (let's stick with 50 people for now) and save all of these sample means and see what happens.

In [5]:

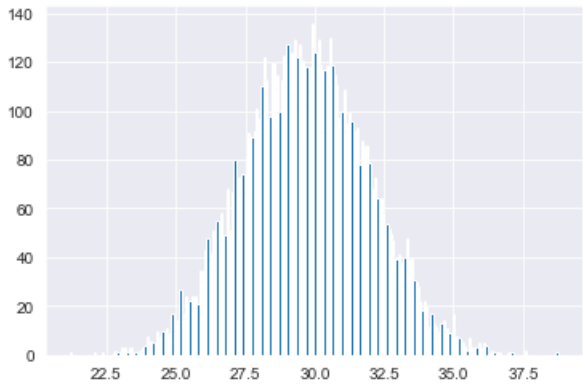
```
sample_means = []
for i in range(10**4):
    sample = df.sample(n=50, random_state=i) #Take a sample of 50 people
    sample_means.append(sample.Age.mean()) #Calculate the sample mean
```

The first thing we'll look at is the distribution of our sample means.

In [6]:

```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

sns.set_style('darkgrid') #Pretty background including grid lines for our backdrop
plt.hist(sample_means, bins=250);
```



Interesting! The first thing to note here is that the sample means form a normal distribution! What's more, let's take a look at the mean of our sample means:

In [7]:

```
import numpy as np
np.mean(sample_means)
```

Out[7]:

```
29.678139189972246
```

Wow! Look at that! The mean of our sample means is extremely close to the actual mean of the population! The mean of means of this simulation shows an accuracy of 99.9%

In [8]:

```
population_mean = df.Age.mean()
mean_sample_means = np.mean(sample_means)
acc = 1 - (np.abs(mean_sample_means - population_mean) / population_mean)
print(acc)
```

```
0.9992936336582157
```

Summary

We've begun to investigate and think about how we can gain knowledge about a population using sampling techniques. From there, we further explored how the estimates provided by samples can themselves be analyzed as a mathematical distribution. These observations serve as the intuition behind confidence intervals. In future sections, we'll talk about point estimators (including parameters other than the mean such as standard deviation) and confidence intervals in more detail!

Central Limit Theorem

Introduction

In this lesson, we'll start to investigate a *central* statistical concept; the central limit theorem! (And how to write a good dry math pun.)

Objectives

You will be able to:

- Describe how the central limit theorem is related to sampling
- Describe how the central limit theorem can be used for parameter estimation

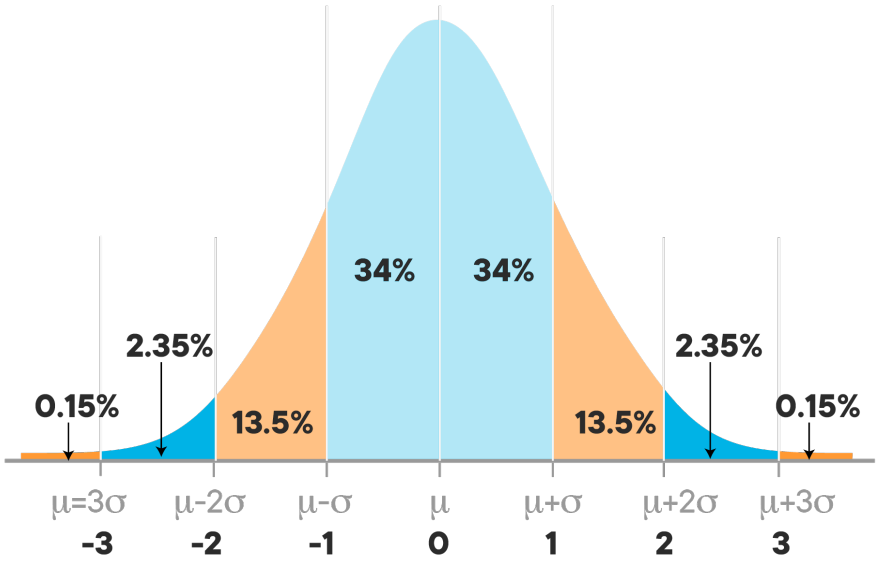
What does the Central Limit Theorem stand for?

The central limit theorem states that, under many conditions, independent random variables summed together will converge to a normal distribution as the number of variables increases. This becomes very useful for applying statistical logic to sample statistics in order to estimate population parameters. For example, as we saw in the previous lecture, the averages of samples will form a normal distribution. We can then use this information to put further bounds on our estimates of the population. We can also use this information to estimate the probability of samples taking on extreme values that deviate from the population mean.

For example, let's say that we know the mean and standard deviation of asthma rates in the United States. If we then take a sample from a specific city and find that the mean of this sample is substantially lower than that of the overall population, we may be interested in questions such as "what is the probability that this was just caused by random chance in sampling?" If the probability is exceedingly low, we have further reason to believe that this city has higher rates of asthma and that its population is statistically different then that of the general population.

The computation would be something like this: we know the mean population, and by the central limit theorem, the average of various samples takes on a normal distribution. From that normal distribution of sample means, we can then compare the mean of our actual sample and compare it to the distribution of means. It should be quite rare that our sample mean falls outside 2 or 3 standard deviations from the mean of sample means, (roughly 2.35% and .15% respectively for each tail). As such, having a sample mean that falls outside of these scopes is worthy of further investigation.

For reference, here's is a rough empirical rule for percentiles within a normal distribution. (And again, by the central limit theorem, we expect our sample means to take on a normal distribution!)



Additional Resources

http://sphweb.bumc.bu.edu/otlt/MPH-Modules/BS/BS704_Probability/BS704_Probability13.html

Summary

Summary

In this brief lesson, we continued to discuss the central limit theorem and its application for sampling statistics and confidence intervals.

Central Limit Theorem - Lab

Introduction

In this lab, we'll learn how to use the Central Limit Theorem to work with non-normally distributed datasets as if they were normally distributed.

Objectives

You will be able to:

- Use built-in methods to detect non-normal datasets
- Create a sampling distribution of sample means to demonstrate the central limit theorem

Let's get started!

First, import the required libraries:

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import scipy.stats as st
np.random.seed(0) #set a random seed for reproducibility
```

Next, read in the dataset. A dataset of 10,000 numbers is stored in `non_normal_dataset.csv`. Use pandas to read the data into a series.

Hint: Any of the `read_` methods in pandas will store 1-dimensional in a Series instead of a DataFrame if passed the optimal parameter `squeeze=True`.

In [2]:

```
# Your code here
```

Detecting Non-Normal Datasets

Before we can make use of the normal distribution, we need to first confirm that our data is normally distributed. If it is not, then we'll need to use the Central Limit Theorem to create a sample distribution of sample means that will be normally distributed.

There are two main ways to check if a sample follows the normal distribution or not. The easiest is to simply plot the data and visually check if the data follows a normal curve or not.

In the cell below, use `seaborn`'s `distplot` method to visualize a histogram of the distribution overlaid with the probability density curve.

In [3]:

```
# Your code here
```

As expected, this dataset is not normally distributed.

For a more formal way to check if a dataset is normally distributed or not, we can make use of a statistical test. There are many different statistical tests that can be used to check for normality, but we'll keep it simple and just make use of the `normaltest()` function from `scipy.stats`, which we imported as `st` --see the documentation if you have questions about how to use this method.

In the cell below, use `normaltest()` to check if the dataset is normally distributed.

In [4]:

```
# Your code here
```

The output may seem a bit hard to interpret since we haven't covered hypothesis testing and p-values in further detail yet. However, the function tests the hypothesis that the distribution passed into the function differs from the normal distribution. The null hypothesis would then be that the data *is* normally distributed. We typically reject the null hypothesis if the p-value is less than 0.05. For now, that's all you need to remember--this will make more sense once you work with p-values more which you'll do subsequently.

Since our dataset is non-normal, that means we'll need to use the ***Central Limit Theorem.***

Sampling With Replacement

In order to create a Sample Distribution of Sample Means, we need to first write a function that can sample *with* replacement.

In the cell below, write a function that takes in an array of numbers `data` and a sample size `n` and returns an array that is a random sample of `data`, of size `n`.

In [5]:

```
def get_sample(data, n):
    pass

test_sample = get_sample(data, 30)
print(test_sample[:5])
# [56, 12, 73, 24, 8] (This will change if you run it multiple times)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-5-d4c4bcd752fb> in <module>
      2     pass
      3
----> 4 test_sample = get_sample(data, 30)
      5 print(test_sample[:5])
      6 # [56, 12, 73, 24, 8] (This will change if you run it multiple times)

NameError: name 'data' is not defined
```

Generating a Sample Mean

Next, we'll write another helper function that takes in a sample and returns the mean of that sample.

In [6]:

```
def get_sample_mean(sample):
    pass

test_sample2 = get_sample(data, 30)
test_sample2_mean = get_sample_mean(test_sample2)
print(test_sample2_mean)
# 45.3 (This will also change if you run it multiple times)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-6-5ca84f572f3f> in <module>
      2     pass
      3
----> 4 test_sample2 = get_sample(data, 30)
      5 test_sample2_mean = get_sample_mean(test_sample2)
      6 print(test_sample2_mean)

NameError: name 'data' is not defined
```

Creating a Sample Distribution of Sample Means

Now that we have helper functions to help us sample with replacement and calculate sample means, we just need to bring it all together and write a function that creates a sample distribution of sample means!

In the cell below, write a function that takes in 3 arguments: the dataset, the size of the distribution to create, and the size of each individual sample. The function should return a sample distribution of sample means of the given size.

individual sample. The function should return a sample distribution of sample means of the given size.

In [7]:

```
def create_sample_distribution(data, dist_size=100, n=30):
    pass

test_sample_dist = create_sample_distribution(data)
print(test_sample_dist[:5])
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-7-a28f7d35cf69> in <module>
      2     pass
      3
----> 4 test_sample_dist = create_sample_distribution(data)
      5 print(test_sample_dist[:5])

NameError: name 'data' is not defined
```

Visualizing the Sample Distribution as it Becomes Normal

The sample distribution of sample means isn't guaranteed to be normal after it hits a magic size. Instead, the distribution begins to approximate a normal distribution as it gets larger and larger. Generally, 30 is accepted as the sample size where the Central Limit Theorem begins to kick in--however, there are no magic numbers when it comes to probability. On average, and only on average, a sample distribution of sample means where the individual sample sizes were 29 would only be slightly less normal, while one with sample sizes of 31 would likely only be slightly more normal.

Let's create some sample distributions of different sizes and watch the Central Limit Theorem kick in. As the sample size increases, you'll see the distributions begin to approximate a normal distribution more closely.

In the cell below, create a sample distribution from `data` of `dist_size` 10, with a sample size `n` of 3. Then, visualize this sample distribution with `distplot`.

In [8]:

```
# Your code here
```

Now, let's increase the `dist_size` to 30, and `n` to 10. Create another visualization to compare how it changes as size increases.

In [9]:

```
# Your code here
```

The data is already looking much more 'normal' than the first sample distribution, and much more 'normal' that the raw non-normal distribution we're sampling from.

In the cell below, create another sample distribution of `data` with `dist_size` 1000 and `n` of 30. Visualize it to confirm the normality of this new distribution.

In [10]:

```
# Your code here
```

Great! As you can see, the dataset *approximates* a normal distribution. It isn't pretty, but it's generally normal enough that we can use it to answer statistical questions using z-scores and p-values.

Another handy feature of the Central Limit Theorem is that the mean and standard deviation of the sample distribution should also approximate the population mean and standard deviation from the original non-normal dataset! Although it's outside the scope of this lab, we could also use the same sampling methods seen here to approximate other parameters from any non-normal distribution, such as the median or mode!

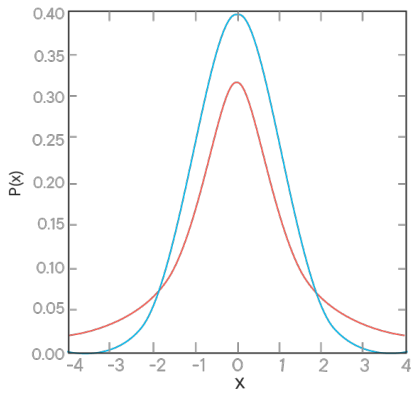
Summary

In this lab, we learned to apply the central limit theorem in practice. We learned how to determine if a dataset is normally distributed or not. From there, we used a function to sample with replacement and generate sample means. Afterwards, we created a normal distribution of sample means in order to answer questions about non-normally distributed datasets.

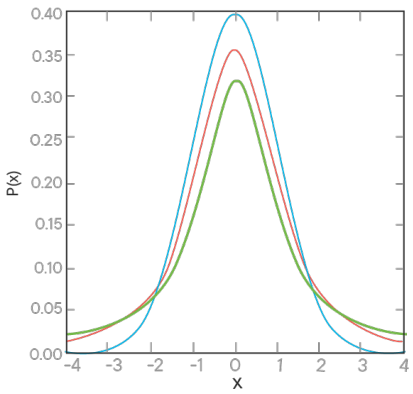
Confidence Intervals with T-Distribution

Introduction

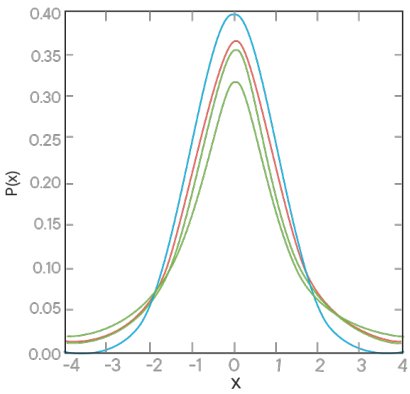
We've started to take a look at confidence intervals, but our example doesn't match what typically happens in practice. That is, when we previously calculated confidence intervals, we assumed we knew the population standard deviation. This is extremely rare, after all, when do you know the population standard deviation but not the population mean? To solve this problem, we use what's known as a t-distribution. T-distributions are similar to the normal distribution in shape but have heavier tails. T-distributions also have a parameter known as **degrees of freedom**. The higher the degrees of freedom, the closer the distribution resembles that of the normal distribution. Here the normal distribution is pictured in blue with the current t-distribution in red and previous t-distributions (with lower degrees of freedom) in green.



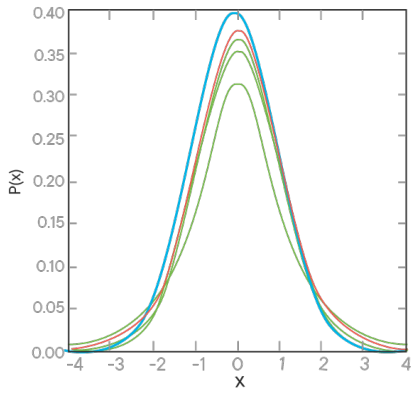
1 degree of freedom



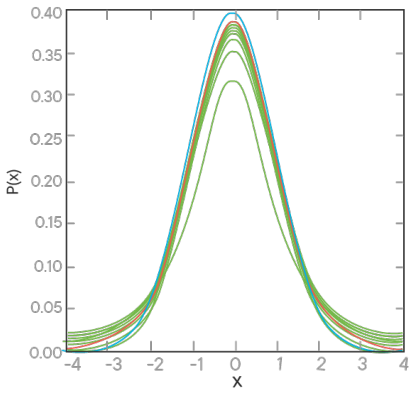
2 degree of freedom



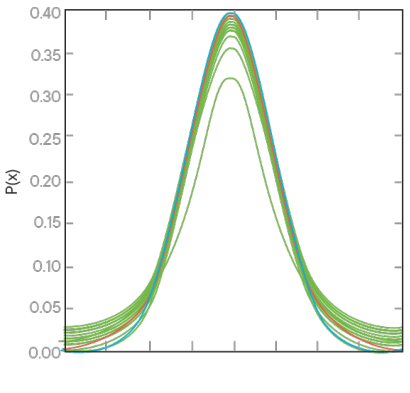
3 degree of freedom



5 degree of freedom



10 degree of freedom



30 degree of freedom

Objectives

You will be able to:

- Calculate confidence intervals
- Interpret confidence intervals in relation to true population parameters

Let's get started!

As stated above, we are often trying to infer population parameters from a sample. As such, we typically don't know the population variance or standard deviation. To start, it is thus natural to use the standard deviation of our sample as an estimate for the standard deviation of our population.

$$S = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})^2}$$

So, when we go to find our confidence interval as before, our equation,

$$\frac{\bar{X} - \mu}{\sigma/\sqrt{n}} \sim N(0, 1)$$

will become

$$\frac{\bar{X} - \mu}{S/\sqrt{n}}$$

(substituting S, the sample standard deviation, in for σ)

As a result, our question now becomes how is this quantity distributed?

$$T = \frac{\bar{X} - \mu}{S/\sqrt{n}}$$

While outside the scope of this discussion, it can be shown that this quantity can be modeled by a t-distribution. (Hence the use of the letter T).

With this, you can calculate confidence intervals for the population mean with your sample alone by using the equation:

$$\bar{x} \pm t_{\alpha/2, n-1}\left(\frac{S}{\sqrt{n}}\right)$$

To review some vocabulary and terms:

(1) $\bar{x}$ is a "point estimate" of μ

(2) $\bar{x} \pm t_{\alpha/2, n-1}\left(\frac{S}{\sqrt{n}}\right)$ is an "interval estimate" of μ

(3) $\frac{S}{\sqrt{n}}$ is the "standard error of the mean"

(4) $t_{\alpha/2, n-1}\left(\frac{S}{\sqrt{n}}\right)$ is the "margin of error"

Let's try this out in practice.

First, we start with a sample of patient cholesterol levels that we'll use to calculate the population mean with a 95% confidence.

In [1]:

```
import numpy as np
```

In [2]:

```
sample_chol_levels = [66.0, 36.0, 73.0, 48.0, 81.0, 69.0, 75.0, 81.0, 73.0,
                      69.0, 101.0, 70.0, 50.0, 42.0, 36.0, 71.0, 65.0, 43.0, 76.0, 24.0]
```

From our sample, we then calculate our sample mean ($\bar{x}$) and our sample standard deviation (S).

We pass the parameter `ddof = 1` to `np.std` to make sure we correctly compute the standard deviation of the sample.

In [3]:

```
x_bar = np.mean(sample_chol_levels)
s = np.std(sample_chol_levels, ddof = 1)
print(x_bar, s)
```

```
62.45 19.209304214912432
```

We then calculate our interval estimate using a t-distribution and our various parameters. The t-distribution requires 4 parameters:

- The sample mean
- The sample standard deviation
- The degrees of freedom (this is 1 less then the number of items in the sample)
- The confidence level we wish to have in our estimate

In [4]:

```
import scipy.stats as stats
```

In [5]:

```
stats.t.interval(alpha = 0.95,                              # Confidence level
                 df= len(sample_chol_levels)-1,             # Degrees of freedom
                 loc = x_bar,                               # Sample mean
                 scale = s)                                 # Standard deviation estimate
```

Out[5]:

```
(22.244464209742247, 102.65553579025776)
```

Note that this confidence interval is particularly wide! In order to achieve a 95% confidence level, we had to make a very general statement that we believe the average cholesterol level is between 22 and 102.

As a preview of running further simulations to investigate some of these relationships, here is a similar dataset, generated at random, and the associated statistical techniques used to estimate the population mean. Note that with the large sample size, the sample point estimates are fairly accurate on their own. Despite this, the confidence interval is still quite large for the population mean. In part, this is due to a large standard deviation.

Note we pass the parameter `ddof = 1` to `np.std` to make sure we correctly compute the standard deviation of the sample.

In [6]:

```
sample_chol_levels = np.random.normal(loc=54, scale=17, size=1000)
```

In [7]:

```
x_bar = np.mean(sample_chol_levels)
s = np.std(sample_chol_levels, ddof = 1)
print('Sample mean:', x_bar)
print('Sample standard deviation:', s)
```

```
Sample mean: 54.124158710009254
Sample standard deviation: 17.08507981915454
```

In [8]:

```
#Min and Max of Confidence Interval
stats.t.interval(alpha = 0.95,                              # Confidence level
                 df= len(sample_chol_levels)-1,             # Degrees of freedom
                 loc = x_bar,                               # Sample mean
                 scale = s)
```

Out[8]:

```
(20.59739821410794, 87.65091920591057)
```

Additional Resources

https://onlinecourses.science.psu.edu/stat414/node/199/

Summary

In this lecture, we investigated the more common method for calculating confidence intervals, as we will rarely know the population's standard deviation. As a result, we use the t-distribution, allowing us to find estimates for the population mean even when not knowing any specific parameters concerning the population.