

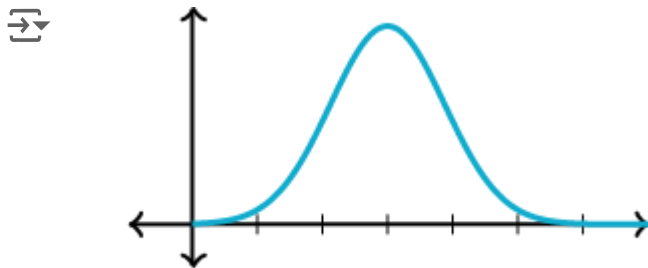
## ✓ Introduction

What is the core concept of the statistics that enable us to do predictive model, yet often confuses the data scientists?

Yes, that's the **Central Limit Theorem**.

Well, CLT is the critical component of the data science life cycle, and a main core of the hypothesis testing. In actually scenario, it's just a notion to understand, and most floundering question during an interview.

```
from IPython.display import Image  
Image("../input/imagedistribution/distribution.png")
```



We will understand the concept of Central Limit Theorem and will see why it is important and when to use it.

First, we will understand the CLT and its applicability and followed by an implementation on Heart Disease UCI dataset.

## ✓ Table of Contents

### 1. What is the Central Limit Theorem (CLT)?

## 2. Significance of CLT

### 1. Statistical Significance

### 2. Practical Applications

## 3. Assumptions Behind the Central Limit Theorem

## 4. Implementing the Central Limit Theorem in Python

## What is the Central Limit Theorem (CLT)?

The Central Limit Theorem (CLT) is a statistical theory states that given a sufficiently large sample size from a population with a finite level of variance, the mean of all samples from the same population will be approximately equal to the mean of the population.

Ohh, wait!. Are you overwhelm with technical terms? let's take an example for a naive understanding.

Consider that we have a section of football team of 30 members in tournament. Now, the question is what willbe the average weight of the team members? seems easy, ahh! right?.

An amature or normal **apporach** will be take weight of all members and take division of it by the total number of members in the team.

Well, if I say calculate the average weight of all the team in the tournament or, if we increase the discourse, all the individuals from each country who qualified for national team, let's say **SET**. Since, expansion in number of person to measure weight is humongous our approach will be changing, measuring the weight of all the individuals will be a very tiresome and long process.

## ✓ Different Approach

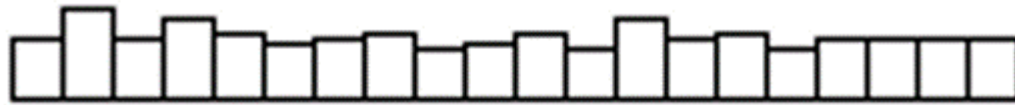
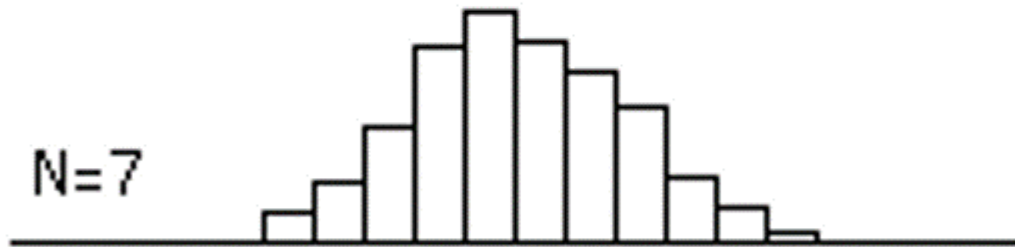
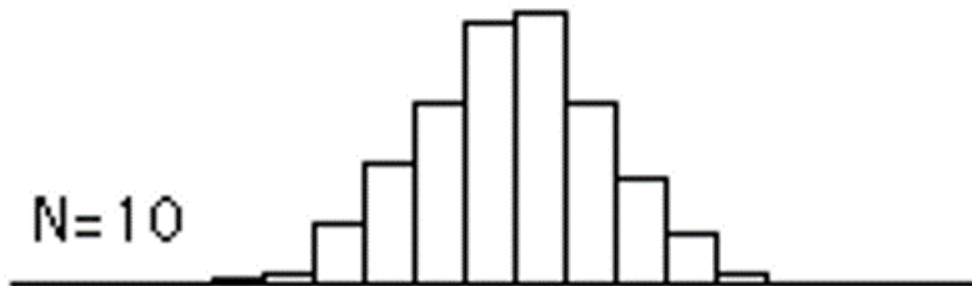
- First, draw groups of individuals at random from the SET. We will call this as a sample. We'll draw multiple samples, each consisting of 30 individuals.
- Calculate the individual mean of these samples
- Calculate the mean of these sample means
- Resultant will give the approximate mean weight of the individuals in the SET
- Additionally, the histogram of the sample mean weights of individuals will resemble a bell curve (or normal distribution)

## ✓ Formally defining CLT

Given a dataset with unknown distribution (it could be uniform, binomial or completely random), the sample means will approximate the normal distribution.

These samples should be large in size, applicable for random sampling, according to random probability for sampling. The distribution of sample means, calculated from repeated sampling, will tend to normality as the size of your samples gets larger.

```
Image("../input/imagedistribution/sampling.png")
```

 $N=1$  $N=4$  $N=7$  $N=10$ 

When  $n$  increases:

1. the distributions becomes more and more normal.
2. the spread of the distributions decreases.

## Significance of CLT

The CLT has significance for both statistical as well as practical applications.

### Statistical Significance

- Analyzing data involves statistical methods like hypothesis testing and constructing confidence intervals. These methods assume that the population is normally distributed. In the case of unknown or non-normal distributions, we treat the sampling distribution as normal according to the central limit theorem
- If we increase the samples drawn from the population, the standard deviation of sample means will decrease. This helps us estimate the population mean much more accurately
- Also, the sample mean can be used to create the range of values known as a confidence interval (that is likely to consist of the population mean)

### Practical Application

As an example:

- Political/election polls are prime CLT applications. These polls estimate the percentage of people who support a particular candidate. You might have seen these results on news channels that come with confidence intervals. The central limit theorem helps calculate that
- Confidence interval, an application of CLT, is used to calculate the mean family income for a particular region

## ✓ Assumptions Behind the Central Limit Theorem

To use the normal model, we must meet some assumptions and conditions. The Central Limit Theorem assumes the following:

1. Randomization Condition: The data must be sampled randomly. Is one of the good sampling methodologies discussed in the chapter "Sampling and Data" being used?
2. Independence Assumption: The sample values must be independent of each other. This means that the occurrence of one event has no influence on the next event. Usually, if we know that people or items were selected randomly we can assume that the independence assumption is met.

3. 10% Condition: When the sample is drawn without replacement (usually the case), the sample size,  $n$ , should be no more than 10% of the population.
4. Sample Size Assumption: Now, how we will figure out how large this size should be? Well, it depends on the population. When the population is skewed or asymmetric, the sample size should be large. If the population is symmetric, then we can draw small samples as well

In general, a sample size of 30 is considered sufficient when the population is symmetric.

The mean of the sample means is denoted as:

$$\mu_{\bar{X}} = \mu$$

where,

$\mu_{\bar{X}}$  = Mean of the sample means

$\mu$  = Population mean

And, the standard deviation of the sample mean is denoted as:

$$\sigma_{\bar{X}} = \sigma / \sqrt{n}$$

where,

$\sigma_{\bar{X}}$  = Standard deviation of the sample mean

$\sigma$  = Population standard deviation

$n$  = sample size

## ✓ Implementing the Central Limit Theorem in Python

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load in
```

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt

# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list the files in the input directory

import os
print(os.listdir("../input"))

# Any results you write to the current directory are saved as output.
```

→ ['imagedistribution', 'heart-disease-uci']

```
# We will check the what are the columns present and what datatypes are used and if there is any missing data.
df = pd.read_csv("../input/heart-disease-uci/heart.csv")
df.info()
```

→ <class 'pandas.core.frame.DataFrame'>  
RangeIndex: 303 entries, 0 to 302  
Data columns (total 14 columns):  
age            303 non-null int64  
sex           303 non-null int64  
cp            303 non-null int64  
trestbps      303 non-null int64  
chol          303 non-null int64  
fbs           303 non-null int64  
restecg       303 non-null int64  
thalach       303 non-null int64  
exang         303 non-null int64  
oldpeak       303 non-null float64  
slope         303 non-null int64  
ca            303 non-null int64  
thal          303 non-null int64  
target        303 non-null int64  
dtypes: float64(1), int64(13)

memory usage: 33.2 KB

As, we could see all the columns have equal numbers of row data, hence, there is no missing data.

For illustration of Central Limit Theorem for sampling we will take "chol" cholesterol column.

We are assuming the column chol is normally distributed. Normally distributed means upon giving graphical representation produces a bell shaped distribution.

```
df_chol = df['chol']  
# Looking the statisc of cholesterol  
df_chol.describe()
```

```
↗ count    303.000000  
mean      246.264026  
std        51.830751  
min        126.000000  
25%        211.000000  
50%        240.000000  
75%        274.500000  
max        564.000000  
Name: chol, dtype: float64
```

```
# let's take out the mean of the chol data  
df_chol.mean()
```


```
↗ 246.26402640264027
```

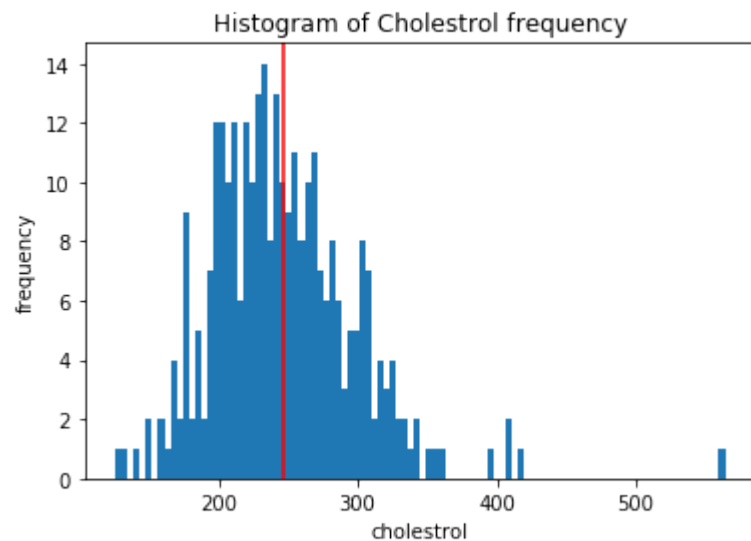
```
# plot all the observation in chol data  
plt.hist(df_chol, bins=100)
```

```
plt.xlabel('cholesterol')  
plt.ylabel('frequency')
```



```
plt.title('Histogram of Cholestrol frequency')
plt.axvline(x=df_chol.mean(),color='r')
```

 <matplotlib.lines.Line2D at 0x7fada1c1cdd8>



Interpretation:

1. We can see the vertical red line, mean of data, almost at the centre of main distribution.
2. Most of the distribution is in normal but not 100%.
3. Here, the data point after the 500 on x-axis is an outlier, and the points around 400 maybe or maynot be an outlier since they are very close to out main distribution graph.

We can also see from the above plot that the population is not normal, right? Therefore, we need to draw sufficient samples of different sizes and compute their means (known as sample means). We will then plot those sample means to get a normal distribution.

In our example,will draw 30 sample size (10% of total population), calculate thier mean, and plot them in python.

✓ here beolw upon every execution of code, will get a diffrent shap since drawing of sample is random.

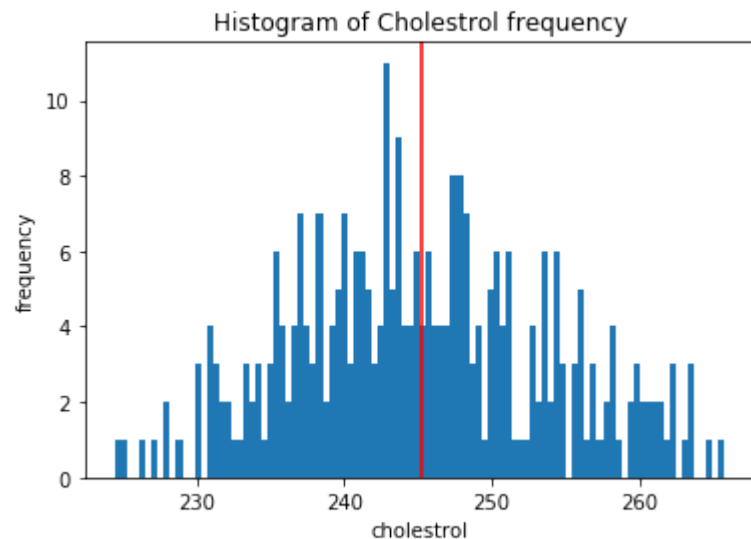
```
#We will take sample size=10, samples=300
#Calculate the arithmetice mean and plot the mean of sample 300 times
```

```
array = []
n = 300
for i in range(1,n):
    array.append(df_chol.sample(n=30,replace= True).mean())
```

```
#print(array)
plt.hist(array, bins=100)
```

```
plt.xlabel('cholesterol')
plt.ylabel('frequency')
plt.title('Histogram of Cholestrol frequency')
plt.axvline(x=np.mean(array),color='r') # for giving mean line
```

↗ <matplotlib.lines.Line2D at 0x7fada81854a8>



```
#We will take sample size=20, 60 & 500 samples=300
#Calculate the arithmetic mean and plot the mean of sample 300 times

array1 = []
array2 = []
array3 = []
n = 300
for i in range(1,n):
    array1.append(df_chol.sample(n=20,replace= True).mean())
    array2.append(df_chol.sample(n=60,replace= True).mean())
    array3.append(df_chol.sample(n=500,replace= True).mean())

#print(array)
fig , (ax1,ax2,ax3) = plt.subplots(nrows=1, ncols=3,figsize=(18,5))
#plt.figure()

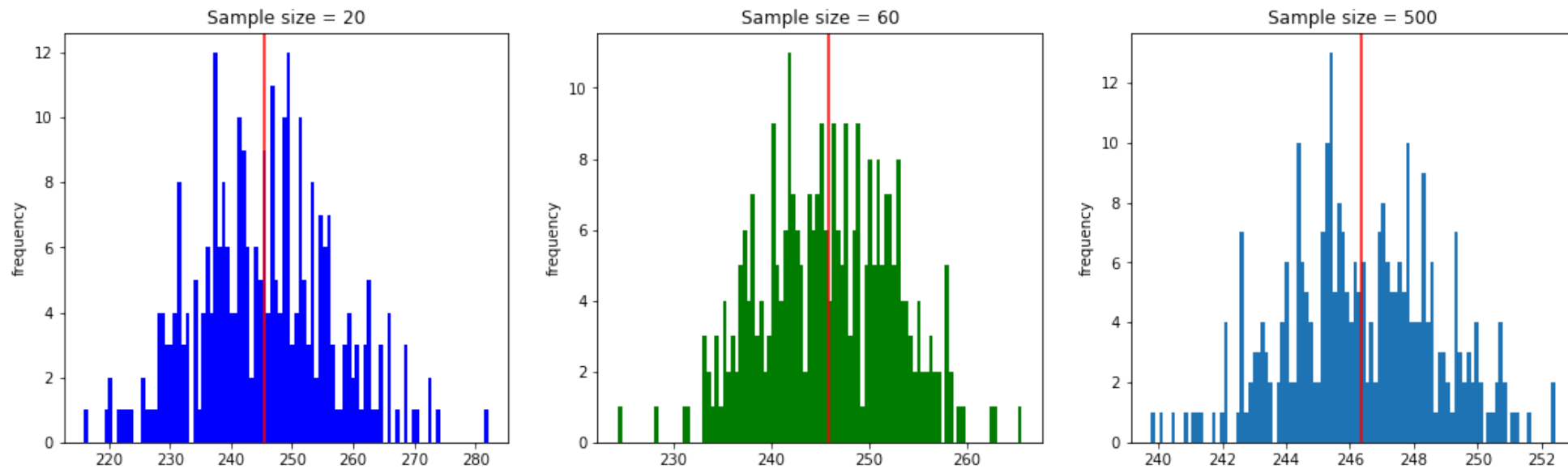
#plt.subplot(311)
ax1.hist(array1, bins=100,color='b')
ax1.set_xlabel('cholesterol')
ax1.set_ylabel('frequency')
ax1.set_title('Sample size = 20')
ax1.axvline(x=np.mean(array1),color='r') # for giving mean line

#ax2.subplot(312)
ax2.hist(array2, bins=100, color='g')
ax2.set_xlabel('cholesterol')
ax2.set_ylabel('frequency')
ax2.set_title('Sample size = 60')
ax2.axvline(x=np.mean(array2),color='r') # for giving mean line

#ax3.subplot(313)
ax3.hist(array3, bins=100)
ax3.set_xlabel('cholesterol')
```

```
ax3.set_ylabel('frequency')
ax3.set_title('Sample size = 500')
ax3.axvline(x=np.mean(array3),color='r') # for giving mean line
```

```
↪ <matplotlib.lines.Line2D at 0x7fada18d9588>
```



We can still see some of the outliers are there, so we need to remove those and after handling outlier we can conclude on the following:

Here, we get a good bell-shaped curve and the sampling distribution approaches normal distribution as the sample sizes increase. Therefore, we can consider the sampling distributions as normal and can use these distributions for further analysis.