Machine Learning is an interdisciplinary field that uses statistics, probability, algorithms to learn from data and provide insights which can be used to build intelligent applications. In this python notebook, we will discuss some of the key concepts widely used in machine learning.

What is statistics?

Statistics is the discipline that concerns the collection, organization, analysis, interpretation, and presentation of data. There are 2 types of statistics

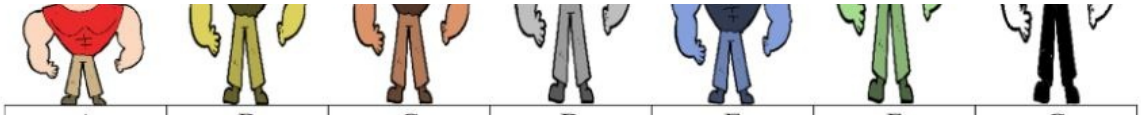
1) Descriptive statistics: Descriptive statistics is understanding, analyzing, summarizing the data in form of numbers and graphs. We analyze the data using different plots and charts on different kinds of data(numerical and categorical) like bar plot, pie chart, scatter plot, Histogram, etc. All the kind of interpretation, visualizing is part of descriptive statistics. Remember that descriptive statistics can be performed on a sample as well as population data but never do we get or take population data.

2) Inferential statistics: We are extracting some sample of data from population data, and from that sample of data, we are inferencing something(driving conclusion) for population data. It means we perform some tests on sample data and make a conclusion specific to that population. we use various techniques to drive conclusions including data visualization, manipulation, etc.

Mean,Mode and Median:

To represent a dataset as a 1-number summary, we use central tendency measure. There exist three central tendency measures i.e. Mean, Median and Mode.**Mean,Mode and Median** are different measures of center in a numerical data set. They each try to summarize a dataset with a single number to represent a "typical" data point from the dataset.

Let’s begin with the visual representations to better interpret the concepts:
*Dataset used — Heights of seven Bodybuilders(Assumed Discrete Series)



A	B	C	D	E	F	G
150 cm	160 cm	160 cm	170 cm	155 cm	180 cm	175 cm

Names as shown in table – A, B, C, D, E, F, G
Heights (in cm) – 150, 160, 160, 170,155,180,175

Now we will calculate the central tendency of this data using Mean, Median, and Mode.
After the calculations, we will identify how each of these central tendency measures behaves when a new data-point is added to the data which will further enable us to understand the importance of each central tendency measure and application suitability for different conditions.

Mean:

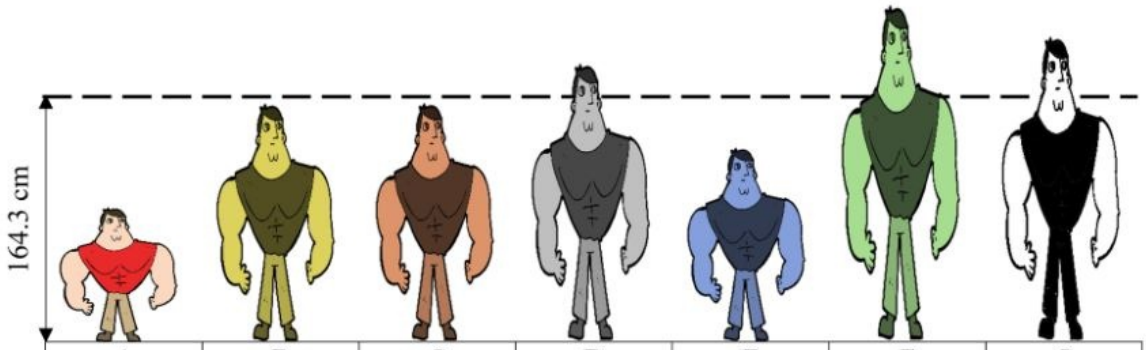
1. Mean: Let's start with the mean. Mean is defined by the "average" number found by adding all data points divided by the total number of datapoints.

Calculating the Mean: The arithmetic mean is sum of all data points divided by the total number of data points

$$\text{mean} = \frac{\text{sum of data}}{\#\text{ of data points}}$$

Here's the same formula written more formally:

$$\text{mean} = \frac{\sum x_i}{n}$$



A	B	C	D	E	F	G
150 cm	160 cm	160 cm	170 cm	155 cm	180 cm	175 cm

$$\textbf{Mean} = \frac{(150+160+160+170+155+180+175)}{7} = 164.3 \text{ cm}$$

Mathematically solved: Let's first understand the underlying math for how to calculate the mean.

Question: Find the mean of this data: 150,160,160,170,155,180,175.

Answer: First we have to add all the numbers:

```
= 150 + 160 + 160 + 170 + 155 + 180 + 175
= 1150
Sum of data = 1150
```

```
Total number of datapoints = 7

Mean=1150/7=164.28571428

The mean of the data points is 164.28571428
```

Implementation in python: Here we will see how to find mean using inbuilt function in numpy.

In [1]:
```
import numpy as np
import pandas as pd

data=np.array([150,160,160,170,155,180,175])

print("The mean of the data point is ",data.mean())
```

```
The mean of the data point is  164.28571428571428
```

In [2]:
```
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings('ignore')
'''downlaod iris.csv from https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/data/iris.csv'''
#Load Iris.csv into a pandas dataFrame.
iris = pd.read_csv("https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/data/iris.csv")
iris.mean()
```

Out[2]:
```
sepal_length    5.843333
sepal_width     3.054000
petal_length    3.758667
petal_width     1.198667
dtype: float64
```

Naive implementation in python: Let's see the naive implementation of mean concept in python.

In [3]:
```
def naive_mean(data):
    '''This function will calculate the mean of data'''
    total_no_of_datapoints=0
    sum_of_data=0
    for element in data:
        total_no_of_datapoints=total_no_of_datapoints+1
        sum_of_data=sum_of_data+element

    mean=sum_of_data/total_no_of_datapoints
    return mean


data=[150,160,160,170,155,180,175]
print("The mean of the data point is ",naive_mean(data))
```

```
The mean of the data point is  164.28571428571428
```

In [4]:
```
import numpy as np
import pandas as pd
'''downlaod iris.csv from https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/data/iris.csv'''
#Load Iris.csv into a pandas dataFrame.
iris = pd.read_csv("https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/data/iris.csv")

def naive_mean(dataset):
    '''This function will calculate the mean of data'''
    for column in dataset:
        if(dataset[column].dtypes=='float64'):
            col_sum=0
            col_count=0
            for value in dataset[column]:
                col_sum=col_sum+value
                col_count=col_count+1
            print("{}   {}".format(column,col_sum/col_count))

naive_mean(iris)
```

```
sepal_length   5.843333333333335
sepal_width   3.0540000000000007
petal_length   3.7586666666666693
petal_width   1.1986666666666672
```

Note: Mean is the mostly use common measures of central tendency but it has a huge downside because it is easily affected by outliers

which value is significantly greater than other values in the dataset.

Let us see an example of mean with an outlier datapoint:

Mathematically solved:

Question: Find the mean of this data: 150,160,160,170,155,180,175,500

Answer: First we have to add all the numbers:

```
150 + 160 + 160 + 170 + 155 + 180 + 175 + 500
Sum of data = 1650
Total number of datapoints = 8
Mean=1650/8

The mean of the data points is 206.25
```

Implementation in python:

In [5]:
```
import numpy as np
import pandas as pd

data=np.array([150,160,160,170,155,180,175,500])

print("The mean of the data point is ",data.mean())
```

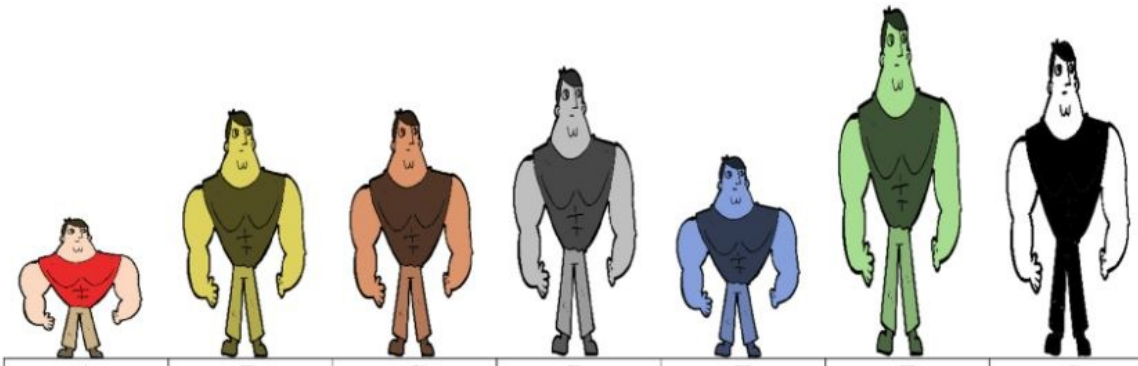
```
The mean of the data point is  206.25
```

Explanation: As you can see because of one outlier value 500 your mean get corrupted by a high margin from 164.28 to 206.25. So, mean is outlier prone central tendency measure.

Mode:

Mode: This is the easiest one to calculate, just determine the frequency of occurrence of each data point in the data, and the one with the highest frequency is the mode of the data. This measure can also be used when the data is non-numerical.

Calculating the mode: The mode is the most commonly occuring data points in a dataset. The mode is useful when there are lots of repeated values in a dataset. There can be no mode, one mode or multiple modes.



A	B	C	D	E	F	G
150 cm	160 cm	160 cm	170 cm	155 cm	180 cm	175 cm

Height	150 cm	**160 cm**	170 cm	155 cm	180 cm	175 cm
Count	1	**2**	1	1	1	1

Mathematically solved: Let's first understand the underlying math for how to calculate the mode.

Question: Find the mode of this data: 150,160,160,170,155,180,175

Answer: Let's find the occurence of number in a dataset.

```
150:1
160:2
```

```
170:1
155:1
180:1
175:1

As we can see the 160 is occuring the highest times in the given dataset.
The mode of the datapoint is 160
```

Implementation in python: Here we will see how to find the mode using inbuilt function in statistics

In [6]:
```
import numpy as np
import statistics

data=np.array([150,160,160,170,155,180,175])

print("The mode of the data point is ",statistics.mode(data))
```

```
The mode of the data point is  160
```

Naive implementation in python: Let's see the naive implementation of mode in python.

In [7]:
```
def naive_mode(data):
    '''This function find the mode of the data'''
    occurence_dict={}
    for element in data:
        if element not in occurence_dict:
            occurence_dict[element]=1
        else:
            occurence_dict[element]=occurence_dict[element]+1
    mode=sorted(occurence_dict.items(), key=lambda v:v[1], reverse=True)[0][0]
    return mode

data=[150,160,160,170,155,180,175]
print("The mode of the data point is ",naive_mode(data))
```

```
The mode of the data point is  160
```

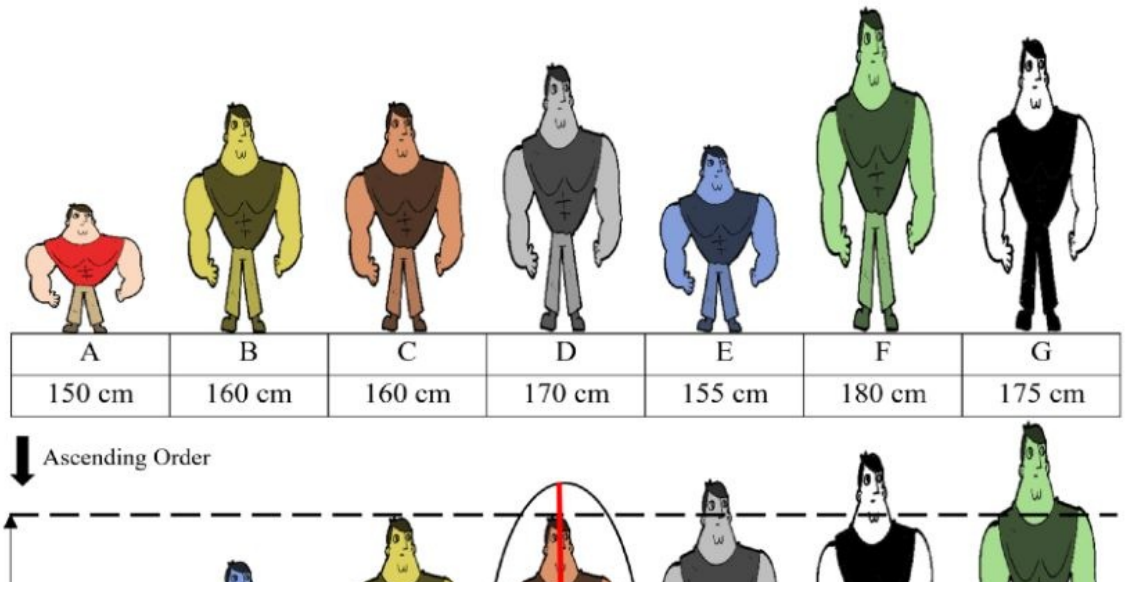
Median:

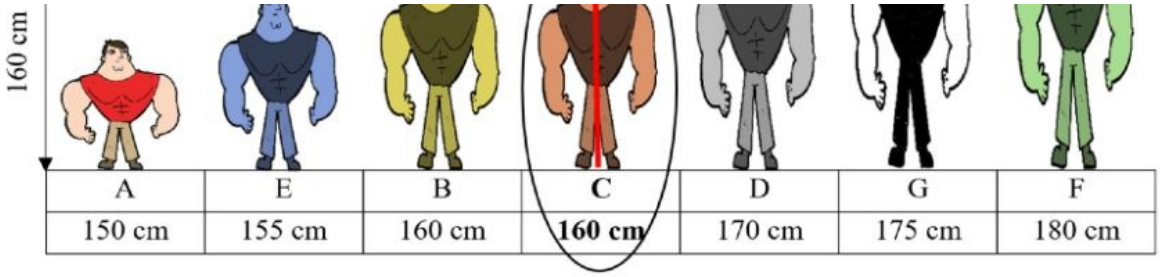
Median: The middle number found by ordering all the data points and picking out the one in the middle (or if there are two middle numbers, taking the mean of those two numbers).

Finding the median: The median is the middle point in a dataset - half of the datapoints are smaller than the median and half of the data points are larger.

To calculate the median:

1. Arrange the data points from smallest to largest.
2. If the number of data points is odd, the median is the middle data points in the list.
3. If the number of data point is even , then the median is the average of the two middle data points in the list





Mathematically solved(Odd number of points) :Let's first see how to calculate the median of dataset.

Question: Find the median of this data: 150,160,160,170,155,180,175

Answer: Step1: Sort the data in ascending order first: 150,155,160,160,170,175,180

Step2: There is an odd number in data points. so, the median is 160 the middle data point.

The median of the datapoint is 160.

Implementation in python:

In [8]:
```
import numpy as np
import statistics

data=np.array([150,160,160,170,155,180,175])

print("The median of the data point is ",statistics.median(data))
```

```
The median of the data point is  160
```

Mathematically solved:(Even number of points)

Question: Find the median of this data: 150,160,160,170,190,155,180,175 Answer: Step1: Sort the data in ascending order first: 150,155,160,160,170,175,180,190 Step2: There is an even number in data points. so, the median is mean of 160 and 170 which is 165. The median of the datapoint is 165.

Implementation in python:

In [9]:
```
import numpy as np
import statistics

data=np.array([150,160,160,170,190,155,180,175])

print("The median of the data point is ",statistics.median(data))
```

```
The median of the data point is  165.0
```

In [10]:
```
import pandas as pd
'''downlaod iris.csv from https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/data/iris.csv'''
#Load Iris.csv into a pandas dataFrame.
iris = pd.read_csv("https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/data/iris.csv")

iris.median()
```

Out[10]:
```
sepal_length    5.80
sepal_width     3.00
petal_length    4.35
petal_width     1.30
dtype: float64
```

Naive implementation in python: Let's see the naive implementation of median in python.

In [11]:
```
def naive_median(data):
    '''This function will calculate the median'''
    data.sort()

    if(len(data)%2==0):
        median=(data[(len(data)//2)-1]+data[(len(data)//2)])/2
    else:
        median=data[(len(data)//2)]
    return median
```

```
data=[150,160,160,170,155,180,175]
print("The median of the data point is ",naive_median(data))
data=[150,160,160,170,190,155,180,175]
print("The median of the data point is ",naive_median(data))
```

```
The median of the data point is  160
The median of the data point is  165.0
```

In [12]:
```
import pandas as pd
'''downlaod iris.csv from https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/data/iris.csv'''
#Load Iris.csv into a pandas dataFrame.
iris = pd.read_csv("https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/data/iris.csv")

def naive_median(dataset):
    '''This function will calculate the median'''
    for column in dataset:
        if(dataset[column].dtypes=='float64'):
            sorted_df=dataset[column].sort_values()
            sorted_df=sorted_df.reset_index(drop=True)
            if(len(sorted_df)%2==0):
                median=(sorted_df[(len(sorted_df)//2)-1]+sorted_df[(len(sorted_df)//2)])/2
            else:
                median=sorted_df[(len(sorted_df)//2)]
            print("{}   {}".format(column,median))
naive_median(iris)
```

```
sepal_length   5.8
sepal_width   3.0
petal_length   4.35
petal_width   1.3
```

Dispersion of data:

Dispersion of data used to understands the distribution of data. It helps to understand the variation of data and provides a piece of information about the distribution data. Variance, Standard Deviation,Range and IQR are the methods used to understand the distribution data.

Dispersion of data helps to identify outliers in a given dataset.

Measures of variability:

Following are some of the measures of variability that offers to differentiate between datasets:

- Variance
- Standard Deviation
- Range
- Interquartile range

1. Variance </h2

The variance is a numerical measure of how the data values is dispersed around the mean..
Mathematically, it is defined as the average of squared differences from the mean value.

To calculate the variance follows these steps:

1. Find out the mean
2. Then for each number subtract the mean and square the result
3. The worked out the average of those squared difference

Formulae:

Population Variance	Sample Variance
$\sum^{N}(x_i-\mu)^2$	$\sum^{n}(x_i-\bar{x})^2$

$\sigma^2 = \frac{\sum_{i=1}^{N}}{N}$	$s^2 = \frac{\sum_{i=1}^{n}}{n-1}$
σ^2 = population variance x_i = value of i^{th} element μ = population mean N = population size	s^2 = sample variance x_i = value of i^{th} element $\bar{x}$ = sample mean n = sample size

Mathematically solved: Let's see how to calculate the variance of data mathematically digest the mathematics logic first then we will jump into code.

Example: Find the variance of the numbers 3, 8, 6, 10, 12, 9, 11, 10, 12, 7.

Solution:

Given,

3, 8, 6, 10, 12, 9, 11, 10, 12, 7

Step 1: Compute the mean of the 10 values given.

Mean = (3+8+6+10+12+9+11+10+12+7) / 10 = 88 / 10 = 8.8

Step 2: Make a table with three columns, one for the X values, the second for the deviations and the third for squared deviations. As the data is not given as sample data so we use the formula for population variance. Thus, the mean is denoted by μ.

Value X	X − μ	$(X - \mu)^2$
3	-5.8	33.64
8	-0.8	0.64
6	-2.8	7.84
10	1.2	1.44
12	3.2	10.24
9	0.2	0.04
11	2.2	4.84
10	1.2	1.44
12	3.2	10.24
7	-1.8	3.24
Total	0	73.6

Step 3:

$\sigma^2 = \frac{\sum(X-\mu)^2}{N}$

= 73.6 / 10

= 7.36

Naive implementation in python: Let's see the naive implementation of variance in python.

In [13]:
```
def naive_mean(data):
    '''This function will calculate the mean of data'''
    total_no_of_datapoints=0
    sum_of_data=0
    for element in data:
        total_no_of_datapoints=total_no_of_datapoints+1
        sum_of_data=sum_of_data+element
```

```
    mean=sum_of_data/total_no_of_datapoints
    return mean,total_no_of_datapoints

def naive_variance(data):
    '''This function will calculate the variance of data'''
    mean,total_no_of_datapoints=naive_mean(data)
    variance_sum=0
    for element in data:
        variance_sum=variance_sum+(element-mean)**2
    variance=variance_sum/(total_no_of_datapoints-1)
    return variance


data=[3,8,6,10,12,9,11,10,12,7]
print("The variance of the data point is ",naive_variance(data))
print("The variance of sepal length is ",naive_variance(iris['sepal_length']))
print("The variance of sepal width is ",naive_variance(iris['sepal_width']))
print("The variance of petal length is ",naive_variance(iris['petal_length']))
print("The variance of petal width is ",naive_variance(iris['petal_width']))
```

```
The variance of the data point is  8.177777777777777
The variance of sepal length is  0.6856935123042505
The variance of sepal width is  0.18800402684563763
The variance of petal length is  3.1131794183445156
The variance of petal width is  0.5824143176733784
```

Implementation in python:

In [14]:
```
import numpy as np
import statistics

X=np.array([3,8,6,10,12,9,11,10,12,7])

print("The variance is: ",statistics.pvariance(X))
```

```
The variance is:  7
```

In [15]:
```
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')
'''downlaod iris.csv from https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/data/iris.csv'''
#Load Iris.csv into a pandas dataFrame.
iris = pd.read_csv("https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/data/iris.csv")
iris
```

Out[15]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

150 rows × 5 columns

In [16]:
```
statistics.variance(iris['sepal_length'])
```

Out[16]: 0.6856935123042506

In [17]:
```
statistics.variance(iris['sepal_width'])
```

Out[17]: 0.18800402684563758

Code Explanation: Here you can see that we have plot the distribution plot of "sepal length" and "sepal width" and calculate the variance for both as well. If you notice in the sepal length plot the shoulder of plot is wider than the shoulder of sepal width. It means more and more value is dispersed around the mean in sepal length then sepal width thats why the variance of sepal length is high than sepal width.

How to interpreat the variance value: Low value of variance indicate that the data are cluster together and not spread apart widely whereas a hight value would indicate that the data in the given set are much more spread apart from the average value.

2. Standard Deviation

- Standard Deviation is a measure of spread in statistics. It is used to quantify the measure of spread. Variation of a set of data values. It is very much similar to variance, gives the measure of deviation whereas variation provide the squared value.
- Standard Deviation measures the spreadness of data values with respect to mean and mathematically, is calculated as square root of variance.

Formulae:

Standard Deviation Formula	
Population	**Sample**
$\sigma = \sqrt{\frac{\sum(X-\mu)^2}{N}}$	$s = \sqrt{\frac{\sum(X-\bar{x})^2}{n-1}}$
X – The Value in the data distribution *μ – The population Mean* *N – Total Number of Observations*	*X – The Value in the data distribution* *$\bar{x}$ – The Sample Mean* *n - Total Number of Observations*

Mathematically solved:

Question: During a survey, 6 students were asked how many hours per day they study on an average? Their answers were as follows: 2, 6, 5, 3, 2, 3. Evaluate the standard deviation.

Solution:

Find the mean of the data:

$\frac{(2+6+5+3+2+3)}{6} = 3.5$

Step 2: Construct the table:

x_1	$x_1 - \bar{x}$	$(x_1 - \bar{x})^2$
2	-1.5	2.25
6	2.5	6.25
5	1.5	2.25
3	-0.5	0.25
2	-1.5	2.25
3	-0.5	0.25
= 13.5		

Step 3: Now, use the Standard Deviation formula

Sample Standard Deviation = $s = \sqrt{\frac{\sum(X-\bar{X})^2}{n-1}}$

=√(13.5/[6-1])

=√[2.7]

=1.643

Naive implementation in python: Let's see the naive implementation of variance in python.

In [18]:
```
import math

def naive_mean(data):
    '''This function will calculate the mean of data'''
    total_no_of_datapoints=0
    sum_of_data=0
    for element in data:
        total_no_of_datapoints=total_no_of_datapoints+1
        sum_of_data=sum_of_data+element

    mean=sum_of_data/total_no_of_datapoints
    return mean,total_no_of_datapoints

def naive_standard_deviation(data):
    '''This function will calculate the variance of data'''
    mean,total_no_of_datapoints=naive_mean(data)
    variance_sum=0
    for element in data:
        variance_sum=variance_sum+(element-mean)**2
    variance=math.sqrt(variance_sum/total_no_of_datapoints)
    return variance


data=[2,6,5,3,2,3]
print("The standard deviation of the data point is ",naive_standard_deviation(data))
print("The variance of sepal length is ",naive_standard_deviation(iris['sepal_length']))
print("The variance of sepal width is ",naive_standard_deviation(iris['sepal_width']))
print("The variance of petal length is ",naive_standard_deviation(iris['petal_length']))
print("The variance of petal width is ",naive_standard_deviation(iris['petal_width']))
```

```
The standard deviation of the data point is  1.5
The variance of sepal length is  0.8253012917851409
The variance of sepal width is  0.4321465800705435
The variance of petal length is  1.7585291834055201
The variance of petal width is  0.760612618588172
```

Implementation in python:

In [19]:
```
import numpy as np
import statistics

X=np.array([2,6,5,3,2,3])

print("The variance is: ",statistics.stdev(X))
```

```
The variance is:  1.4142135623730951
```

In [20]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
'''downlaod iris.csv from https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/data/iris.csv'''
#Load Iris.csv into a pandas dataFrame.
iris = pd.read_csv("https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/data/iris.csv")
iris
```

Out[20]:

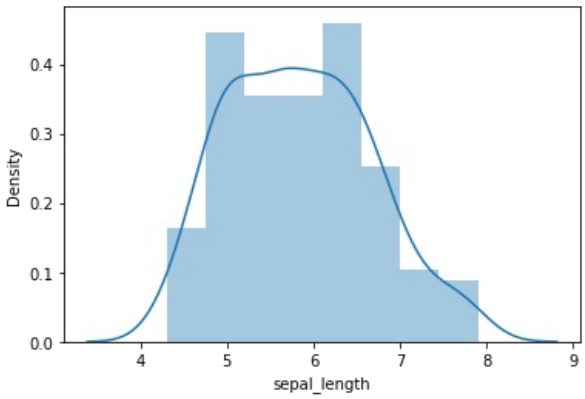
	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica

146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

150 rows × 5 columns

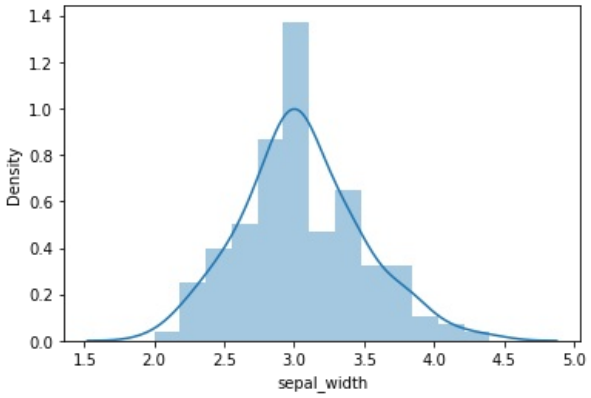
In [21]:
```
sns.distplot(iris['sepal_length'])
statistics.stdev(iris['sepal_length'])
```

Out[21]: 0.8280661279778629



In [22]:
```
sns.distplot(iris['sepal_width'])
statistics.stdev(iris['sepal_width'])
```

Out[22]: 0.43359431136217363



Real life example of using standard deviation:

Example 1: Standard Deviation in Weather Forecasting Standard deviation is widely used in weather forecasting to understand how much variation exists in daily and monthly temperatures in different cities.

For example:

A weatherman who works in a city with a small standard deviation in temperatures year-round can confidently predict what the weather will be on a given day since temperatures don't vary much from one day to the next.

A weatherman who works in a city with a high standard deviation in temperatures will be less confident in his predictions because there is much more variation in temperatures from one day to the next.

Example 2: Standard Deviation in Real Estate Standard deviation is a metric that is used often by real estate agents.

For example:

Real estate agents calculate the standard deviation of house prices in a particular area so they can inform their clients of the type of variation in house prices they can expect.

Real estate agents also calculate the standard deviation of the square footage of house prices in certain areas so they can inform their clients

on what type of variation to expect in terms of square footage of houses in a particular area.

Example 3: Standard Deviation in Human Resources Standard deviation is often used by individuals who work in Human Resource departments at companies.

For example:

Human Resource managers often calculate the standard deviation of salaries in a certain field so that they can know what type of variation in salaries to offer to new employees.

Example 4: Standard Deviation in Marketing Standard deviation is often used by marketers to gain an understanding of how their advertisements perform.

For example:

Marketers often calculate the standard deviation of revenue earned per advertisement so they can understand how much variation to expect in revenue from a given ad.

Marketers also calculate the standard deviation of the number of ads used by competitors to understand whether or not competitors are using more or less ads than normal during a given period.

Example 5: Standard Deviation in Test Scores Standard deviation is used by professors at universities to calculate the spread of test scores among students.

For example:

Professors can calculate the standard deviation of test scores on a final exam to better understand whether most students score close to the average or if there is a wide spread in test scores.

Professors can also calculate the standard deviation of test scores for multiple classes to understand which classes had the highest variation in test scores among students.

Important Interview Question: Let's discuss few important interview questions.

Qus: While calculating the variance why we square the difference of mean value why not we just take absolute value in variance formulae?

Ans:

1. Calculate the Mean :

$$\textit{Mean} = \frac{5+7+9+3}{4} = 6$$

2. To compute variance, first need to subtract the mean from all observations to find the distance of all values from the average (mean).

$$\textit{Mean Deviation} = \frac{(5-6)+(7-6)+(9-6)+(3-6)}{4} = \frac{(-1)+(1)+(3)+(-3)}{4} \Rightarrow 0$$

We noticed the result as zero because whenever we discussed the mean, the identity properties of a sample mean *"deviation of a means is always sum to zero. "*

Therefore we have to work on it. We don't want negative values (as distance cannot be negative). We need something that secures negative values into positive numbers. We can practice either **Mean absolute deviation** or **squared deviation (Standard deviation).**

Mean absolute deviation (MAD):
MAD is the measure of spread (variability).

$$\textbf{Mean Absolute Deviation} = \frac{\sum \text{Absolute values of Deviation from central measure}}{\text{Total Number of observations}}$$

$$\textbf{Mean Absolute Deviation} = \frac{|5-6| + |7-6| + |9-6| + |3-6|}{4}$$

$$= \frac{(1) + (1) + (3) + (3)}{4} \Rightarrow 2$$

Here | gives the absolute value that means all negative deviation (distance) made positive.

The problem with MAD is that it is not well behaved as squared deviation dose in a normal distribution, and it is not differential. It is most salutary to use squared deviation than MAD as squared deviation has well mathematical properties for the normal distribution.

Squared Difference

$$\textit{Variance} = \frac{(5-6)^2 + (7-6)^2 + (9-6)^2 + (3-6)^2}{4}$$ } Average Squared Deviation

$$= \frac{(1) + (1) + (9) + (9)}{4} \Rightarrow 5$$

Qus: When we already have the variance to calculate dispersion of data. So, why we need standard deviation?

Ans: Both standard deviation and variance gives the information about spread in data but the units of the variance are the original units squared. That's not always intuitive and sometimes downright confusing. Taking the square root to get the standard deviation ensures the units of spread are the same as the units for location (mean, mode, median, etc). This can make interpretation easier.

Qus: Why do we use N-1 in sample variance and N in population variance formulae?

Ans: https://youtu.be/sHRBg6BhKjI(go through this link)

3. Range:

The range can measure by subtracting the lowest value from the largest value. The wide range indicates high variability, and the small range specifies low variability in the distribution. To calculate a range, prepare all the values in ascending order, then subtract the lowest value from the highest value.
Intuition: The range will tell us how spread our data is.

$$\text{Range}(X) = \text{Max}(X) - \text{Min}(X)$$

Student_id	1	2	3	4	5
Marks	37	33	19	25	28

Range = H – L
= 37-19 ⟹ 18

The range of marks is 18.

Mathematically solved:

Question: Find the range of this data: 37,33,19,25,28

Answer: The lowest value in the dataset is 19 The highest value in the dataset is 37

```
Range=Max(x) - Min(x)
     =37-19

so, the range is 18
```

Naive implementation in python:

In [23]:
```
def naive_range(data):
    '''This function will calculate the range of data'''
    data.sort()
    min_value=data[0]
    max_value=data[-1]
    return max_value-min_value

data=[37,33,19,25,28]
print("So, the range is ",naive_range(data))
```

```
So, the range is  18
```

Implementation in python:

In [24]:
```
import numpy as np

x=np.array([37,33,19,25,28])

print("The range is: ",max(x)-min(x))
```

```
The range is:  18
```

The range can influence by outliers. If there is one extreme value that can change the value of a range.

Student_id	1	2	3	4	5
Marks	60	33	19	25	28

Range = H – L
= 60-19 ⟹ 41 Now range of marks is 41.

Implementation in python:

In [25]:
```
import numpy as np

x=np.array([60,33,19,25,28])

print("The range is: ",max(x)-min(x))
```

```
The range is:  41
```

Mid range:

The mid range of a set of statistical data values is the arithmetic mean of the maximum and minimum value in dataset

$$\text{Midrange} = \frac{\text{Max}(x) + \text{Min}(x)}{2}$$

Mathematically solved:

Question: Find the midrange of this data: 4,6,9,3,7

Answer: The lowest value in the dataset is 3 The highest value in the dataset is 9

```
Range=(Max(x) + Min(x))/2
     =(9+3)/2

so, the midrange is 6
```

Implementation in python:

In [26]:
```
import numpy as np

x=np.array([4,6,9,3,7])

print("The midrange is: ",(max(x)+min(x))/2)
```

```
The midrange is:  6.0
```

4. Interquartile Range(IQR):

- Before going to the IQR we have to understand the concept of **Percentile** and **Quartile**:

Percentile:

Percentile is a measure used in statistics indicating the values which a given percentage of observation in a group of observation falls

How to calculate percentile:

Let us suppose you have 100 datapoints in a random variable:

X={X1,X2,X3,X4,X5,X6,.....,X95,X96,X97,X98,X99,X100}

Step 1: Rank the values in the dataset in order from smallest to largest.
X={X1',X2',X3',X4',X5',X6',.....,X95',X96',X97',X98',X99',X100'}

Step 2: Multiply K(percent) by n(total number of values in the dataset). This is the index . You will refer to this in the next steps as the position of values in your dataset.

```
formulae to calculate percentile is = percentile/100 * (Total no of item+1)

0.01*100=1st percentile
0.02*100=2nd percentile
.
.
.
0.25*100=25th percentile
.
.
.
0.50*100=50th percentile
.
.
.
0.75*100=75th percentile
.
.
.
0.1*100=100th percentile
```

Step 3: If the answer is not a whole number then rounding the number is required. If it is a whole number, after rounding whatever value you

will get it is a percentile rank.

Practicle intuition of percentile

Let's understand the practically where to use percentile concept. Let's suppose you have appeared in a math exam and your teacher told you that you get 90th percentile read it carefully you got 90th percentile not 90%. So, to score 90th percentile of an exam does not mean, necessarily that you got 90% marks in exam. It means that 90% of the exam score are same or less than your score and 10% of the test score are same or greater than your test score.

Percentile are very useful for comparing values. For this reason university and colleges uses percentile extensively. One instance in which college and university used percentile in which JEE result are used to determine a minimum testing score that will be used as an acceptence factor. For example suppose IIT accept JEE score at or above the 85th percentile. This translate into a score of atleast 90 marks out of 100.

Percentile are mostly used with very large population. Therefore, if you were to say that 90% of the test score are less than your test score, it would be acceptable because removing one particular data value is not significant.

Naive implementation in python: In this code we will see how to calculate the percentile of a particular number in data.

Example: The scores obtained by 10 students are 38, 47, 49, 58, 60, 65, 70, 79, 80, 92. Using the percentile formula, calculate the percentile for score 70?

In [27]:
```
def calculate_percentile_score(data,num):
    '''This function will calculate the percentile score for a particular number'''
    data.sort()
    count=0
    for element in data:
        if(element<num):
            count=count+1
    return int((count/len(data))*100)

data=[38, 47, 49, 58, 60, 65, 70, 79, 80, 92]
num=70
print("Therefore, the percentile for value {} = {}th Percentile".format(num,calculate_percentile_score(data,num)))
```

```
Therefore, the percentile for value 70 = 60th Percentile
```

Implementation in python:

In [28]:
```
import numpy as np

data=range(0,101)

print("1st percentile of data is: ",np.percentile(data,1))
print("5th percentile of data is: ",np.percentile(data,5))
print("25th percentile of data is: ",np.percentile(data,25))
print("50th percentile of data is: ",np.percentile(data,50))
print("75th percentile of data is: ",np.percentile(data,75))
print("100th percentile of data is: ",np.percentile(data,100))
```

```
1st percentile of data is:  1.0
5th percentile of data is:  5.0
25th percentile of data is:  25.0
50th percentile of data is:  50.0
75th percentile of data is:  75.0
100th percentile of data is:  100.0
```

In [29]:
```
import numpy as np

data=range(0,501)

print("1st percentile of data is: ",np.percentile(data,1))
print("5th percentile of data is: ",np.percentile(data,5))
print("25th percentile of data is: ",np.percentile(data,25))
print("50th percentile of data is: ",np.percentile(data,50))
print("75th percentile of data is: ",np.percentile(data,75))
print("100th percentile of data is: ",np.percentile(data,100))
```

```
1st percentile of data is:  5.0
5th percentile of data is:  25.0
25th percentile of data is:  125.0
50th percentile of data is:  250.0
75th percentile of data is:  375.0
100th percentile of data is:  500.0
```

In [30]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import warnings
warnings.filterwarnings('ignore')
'''downlaod iris.csv from https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/data/iris.csv'''
#Load Iris.csv into a pandas dataFrame.
iris = pd.read_csv("https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/data/iris.csv")
print("The 10th percentile value is:",np.percentile(iris['sepal_length'],10))
print("The 20th percentile value is:",np.percentile(iris['sepal_length'],20))
print("The 30th percentile value is:",np.percentile(iris['sepal_length'],30))
print("The 40th percentile value is:",np.percentile(iris['sepal_length'],40))
print("The 50th percentile value is:",np.percentile(iris['sepal_length'],50))
```

```
The 10th percentile value is: 4.8
The 20th percentile value is: 5.0
The 30th percentile value is: 5.27
The 40th percentile value is: 5.6
The 50th percentile value is: 5.8
```

Naive implementation in python: In this code we will see how to calculate the percentile from first principle.

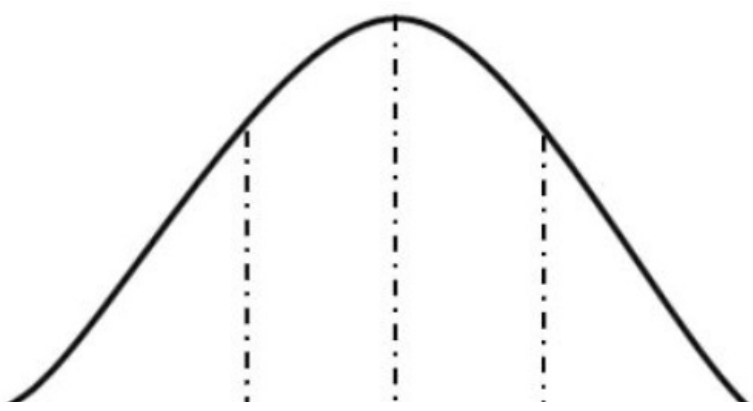
In [31]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import warnings
warnings.filterwarnings('ignore')
'''downlaod iris.csv from https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/data/iris.csv'''
#Load Iris.csv into a pandas dataFrame.
iris = pd.read_csv("https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/data/iris.csv")

def naive_percentile(dataframe,percentile):
    '''This function will calculate the percentile'''
    percentile_value=((percentile/100)*len(dataframe))
    sorted_df=dataframe.sort_values()
    sorted_df=sorted_df.reset_index(drop=True)
    print("The {}th percentile value is: {}".format(percentile,sorted_df[percentile_value]))

naive_percentile(iris['sepal_length'],10)
naive_percentile(iris['sepal_length'],20)
naive_percentile(iris['sepal_length'],30)
naive_percentile(iris['sepal_length'],40)
naive_percentile(iris['sepal_length'],50)
```

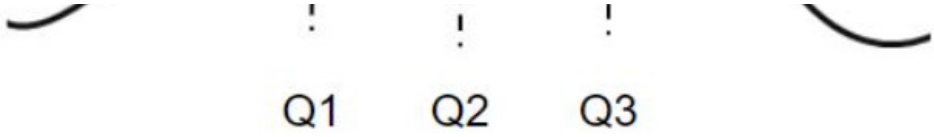
```
The 10th percentile value is: 4.8
The 20th percentile value is: 5.0
The 30th percentile value is: 5.3
The 40th percentile value is: 5.6
The 50th percentile value is: 5.8
```

Quartile:

A quartile is a type of quantile which divides the number of data points into four more or less equal parts or quarters.

1. 1. **Q1(First quratile/Lower quartile/25th percentile):** Splits off the lowest 25% of data from the highest 75%.
2. 2. **Q2(Second quartile/Median/50th percentile):** Cuts the dataset into half.
3. 3. **Q3(Third quartile/Upper quartile/75th percentile):** Splits off the highest 25% of data from the lowest 75%.





Implementation in python:

In [32]:
```
import numpy as np

data=range(0,101)

print("Q1 of data is: ",np.quantile(data,0.25))
print("Q2 of data is: ",np.quantile(data,0.50))
print("Q3 of data is: ",np.quantile(data,0.75))
print("Q4 of data is: ",np.quantile(data,0.1))
```

```
Q1 of data is:  25.0
Q2 of data is:  50.0
Q3 of data is:  75.0
Q4 of data is:  10.0
```

In [33]:
```
import numpy as np

data=range(0,501)

print("Q1 of data is: ",np.quantile(data,0.25))
print("Q2 of data is: ",np.quantile(data,0.50))
print("Q3 of data is: ",np.quantile(data,0.75))
print("Q4 of data is: ",np.quantile(data,0.1))
```

```
Q1 of data is:  125.0
Q2 of data is:  250.0
Q3 of data is:  375.0
Q4 of data is:  50.0
```

IQR:

- The interquartile range(IQR) is the difference between the 75th and 25th percentile of the data. It is a measure of the dispersion similar to standard deviation or variance, but is much more robust against outliers
- The interquartile range(IQR),also called as midspread or middle 50% or technically H-spread is the difference between the third quartile(Q3) and the first quartile(Q1).
- IQR is a amount of spread in the middle 50% of a dataset

$$\mathrm{IQR} = \mathrm{Q}_3 - \mathrm{Q}_1$$

Implementation in python:

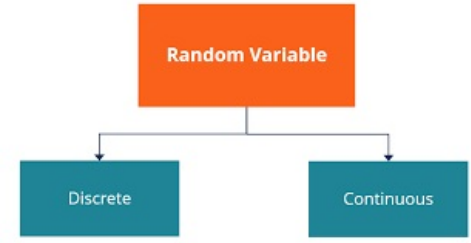
In [34]:
```
from scipy import stats

data = [32, 36, 46, 47, 56, 69, 75, 79, 79, 88, 89, 91, 92, 93, 96, 97, 101, 105, 112, 116]

IQR = stats.iqr(data)

print(IQR)
```

```
30.5
```

Random variable:

A random variable is a set of possible values from a random experiment.
We use a capital letters like **X** or **Y**(variable name can be anything) for random variables.

Random variable are of two types discrete or continous Random variable:



Discrete Random Variable: A discrete random variable is one which may take on only a countable number of distinct values and thus can be quantified.
Example:

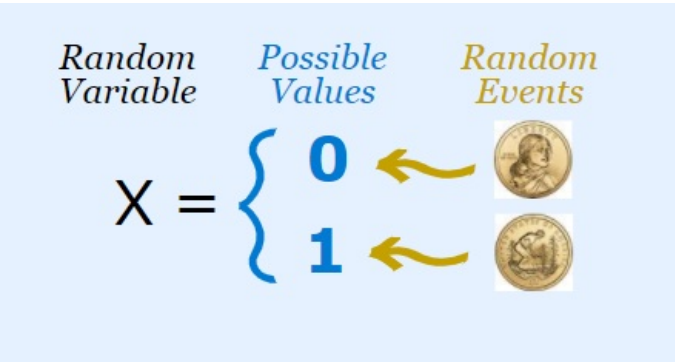
1. **Coin toss:** You can define a random variable X to be the side which comes up when you toss a coin. X can take values : [H,T] and therefore is a discrete random variable.
2. **Rolled fair dice**: You can define a random variable X to be the number which comes up when you roll a fair dice. X can take values : [1,2,3,4,5,6] and therefore is a discrete random variable.

Some examples of discrete probability distributions are Bernoulli distribution, Binomial distribution, Poisson distribution etc.

Continuous Random Variable: A random variable which can take infinite number of values in an interval is known as continuous random variable.
Example:

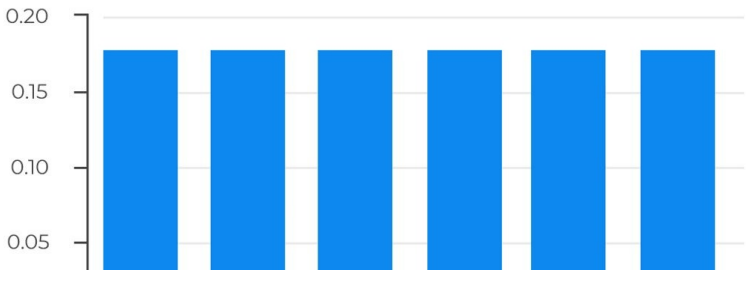
1. **Weight of a group of individuals:** a random variable X to be the height of students in a class
2. **Price of house:** A random variable X to be the price of house in a city

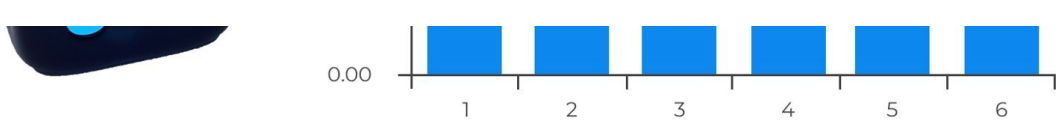
Example: Tossing a coin(discrete random variable).



If we toss a coin we have only two possible outcome either we get **"Head"** or **"Tail"**.so, our random variable **"X"** contains two possible outcome: X={H,T}

Example: Rolling a fair dice(discrete random variable).





we know in fair dice we have 1,2,3,4,5,6 on a single side of a dice.

If we roll a fair dice we have six possible outcomes either **1,2,3,4,5,6**. So, our random variable **"X"** contain six possible outcome:

X={1,2,3,4,5,6}

Example: Height of a randomly picked student(continous random variable)

We know that height lie between the intervals. so we are taking the intervals is [4'2",6'2"]

H={130.2,140.9,172.6,180.3,150,122.1}

Population and Sample:

Population Distribution: The population is the whole set of values, or individuals, you are interested in.
For example: If you want to know the average height of the residents of india, that is your population. i.e., the population of India

Population
Mean =μ
Variance $\sigma^2 = \frac{\sum (X_i - \mu)^2}{N}$

Standard Deviation	$\sigma = \sqrt{\frac{\sum(X_i - \mu)^2}{N}}$

Sample Distribution: The sample is a subset of the population, and is the set of values you actually use in your estimation. Let us think 1000 individuals you have selected for your study to know about average height of the residents of India. This sample has computed from values

Sample
Mean= $\bar{x}$
Variance $s^2 = \sum_{i=1}^{n} \frac{(x_i - \bar{x})^2}{n-1}$
Standard Deviation $s = \sqrt{\frac{\sum(x - \bar{x})^2}{N-1}}$

Real life application of population and sample: Let us understand this with an example:

Question:We have to find out the average height of the mens in India?

Answer: Suppose you have to find out the average height of the mens in India. We have **532 millions** mens in India.

The approach I get from aspiring data scientists is to simply calculate the average:

- First, measure the weights of all the people of india.
- Add all the weights
- Finally, divide the total sum of weights with a total number of people to get the average

We have 532 million people in India. The size of the data is homogeneous. Does this approach make sense? Not really – measuring the height of all the people of India will be a very tiresome and long process. So, what can we do instead? Let's look at an alternate approach.

So, what we can do is randomly and independently pick the 10,000 people as sample and find the average height of 10,000 mens. Which picked up unbiasly.

so the intuition is sample mean is roughly equal to the population means

$\mu \approx \bar{x}$

Note: When the sample size increases the mean of sample try to converge towards μ(mean of population)

Naive implementation in python: In this code example what i did is take the flower feature as population and compute the mean and then take the sample unbiasely from the population and compute the mean. You can see sample mean try to give the good approximation of population mean.

In [35]:
```
import numpy as np
import random
import pandas as pd

'''downlaod iris.csv from https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/data/iris.csv'''
#Load Iris.csv into a pandas dataFrame.
```

```
iris = pd.read_csv("https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/data/iris.csv")

def population_mean(data):
    '''This function will calculate the mean of population'''
    population_height_of_india_mens=np.array(data)
    population_mean=population_height_of_india_mens.mean()
    return population_mean

def sample_mean(data,size_of_sample):
    '''This function will calculate the mean of sample of size 45'''
    population_height_of_india_mens=data
    sample_height_of_india_mens=[]
    for i in range(size_of_sample):
        choice=random.choice(population_height_of_india_mens)
        sample_height_of_india_mens.append(choice)
    sample_height_of_india_mens_array=np.array(sample_height_of_india_mens)
    sample_mean=sample_height_of_india_mens_array.mean()
    return sample_mean

print("The population mean of sepal length is: ",population_mean(iris['sepal_length']))
print("The sample mean of sepel length is: ",sample_mean(iris['sepal_length'],45))
print()
print("The population mean of sepal width is: ",population_mean(iris['sepal_width']))
print("The sample mean of sepel width is: ",sample_mean(iris['sepal_width'],45))
print()
print("The population mean of petal length is: ",population_mean(iris['petal_length']))
print("The sample mean of petal length is: ",sample_mean(iris['petal_length'],45))
print()
print("The population mean of petal width is: ",population_mean(iris['petal_width']))
print("The sample mean of petal width is: ",sample_mean(iris['petal_width'],45))
```

```
The population mean of sepal length is:  5.843333333333334
The sample mean of sepel length is:  5.937777777777779

The population mean of sepal width is:  3.0540000000000003
The sample mean of sepel width is:  3.011111111111111

The population mean of petal length is:  3.758666666666666
The sample mean of petal length is:  3.8933333333333335

The population mean of petal width is:  1.1986666666666668
The sample mean of petal width is:  1.3488888888888888
```

What is PDF and CDF

The PDF and CDF statistical functions are widely used techniques in the Exploratory Data Analysis to find the probabilistic relationships between the variables.

PDF(Probablity Density Function)

PDF of a random variable is the plot between the random variable and the frequency of that random variable. It gives the probability distribution of the random variable. For a random variable, given the distribution type, mean and, variance, we can draw the PDF. Here we are able to construct a PDF, without the help of actual data points.

The Probability Density Function(PDF) defines the probability function representing the density of a continuous random variable lying between a specific range of values. In other words, the probability density function produces the likelihood of values of the continuous random variable. Sometimes it is also called a probability density function or just a probability function.

Probability Density Function Formula

In the case of a continuous random variable, the probability taken by X on some given value x is always 0. In this case, if we find P(X = x), it does not work. Instead of this, we must calculate the probability of X lying in an interval (a, b). Now, we have to figure it for P(a< X< b), and we can calculate this using the formula of PDF. The Probability density function formula is given as,

$$\mathbf{P(a < X < b) = \int_a^b f(x)\,dx}$$

Or

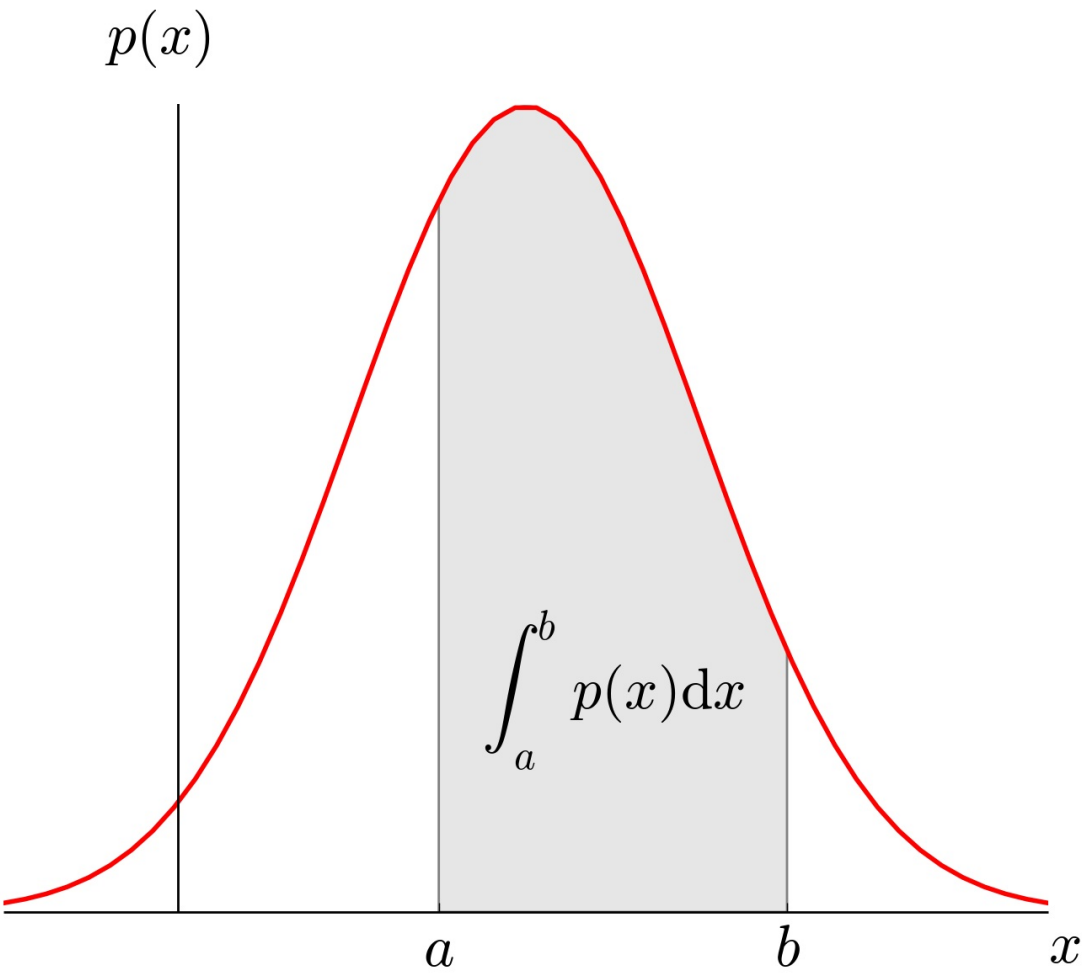
$$\mathbf{P(a \leq X \leq b) = \int_a^b f(x)\,dx}$$

This is because, when X is continuous, we can ignore the endpoints of intervals while finding probabilities of continuous random variables. That means, for any constants a and b,

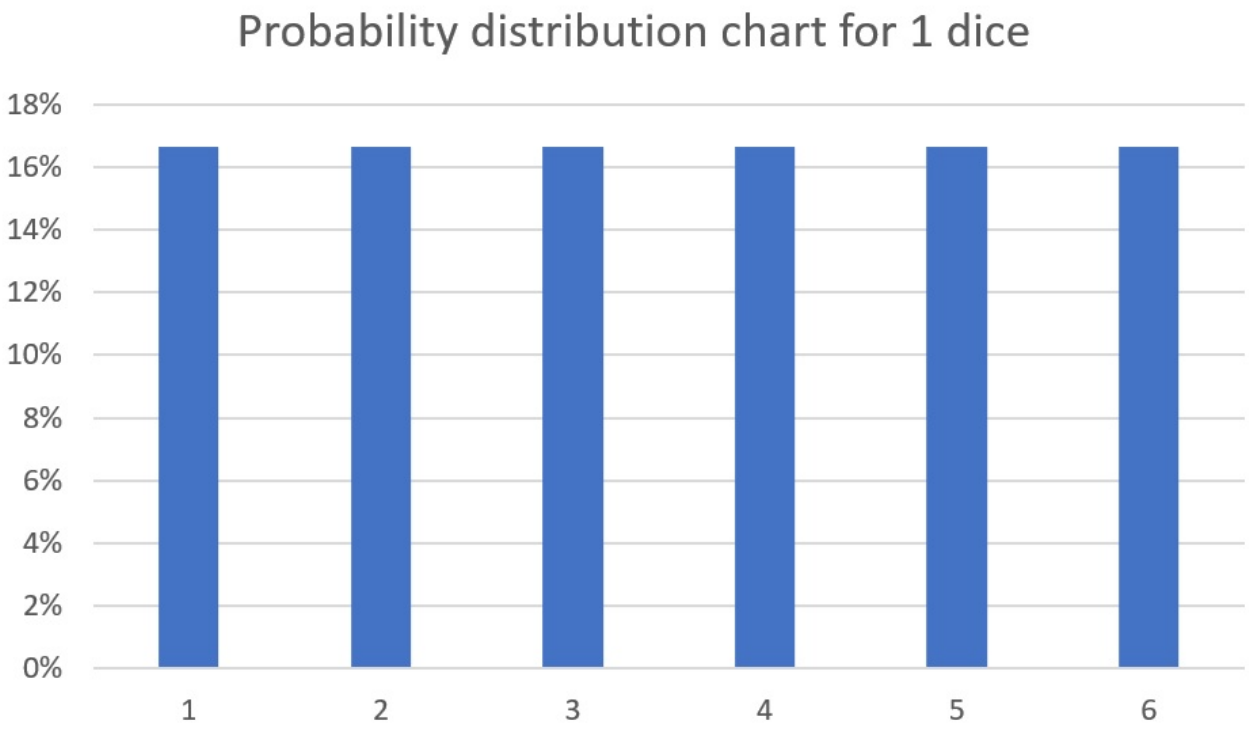
$P(a \le X \le b) = P(a < X \le b) = P(a \le X < b) = P(a < X < b)$.

Probability Density Function Graph

The probability density function is defined as an integral of the density of the variable density over a given range. It is denoted by f (x). This function is positive or non-negative at any point of the graph, and the integral, more specifically the definite integral of PDF over the entire space is always equal to one. The graph of PDFs typically resembles a bell curve, with the probability of the outcomes below the curve. The below figure depicts the graph of a probability density function for a continuous random variable x with function f(x).



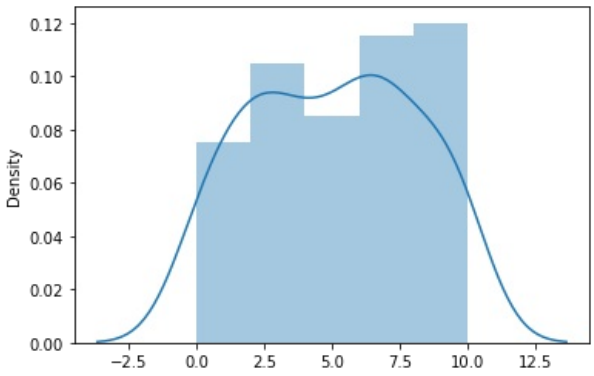
For example, if you roll a die, the probability of obtaining 1, 2, 3, 4, 5, or 6 is 16.667% (=1/6). The probability density function (PDF) or the probability that you will get exactly 2 will be 16.667%.



Naive implementation in python: In this code we randomly generate the value in between 0-10 and plot the pdf.

In [36]:
```
from random import seed
from random import randint
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

seed(10)
value=[]
for i in range(100):
    choice = randint(0, 10)
    value.append(choice)

sns.distplot(value)
```

Out[36]: <AxesSubplot:ylabel='Density'>



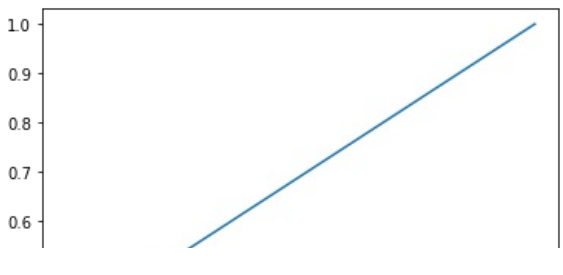
CDF(Cumulative density function):

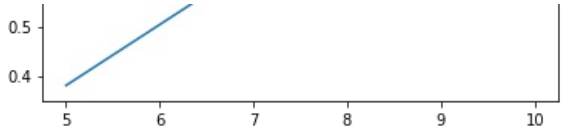
The cumulative distribution function is used to describe the probability distribution of random variables. It can be used to describe the probability for a discrete, continuous or mixed variable. It is obtained by summing up the probability density function and getting the cumulative probability for a random variable.

The Probability Density Function is a function that gives us the probability distribution of a random variable for any value of it. To get the probability distribution at a point, you only have to solve the probability density function for that point.

The cumulative distribution function of a random variable to be calculated at a point x is represented as Fx(X). It is the probability that the random variable X will take a value less than or equal to x.

In [37]:
```
from random import seed
from random import randint
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

value=[]
for i in range(100):
    choice = randint(0, 10)
    value.append(choice)

counts,bin_edges=np.histogram(value,bins=2,density=True)
pdf=counts/sum(counts)
cdf=np.cumsum(pdf)
plt.plot(bin_edges[1:],cdf)
plt.show()
```





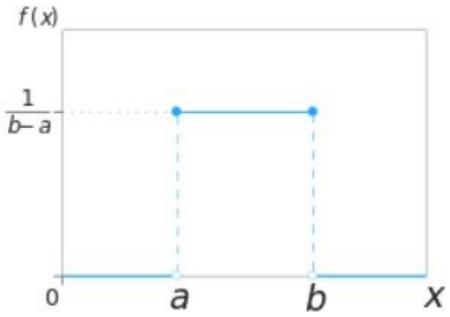
Important point to know

Qus: PDF is not a probablity?

Ans: It is a well known fact that the largest value of a probablity can take is 1. However, for some PDF you will encounter with greater than 1.

Qus: How can a PDF value is greater than 1 and its probablity integrates to 1?

Ans: Even if the PDF f(x) takes on value greater than 1, if the domain that it integrates over is less than 1,it can add upto only 1. Let's take an example of the easiest PDF the uniform distribution defined on the domain [0,0.5]. The PDF of uniform distribution is 1/(b-a), which is continuously 2 throughout



The PDF of uniform distribution

The total probability is the total area under the graph f(x), which is 2 * 0.5 = 1. As you can see, even if a PDF is greater than 1, because it integrates over the domain that is less than 1, it can add up to 1.

Qus: The difference between the PDF and probability?

Ans: Isn't the PDF f(x) a probability? No. Because f(x) can be greater than 1. ("PD" in PDF stands for "Probability Density," not Probability.) f(x) is just a height of the PDF graph at X = x. The whole "PDF = probability" misconception comes about because we are used to the notion of "PMF = probability", which is, in fact, correct. However, a PDF is not the same thing as a PMF, and it shouldn't be interpreted in the same way as a PMF, because discrete random variables and continuous random variables are not defined the same way. For discrete random variables, we look up the value of a PMF at a single point to find its probability P($\mathbf{X}$=x) For continuous random variables, we take an integral of a PDF over a certain interval to find its probability that X will fall in that interval.

Qus: Why is the probability of a continuous random variable at every point zero?

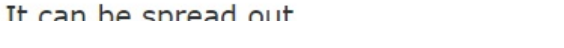
Ans: To answer the this question, we need to answer the following question first: How many total real numbers do we have in [0, 0.5]? Infinite ∞. (To be mathematically thorough, uncountably infinite.) 0.1, 0.01, 0.001, 0.0001,… You can keep inserting 0 in front of the smallest decimal. Therefore, a continuous random variable has an infinite # of possible values that it can take, even if the domain is small and fixed. And let's say, the probability density for each value in [0, 0.5] takes an extremely small value, e.g. 000000001. Still, the sum of infinitely (uncountably) many values will reach infinity, no matter how small their values are. Then, in order to make the sum of probabilities 1, the probability at a single point should be 1/∞, which is 0. Well, this doesn't really make sense, either. If you add the infinite number of zeros, you will still get zero. The total probability should add up to one, not to zero.

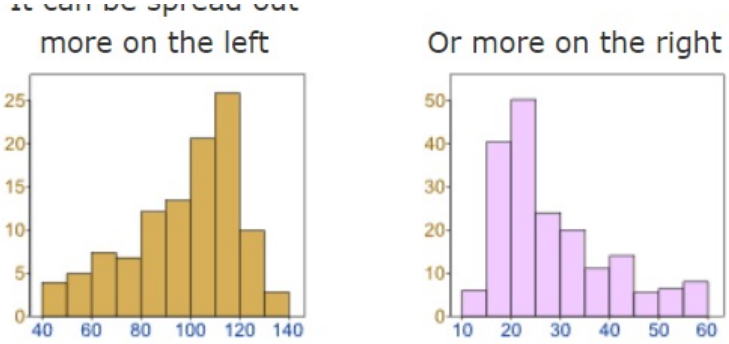
Qus: Then how do we calculate the probability from the probability density f(x)?

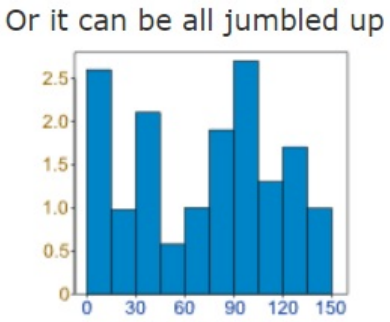
Ans: We will borrow the idea from the "integral".
If the probability of X being exactly at point x is zero, how about an extremely small interval around the point x? Say, [x, x+dx]? Let's say dx is 0.00000000001.
Then the probability that X will fall in [x, x+dx] is the area under the curve f(x) sandwiched by [x, x+dx].
If dx is infinitesimally small, this approximation will be good enough for P($\mathbf{X}$=x).

Gaussain/Normal Distribution:

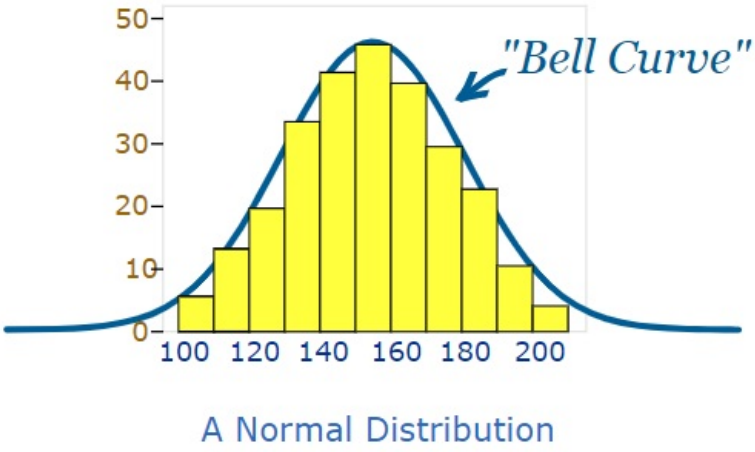
Data can be **"distributed"**(spread out) in different ways:







But there are many cases where the data tends to be around a central value with no bias left or right and it gets close to a "Normal Distribution" like this:



The Bell curve is a normal distribution and the histogram shows some data that follows it closely, but not perfecty(which is usual).

$X \sim N(\mu,\sigma^2)$

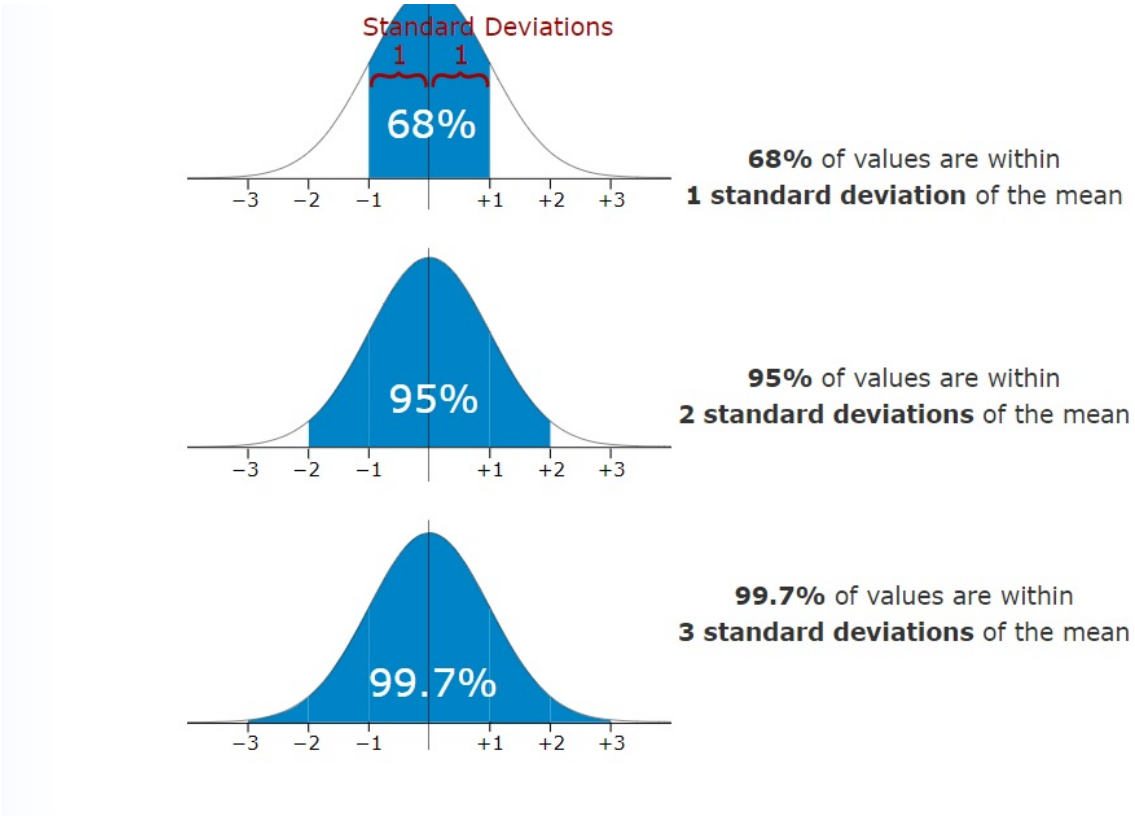
This notation represent as Random variable X follows Normal distribution with mean(μ) and variance(σ^2)

The Gaussain/Normal Distribution has:

1. mean=mode=median
2. symmetric about center
3. 50% value less than mean and 50% value greater than mean

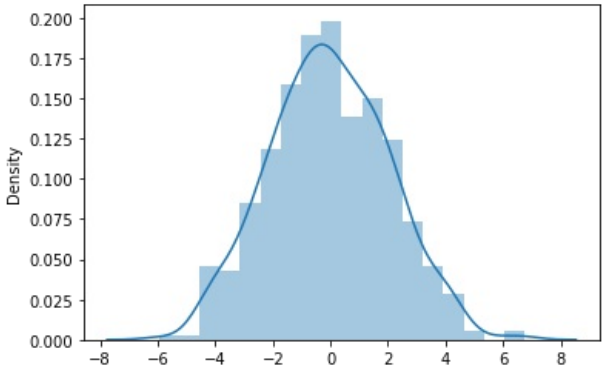
The Gaussain/Normal Distribution 68-95-99.7 property:

$pro[\mu-\sigma<=X<=\mu+\sigma]\approx68\%$
$pro[\mu-2\sigma<=X<=\mu+2\sigma]\approx95\%$
$pro[\mu-3\sigma<=X<=\mu+3\sigma]\approx99.7\%$



Implementation in python:

In [38]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

s1=np.random.normal(0,2,500)      #create a gaussian distribution with mean 0,standarad deviation 2,size 500
sns.distplot(s1)
plt.show()
```

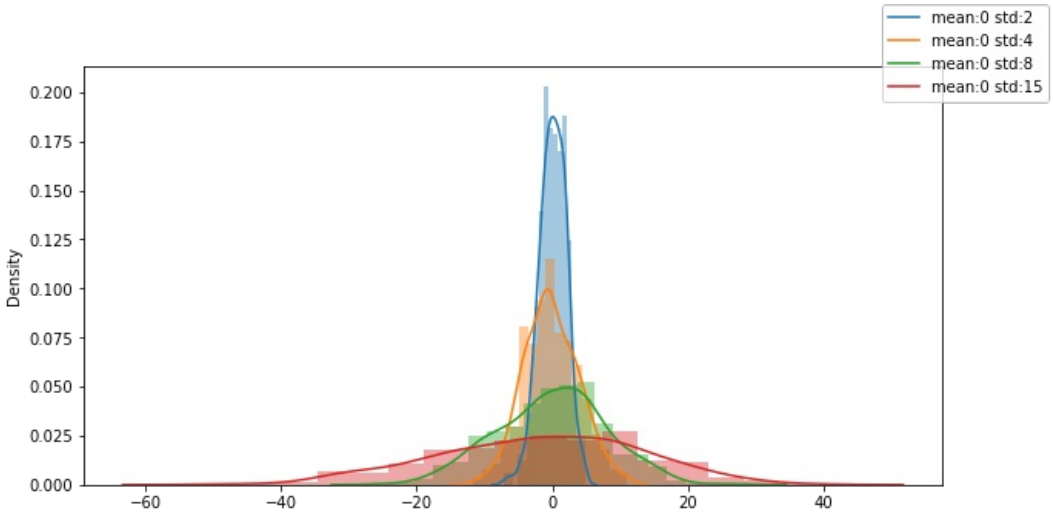


The Gaussain/Normal Distribution property:

- The gaussian distribution retains the shape throughout, unlike other probablity distribution that changes their properties after a transformtion like product/sum/convolution/fourier transformation of any two normal distribution is also a normal distribution(Once a gaussian distribution always a gaussian distribution)
- The standard deviation defines the width of the graph. Usually a smaller standard deviation wrt the mean result in steap curves while a larger standard deviation results in flatter curve.(Let's see graphically below).

In [39]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

fig = plt.figure(figsize=(10,5))
s1=np.random.normal(0,2,500)      #create a gaussian distribution with mean 0,standarad deviation 2,size 500
s2=np.random.normal(0,4,500)      #create a gaussian distribution with mean 0,standarad deviation 4,size 500
s3=np.random.normal(0,8,500)      #create a gaussian distribution with mean 0,standarad deviation 8,size 500
```

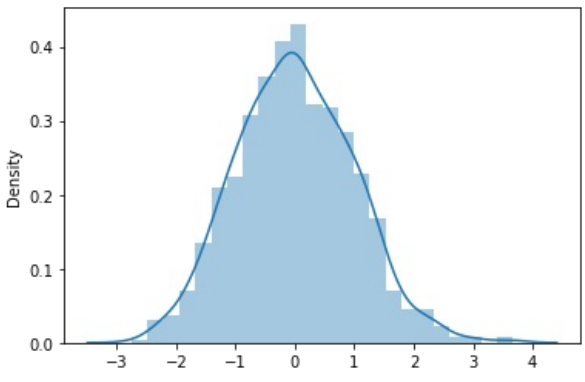
```
s4=np.random.normal(0,15,500)     #create a gaussian distribution with mean 0,standarad deviation 15,size 500
sns.distplot(s1)
sns.distplot(s2)
sns.distplot(s3)
sns.distplot(s4)
fig.legend(labels=['mean:0 std:2','mean:0 std:4', 'mean:0 std:8', 'mean:0 std:15'])
plt.show()
```



Code Explanation: In the above you can we have created 4 gaussian curve with mean 0 and different different standard deviation like 2,4,8,15. You can see as standard deviation start increasing the curve starts flattened.

Interesting experiment: In this experiment what we try is to check for empericial formulae.

In [40]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statistics
import warnings
warnings.filterwarnings('ignore')

def check_emperical_formulae(dataset,mean,stddev):
    '''This function check empericial rule'''
    count1std=0
    count2std=0
    count3std=0

    for i in dataset:
        if(mean-(3*stddev)<i and i<mean+(3*stddev)):                    #data checking within 1 standard deviat
            count3std=count3std+1
    for i in dataset:
        if(mean-(2*stddev)<i and i<mean+(2*stddev)):              #data checking within 2 standard deviation
            count2std=count2std+1
    for i in dataset:
        if(mean-(stddev)<i and i<mean+(stddev)):              #data checking within 3 standard deviation
            count1std=count1std+1
    proportion1=(count1std/len(s1))*100
    proportion2=(count2std/len(s1))*100
    proportion3=(count3std/len(s1))*100
    print("data within 1 std dev: ",proportion1)
    print("data within 2 std dev: ",proportion2)
    print("data within 3 std dev: ",proportion3)


s1=np.random.normal(0,1,1000)      #create a gaussian distribution with mean 0,standarad deviation 2,size 1000
sns.distplot(s1)
plt.show()
check_emperical_formulae(s1,statistics.mean(s1),statistics.stdev(s1))
```



```
data within 1 std dev:  67.4
data within 2 std dev:  95.5
data within 3 std dev:  99.6
```

Real life example that follows normal distribution:

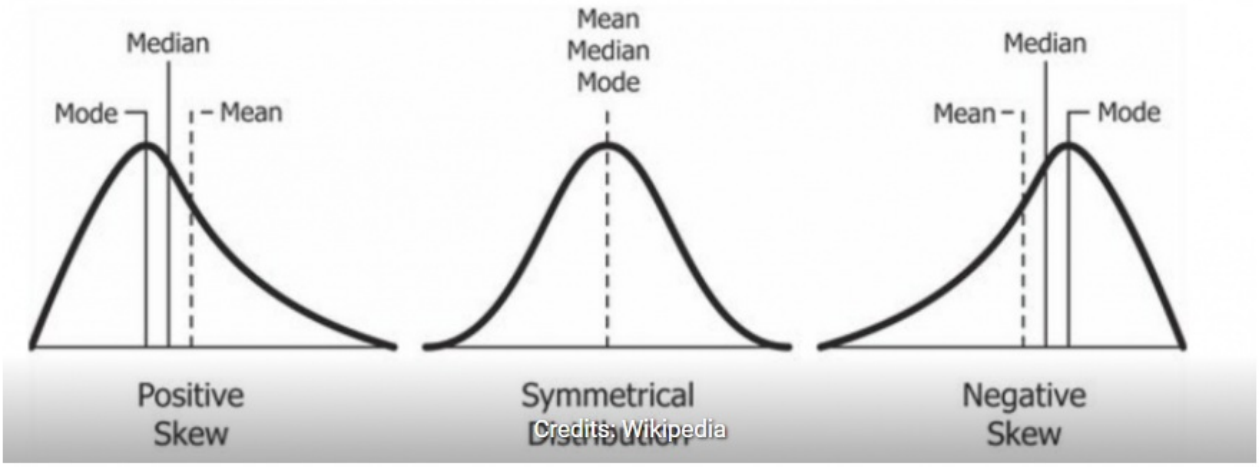
1. Heights of people
2. Blood Pressure
3. Marks on exams
4. Size of things produced by machines

Skewness:

skewness is the measure of how much the probablity distribution of random variable deviates from the normal distribution.
well, the normal distribution is the probablity distribution without skewness.

Well, the normal distribution is the probability distribution without any skewness. You can look at the image below which shows symmetrical distribution that's basically a normal distribution and you can see that it is symmetrical on both sides of the dashed line. Apart from this, there are two types of skewness:

- Positive Skewness
- Negative Skewness



Formula For Skewness Calculate:

$$\gamma_1 = \frac{\frac{1}{n}\sum_{i=1}^{n}\left(x_i - \bar{x}\right)^3}{\left(\frac{1}{n}\sum_{i=1}^{n}\left(x_i - \bar{x}\right)^2\right)^{3/2}}$$

OR

$$\textbf{Skewness} = \frac{\boldsymbol{3\,(Mean - Median)}}{\boldsymbol{Standard\ Deviation}}$$

γ1 represents coefficient of skewness

xi represents ith value in data vector

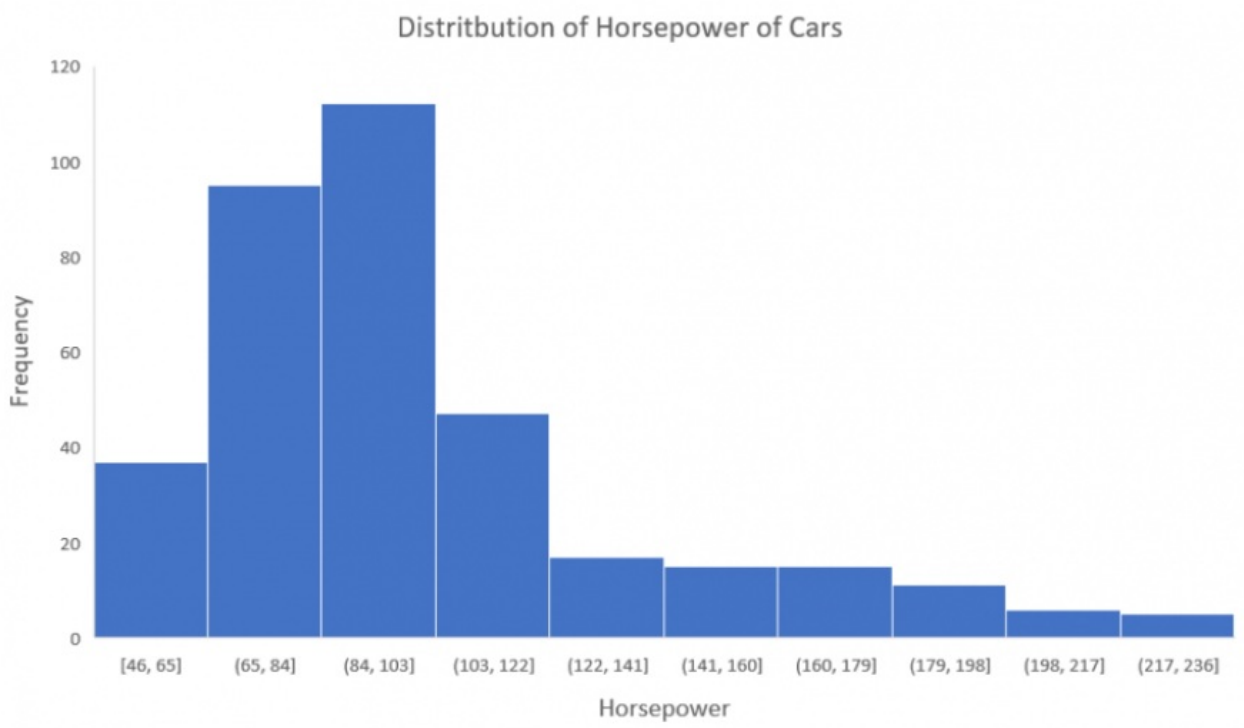
xbar represents mean of data vector

n represents total number of observations

The probability distribution with its tail on the right side is a positively skewed distribution and the one with its tail on the left side is a negatively skewed distribution.

Why is Skewness Important?

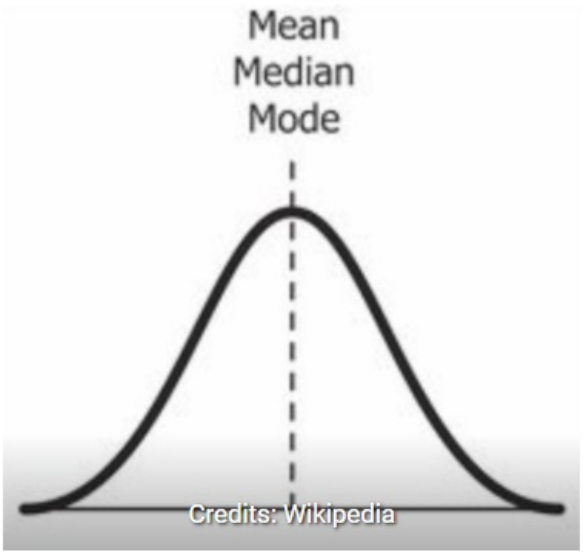
First, linear models work on the assumption that the distribution of the independent variable and the target variable are similar. Therefore, knowing about the skewness of data helps us in creating better linear models.
Secondly, let's take a look at the below distribution. It is the distribution of horsepower of cars:



You can clearly see that the above distribution is positively skewed. Now, let's say you want to use this as a feature for the model which will predict the mpg (miles per gallon) of a car.
Since our data is positively skewed here, it means that it has a higher number of data points having low values, i.e., cars with less horsepower. So when we train our model on this data, it will perform better at predicting the mpg of cars with lower horsepower as compared to those with higher horsepower.
Also, skewness tells us about the direction of outliers. You can see that our distribution is positively skewed and most of the outliers are present on the right side of the distribution.

Note: The skewness does not tell us about the number of outliers. It only tells us the direction.

What is Symmetric/Normal Distribution?



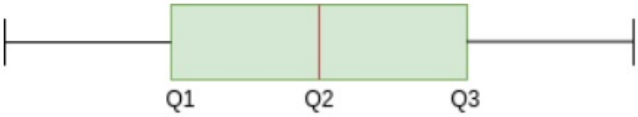
Yes, we're back again with the normal distribution. It is used as a reference for determining the skewness of a distribution. As I mentioned earlier, the ideal normal distribution is the probability distribution with almost no skewness. It is nearly perfectly symmetrical. Due to this, the value of skewness for a normal distribution is zero.

But, why is it nearly perfectly symmetrical and not absolutely symmetrical?

That's because, in reality, no real word data has a perfectly normal distribution. Therefore, **even the value of skewness is not exactly zero; it is nearly zero**. Although the value of zero is used as a reference for determining the skewness of a distribution.

You can see in the above image that the same line represents the mean, median, and mode. It is because the mean, median, and mode of a perfectly normal distribution are equal.

So far, we've understood the skewness of normal distribution using a probability or frequency distribution. Now, let's understand it in terms of a boxplot because that's the most common way of looking at a distribution in the data science space.



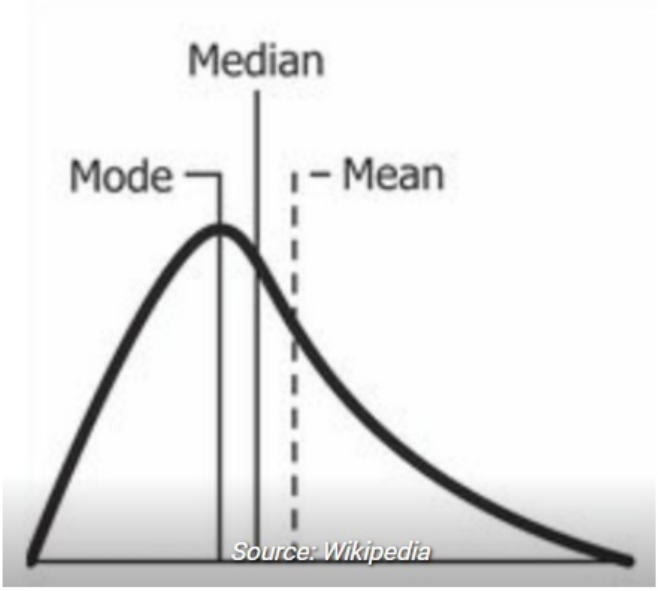
The above image is a boxplot of symmetric distribution. You'll notice here that the distance between Q1 and Q2 and Q2 and Q3 is equal i.e.:

$$Q_3 - Q_2 = Q_2 - Q_1$$

But that's not enough for concluding if a distribution is skewed or not. We also take a look at the length of the whisker. if they are equal, then we can say that the distribution is symmetric, i.e. it is not skewed.

Now that we've discussed the skewness in the normal distribution, it's time to learn about the two types of skewness which we discussed earlier. Let's start with positive skewness.

Understanding Positively Skewed Distribution



A positively skewed distribution is the distribution with the tail on its right side. The value of skewness for a positively skewed distribution is greater than zero. As you might have already understood by looking at the figure, the value of mean is the greatest one followed by median and then by mode.

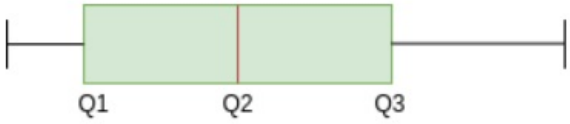
So why is this happening?

Well, the answer to that is that the skewness of the distribution is on the right; it causes the mean to be greater than the median and eventually move to the right. Also, the mode occurs at the highest frequency of the distribution which is on the left side of the median. Therefore, **mode < median < mean**.

In the above boxplot, you can see that Q2 is present nearer to Q1. This represents a positively skewed distribution. In terms of quartiles, it can be given by:

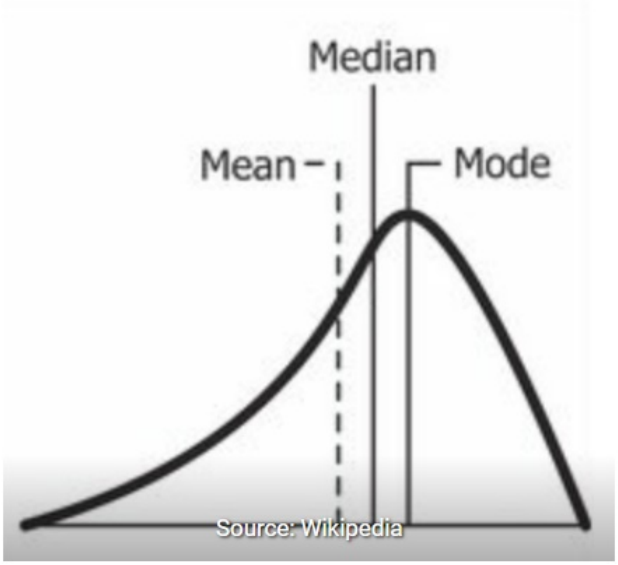
$$Q_3 - Q_2 > Q_2 - Q_1$$

In this case, it was very easy to tell if the data is skewed or not. But what if we have something like this:



Here, Q2-Q1 and Q3-Q2 are equal and yet the distribution is positively skewed. The keen-eyed among you will have noticed the length of the right whisker is greater than the left whisker. From this, we can conclude that the data is positively skewed.
So, the first step is always to check the equality of Q2-Q1 and Q3-Q2. If that is found equal, then we look for the length of whiskers.

Understanding Negatively Skewed Distribution



As you might have already guessed, a negatively skewed distribution is the distribution with the tail on its left side. The value of skewness for a negatively skewed distribution is less than zero. You can also see in the above figure that the **mean < median < mode.**

In the boxplot, the relationship between quartiles for a negative skewness is given by:

$$Q_3 - Q_2 < Q_2 - Q_1$$

Similar to what we did earlier, if Q3-Q2 and Q2-Q1 are equal, then we look for the length of whiskers. And if the length of the left whisker is greater than that of the right whisker, then we can say that the data is negatively skewed.

Q1 Q2 Q3

when

skewness=0 : normally distributed

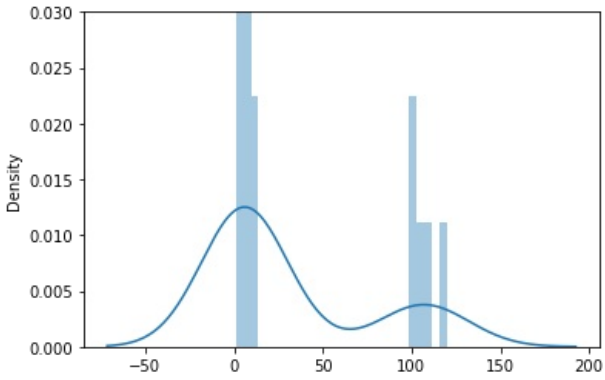
skewness<0 : more weight on the left hand side / Negative skewness

skewness>0 : more weight on the right hand side / Positive skewness

implementation in python:

In [41]:
```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import skew

a=[1,2,3,4,5,5,6,6,6,7,7,8,9,10,11,100,102,104,110,120]
sns.distplot(a)
plt.ylim(0,0.03)

print("Skewness of the data is:",skew(a))
```

```
Skewness of the data is: 1.2439594089986779
```

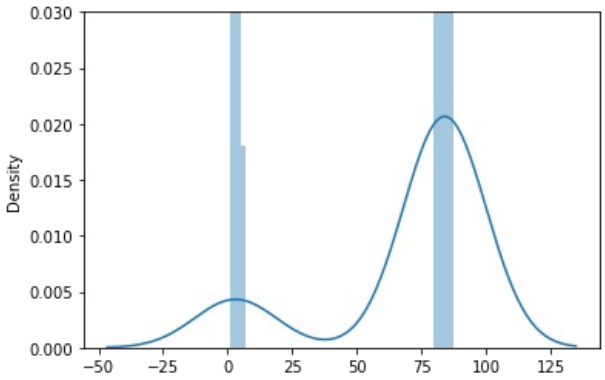


Explanation: As we can see this plot is positive skew because the value of skewness is greater than 1

implementation in python:

In [42]:
```
import numpy as np
import seaborn as sns
from scipy.stats import skew

a=[1,2,3,4,5,80,80,82,82,82,83,83,83,83,84,84,84,84,84,85,85,85,85,85,85,86,86,86,87]
sns.distplot(a)
plt.ylim(0,0.03)

print("Skewness of the data is:",skew(a))
```

```
Skewness of the data is: -1.7250964422584605
```



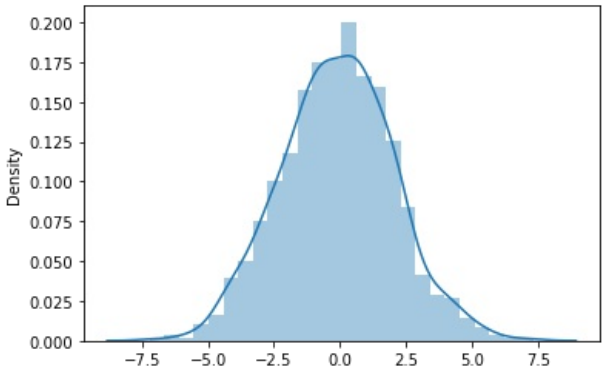
Explanation: As we can see this plot is negative skew because the value of skewness is less than 1

implementation in python:

In [43]:
```
import numpy as np
```

```
import seaborn as sns
from scipy.stats import skew

a=np.random.normal(0,2,1000)

sns.distplot(a)

print("Skewness of the data is:",skew(a))
```

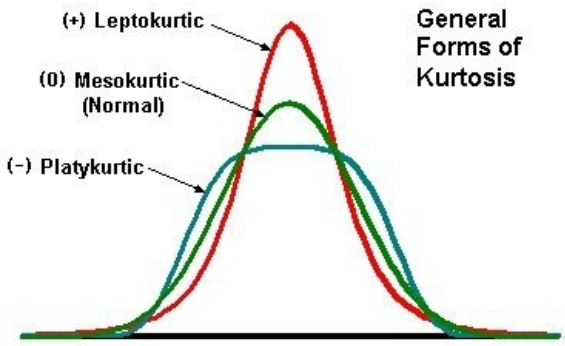
```
Skewness of the data is: 0.03407169415814316
```



Explanation: As we can see this plot follow nearly normal distribution that is why it have skew value nearly about 0(zero)

Kurtosis:

- Kurtosis is the measure of thickness or heaviness of the given distribution.
- Its actually represents the height of the distribution.
- Like skewness, Kurtosis is a descriptor of shape and it describes the shape of the of the distribution in terms of height or flatness. Some of the types of Kurtosis are Leptokurtic, Platykurtic and Mesokurtic.



It Is three types

1. mesokurtic (distribution is normal)
2. platykurtic (negative excess of kurtosis)
3. leptokurtic (positive excess of kurtosis)

Formulae:

$$\text{Kurtosis (X)} = E\left[\left(\frac{X-\mu}{\sigma}\right)^4\right]$$

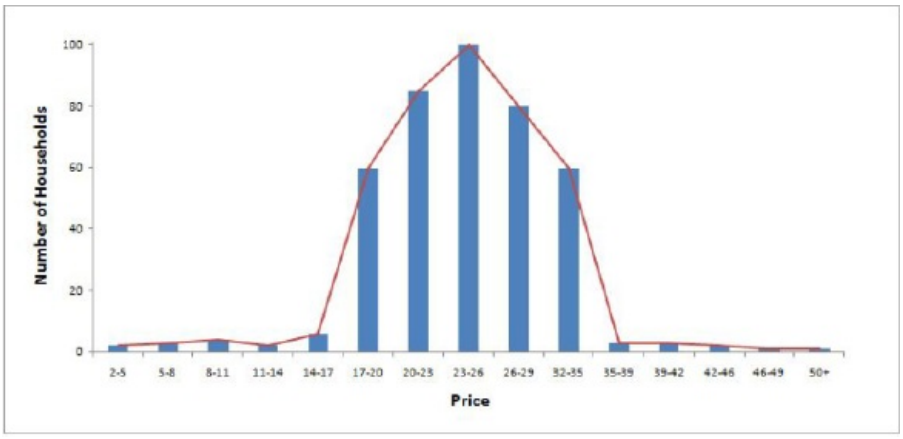
Important Parameter:

- array : Input array or object having the elements.
- axis : Axis along which the kurtosis value is to be measured. By default axis = 0.
- fisher : Bool; Fisher's definition is used (normal 0.0) if True; else Pearson's definition is used (normal 3.0) ifset to False.
- bias : Bool; calculations are corrected for statistical bias, if set to False.

- Returns : Kurtosis value of the normal distribution for the data set

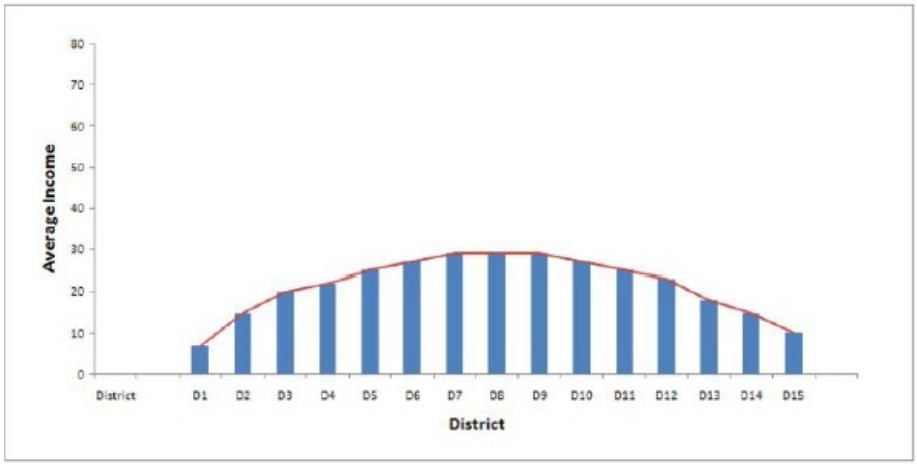
1. Leptokurtic

- When there is a positive excess of kurtosis, the shape of the distribution is called Leptokurtic. To understand this in terms of shape, it has fatter tails and if compared to a normal distribution it has a similar peak (to be precise, such a distribution has higher peak than what is found in a normal bell-shaped distribution and significantly higher if compared to a Platykurtic distribution) and has values clustered around the centre (mean).
- Example- If you are asked to collect a sample to find out the average price of the car that people own in Delhi and you decide to go only in the upper-middle-class localities, then the shape of such a sample'sdistribution will be Leptokurtic.



2. Platykurtic

- When there is a negative excess of kurtosis, the shape of the distribution is called Platykurtic.
- The data points are highly dispersed along the X-axis that results in thinner tails when compared to a normal distribution and has very few values clustered around the centre (mean). Such a distribution will have little central tendency.
- Example- You are asked to go to 15 districts or localities of your city to collect a sample to find out the average income of the state. You decide to go to each district and find the average income of each district by randomly choosing 40 people and finding their annual income. But when the samples were finally plotted on a histogram, the shape of the distribution seemed to be Platykurtic, because all the 15 districts you chose to go were where government households were located where income of people fell in the middle-income band with all having more or less the same average income causing the shape of your distribution to become flatter than the normal.



3. Mesokurtic

- This is when the distribution is normal. Here the tails of the distribution are neither too thin nor they are too thickand also the scores are equally divided with scores neither being clustered around the centre nor being too scattered.

Mathematically solved: Let's see here how to calculate the skewness and kurtosis of the datapoints.

Qus: Find the skewness and kurtosis of this dataset: 13,14,18,19,21.

Ans:

xi	(xi-x̄)	(xi-x̄)^2	(xi-x̄)^3	(xi-x̄)^4
13	-4	16	-64	256
14	-3	9	-27	81
18	1	1	1	1
19	2	4	8	16
21	4	16	64	256
85	**0**	**46**	**-18**	**610**

Mean $\bar{x} = \bar{x} = \frac{\Sigma x_i}{N} = (13 + 14 + 18 + 19 + 21)/5 = 85/5 = 17$

First central moment $= \mu_1 = \frac{\Sigma(x_i - \bar{x})^1}{N} = 0/5 = 0$

Second central moment $= u_2 = \frac{\Sigma(x_i - \bar{x})^2}{N} = 46/5 = 9.2$

Third central moment $= \mu_3 = \frac{\Sigma(x_i - \bar{x})^3}{N} = -18/5 = -3.6$

Fourth central moment $= u_4 = \frac{\Sigma(x_i - \bar{x})^4}{N} = 610/5 = 122$

Measure of skewness $= \beta_1 = \frac{\mu_3^2}{u_2^3} = \frac{-3 \cdot 6^2}{9.2^3} = 12.96/778.68 = 0.0166$

Coefficient of skewness $= \gamma_1 = \sqrt{\beta_1} = 0.1290$

Measure of kurtosis $= \beta_2 = \frac{\mu_4}{u_2^2} = \frac{122}{9 \cdot 2^2} = 122/84.64 = 1.4413$

Coefficient of kurtosis $= \gamma_2 = \beta_2 - 3 = 1.4413 - 3 = -1.5586$

Implementation in python:

In [44]:
```
import numpy as np
from scipy.stats import kurtosis

Data=[13,14,18,19,21]

print("Kurtosis of the Data is: ",kurtosis(Data))
```

```
Kurtosis of the Data is:  -1.5586011342155006
```

Naive implementation in python:

In [45]:
```
import math

def naive_mean(data):
    '''This function will calculate the mean of the data'''
    total_count=0
    sum=0
    for element in data:
        total_count=total_count+1
```

```
        sum=sum+element
    mean=sum/total_count
    return mean,total_count

def naive_central_moment(data):
    '''This function will calculate the skewness and kurtosis of the data'''
    mean,total_count=naive_mean(data)
    value1=0
    value2=0
    value3=0
    value4=0
    for element in data:
        value1=value1+(element-mean)
        value2=value2+((element-mean)**2)
        value3=value3+((element-mean)**3)
        value4=value4+((element-mean)**4)
    first_central_moment=value1/total_count
    second_central_moment=value2/total_count
    third_central_moment=value3/total_count
    fourth_central_moment=value4/total_count
    measure_of_skewness=((third_central_moment)**2)/((second_central_moment)**3)
    coefficient_of_skewness=math.sqrt(measure_of_skewness)
    measure_of_kurtosis=fourth_central_moment/(second_central_moment**2)
    coefficient_of_kurtosis=measure_of_kurtosis-3
    print("Skewness of the data is: ",coefficient_of_skewness)
    print("kurtosis of the data is: ",coefficient_of_kurtosis)

data=[13,14,18,19,21]
naive_central_moment(data)
```

```
Skewness of the data is:  0.12900922305569748
kurtosis of the data is:  -1.5586011342155006
```

Standard Normal Distribution:

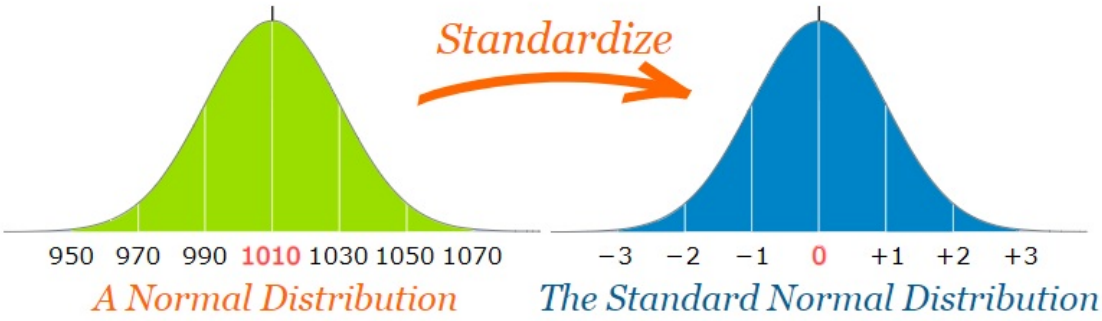
A standard normal distribution is a normal distribution with zero mean (μ=0) and standard deviation (σ=1)

So, to convert a normal distribution to Standard normal distrubution we have to use **Z-score standarization** technique:

Z-score standarization: This technique consists of subtracting the mean of the column from each value in a column, and then dividing the results by the standard deviation of the column. The formulae to achieve this is the following:

$$z = \frac{x - \mu}{\sigma}$$

The result of standarization is that the features will be rescaled. So, that they will have the properties of a standard normal distribution, as follows:
mean(μ)=0 and standard deviation(σ^2)=1



Application of standardization

Put the data on one sacle:

- Application 1: Let's try to understand why we have to perform "standardization" on the data To understand the importance of converting "Normal distribution" into "standard normal distribution". Let's suppose there are two student's Sachin and Ranjeet. Sachin scored 65 in

the math exam and ranjeet scored 80 in science exam.

Can we conclude that ranjeet scored better than sachin? No, because the way people performed in math exam may be different from science exam. The variability may not be the same here. So, a direct comparison by just looking at the scores will not work. Now let's say the math marks follow a normal distribution with mean 60 and std=4 on the other hand, the science marks follows a normal distribution with mean 79 and standard deviation 2. We will have to calculate the z score by:

Z score(math)=(65-60)/4=1.25

Z score(science)=(80-79)/2=0.5 The sachin scored 1.25 std above the mean score while ranjeet score only 0.5 std above the mean score . Hence we can say that sachin perform better than ranjeet.

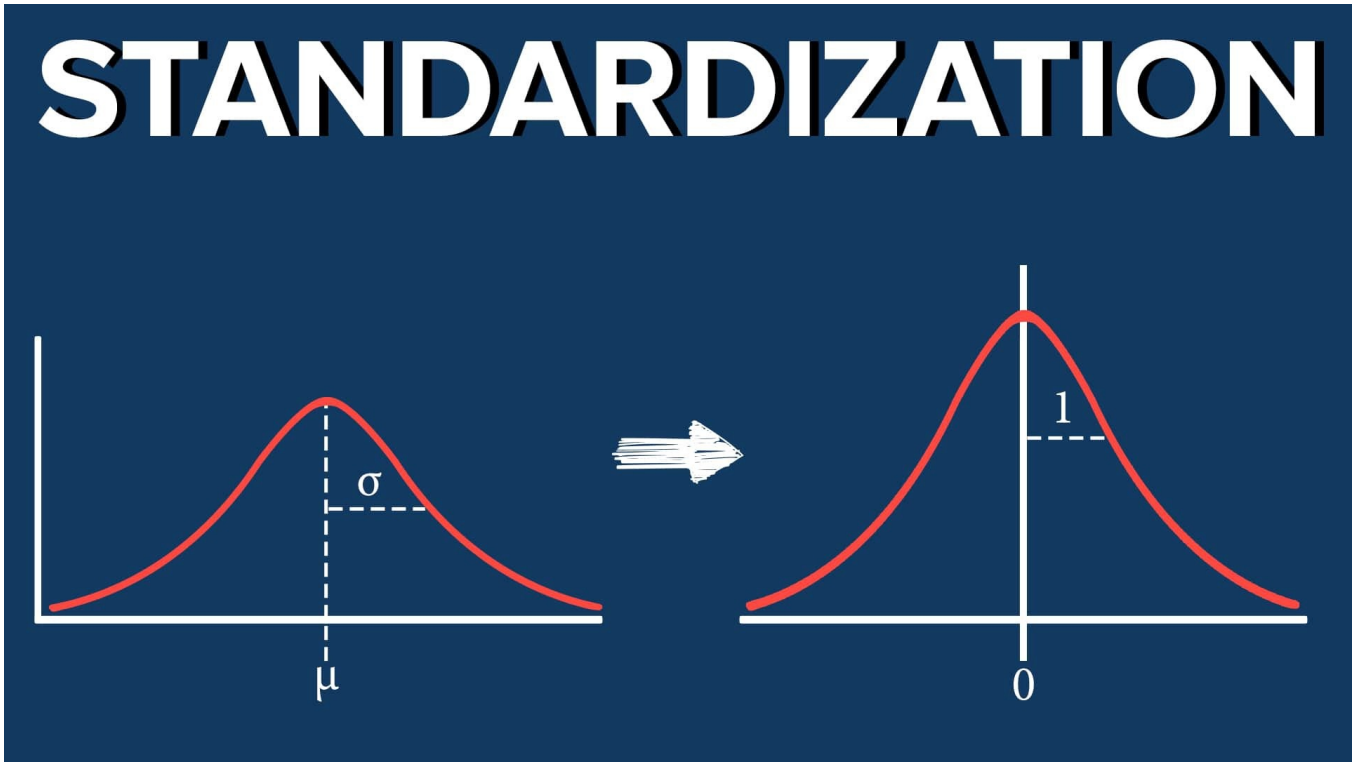
- Application 2: Few machine learning algorithm required to standardized the data to bring all the data on one scale which algorithm uses distance like KNN, SVM etc. So, we have to apply standardization on data before start modelling the algorithm. so no one feature dominate other and make machine leaning algorithm perform biasely.

Standard normal variate:

It is the distribution that occurs when a normal random variable has a **mean of 0(zero)** and **standard deviation of 1(one)**. The normal random variable of a standard normal distribution is called a stancard score or Z-score.

z~N(0,1)
This notation represent the random variable follow normal distribution with mean(μ) 0(zero) and standard deviation(σ) 1

Standarization: standarization is the process of transforming a variable with mean(μ) 0(zero) and standard deviation(σ) 1



Formulae:

$$x_{\text{stand}} = \frac{x - \text{mean}(x)}{\text{standard deviation }(x)}$$

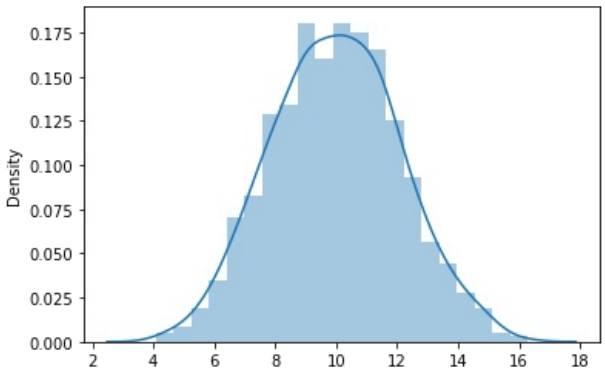
let's take a variable with normal distribution with any mean(μ) and standard deviation(σ).

Implementation in python:

In [46]:
```
import numpy as np
```

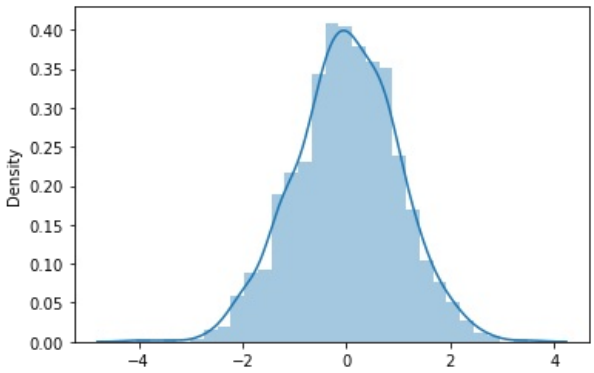
```
import matplotlib as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler

data=np.random.normal(10,2,1000)   #create a normal distributon with mean is 10 and standard deviation is 2
sns.distplot(data)
```

Out[46]: <AxesSubplot:ylabel='Density'>



In [47]:
```
data=np.random.normal(10,2,1000).reshape(-1,1)
scaler=StandardScaler()
datascaled=scaler.fit_transform(data)            #applying standarization on data
sns.distplot(datascaled)
```

Out[47]: <AxesSubplot:ylabel='Density'>



Explanation of above python code: Here you see i have created normal distribution with mean 10 and standard deviation 2 and after standardization the mean become 0 and standard deviation become 1.

In [48]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import warnings
warnings.filterwarnings('ignore')
'''downlaod iris.csv from https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/data/iris.csv'''
#Load Iris.csv into a pandas dataFrame.
iris = pd.read_csv("https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/data/iris.csv")
iris
```

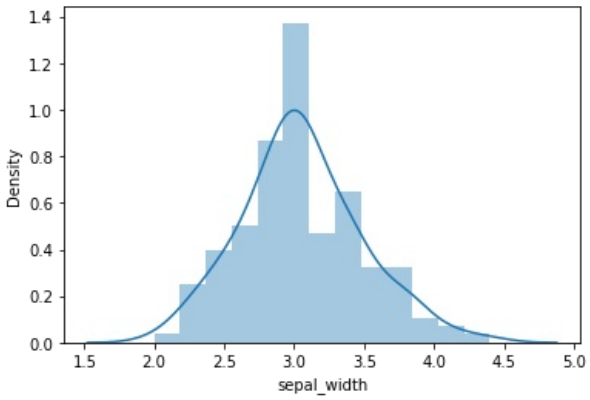
Out[48]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

150 rows × 5 columns

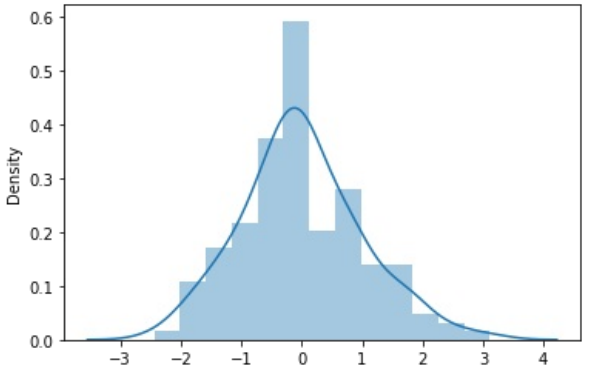
In [49]:
```
sns.distplot(iris['sepal_width'])
```

Out[49]: <AxesSubplot:xlabel='sepal_width', ylabel='Density'>



In [50]:
```
data=iris['sepal_width'].values.reshape(-1,1)
scaler=StandardScaler()
datascaled=scaler.fit_transform(data)            #applying standarization on data
sns.distplot(datascaled)
```

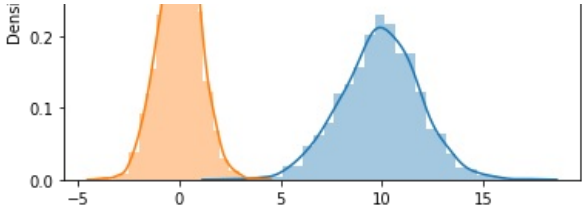
Out[50]: <AxesSubplot:ylabel='Density'>



Naive implementation in python:

In [51]:
```
import numpy as np
import matplotlib as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler

data=np.random.normal(10,2,1000)   #create a normal distributon with mean is 10 and standard deviation is 2

def naive_standard_scaler(data):
    '''This function standardized the data make mean=0 and standard deviation=1'''
    scaled_value=[]
    mean=statistics.mean(data)
    standard_deviation=statistics.stdev(data)
    for i in data:
        scaled=((i-mean)/standard_deviation)
        scaled_value.append(scaled)
    return scaled_value

scaled_value=naive_standard_scaler(data)
sns.distplot(data)
sns.distplot(scaled_value)
```

Out[51]: <AxesSubplot:ylabel='Density'>

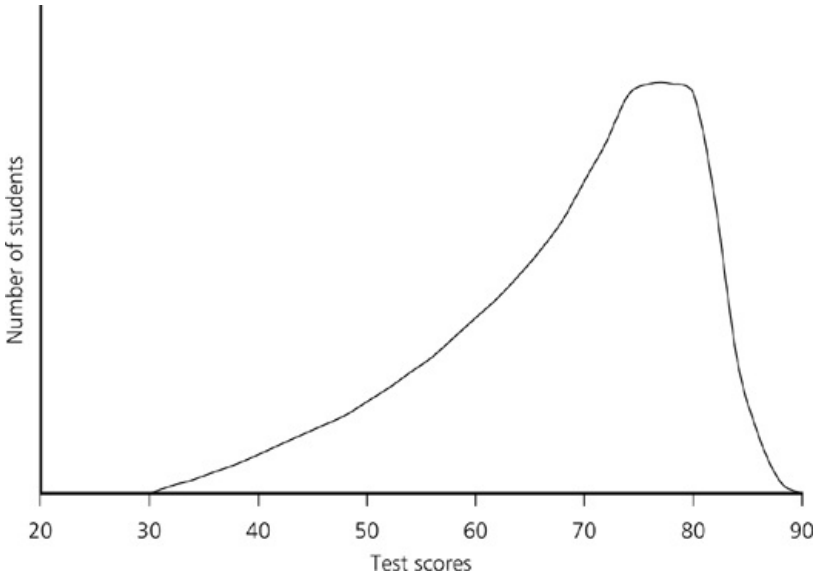


Central Limit Theorm(CLT):

Let us understand CLT therom which is one of the best and eligent theorm in the whole world of the statistics.
So, come to the theory first then we will understand via Real world problem and then we will see the implementaion in python.

Definition: Given a dataset with unknown distribution, the sampling distribution of sample mean(This might look confusing to you but hold on let me start diving the concept into smaller chunk then you will get to know. It's so simple) will be the **Normal distribution** Let us suppose **X** is a random variable follows any distribution. We do not know the distribution of **X** with mean(μ) and variance(σ^2)

X~Any_distribution(μ,σ^2)



Let's assume the sample size is 30 it can be anything.

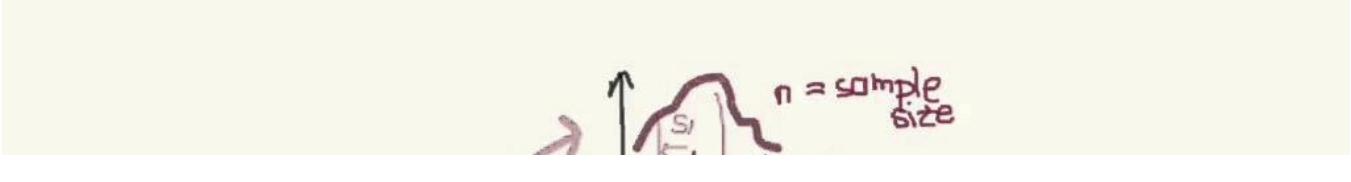
Take random sample of size (n=30)→s1→$\bar{x}1$
Take random sample of size (n=30)→s2→$\bar{x}2$
Take random sample of size (n=30)→s3→$\bar{x}3$
Take random sample of size (n=30)→s4→$\bar{x}4$
.
.
.
.
.
Take random sample of size (n=30)→sm→$\bar{x}m$

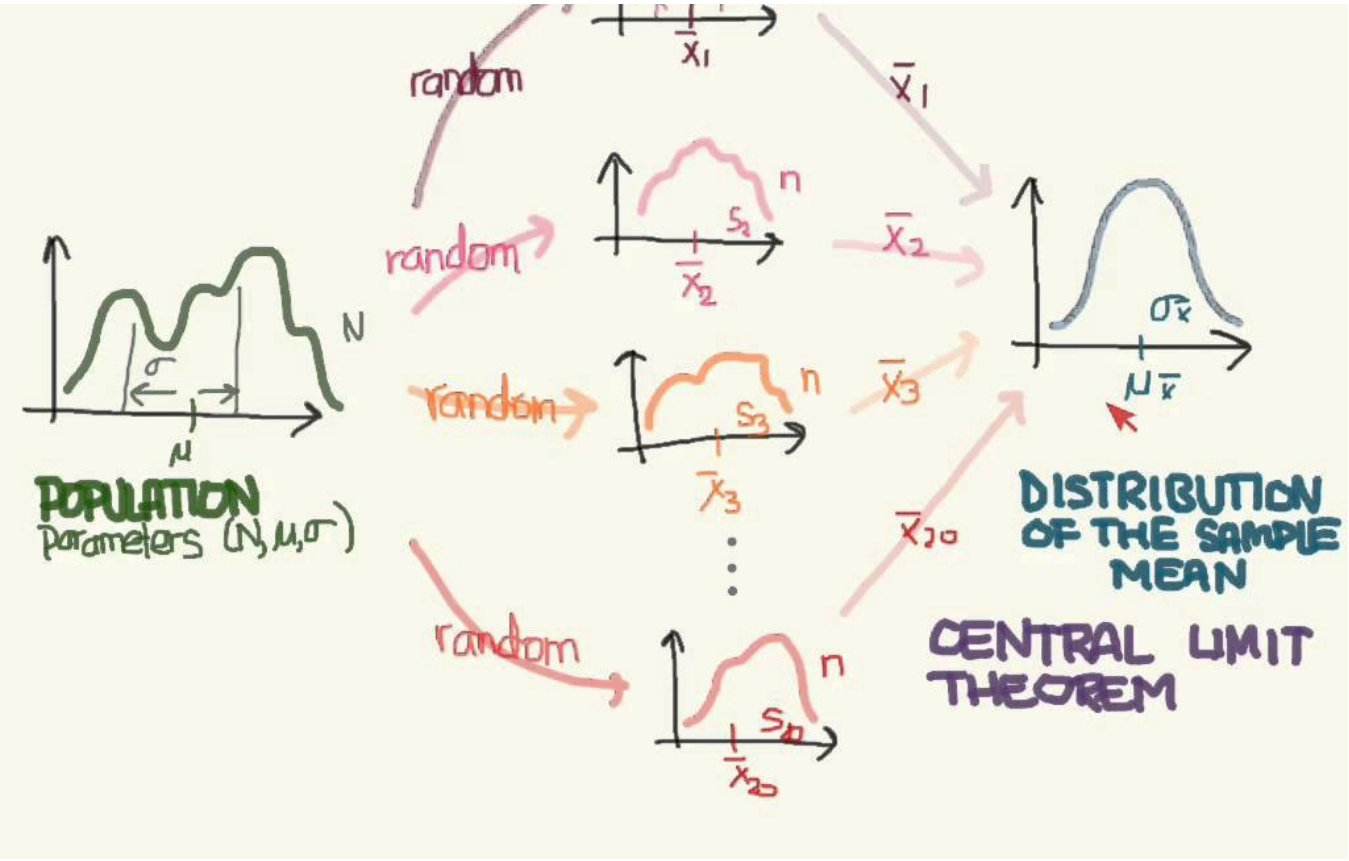
$\bar{x}1$, $\bar{x}2$, $\bar{x}3$, $\bar{x}4$, $\bar{x}5$, $\bar{x}6$,....., $\bar{x}m$

As your sample get bigger the sampling distribution will tend to look more and more like a normal distribution.

xi~N(μ,σ^2/N)

μ:This is same as population mean
σ^2: This is same as population variance
N:This is the same as sample size N





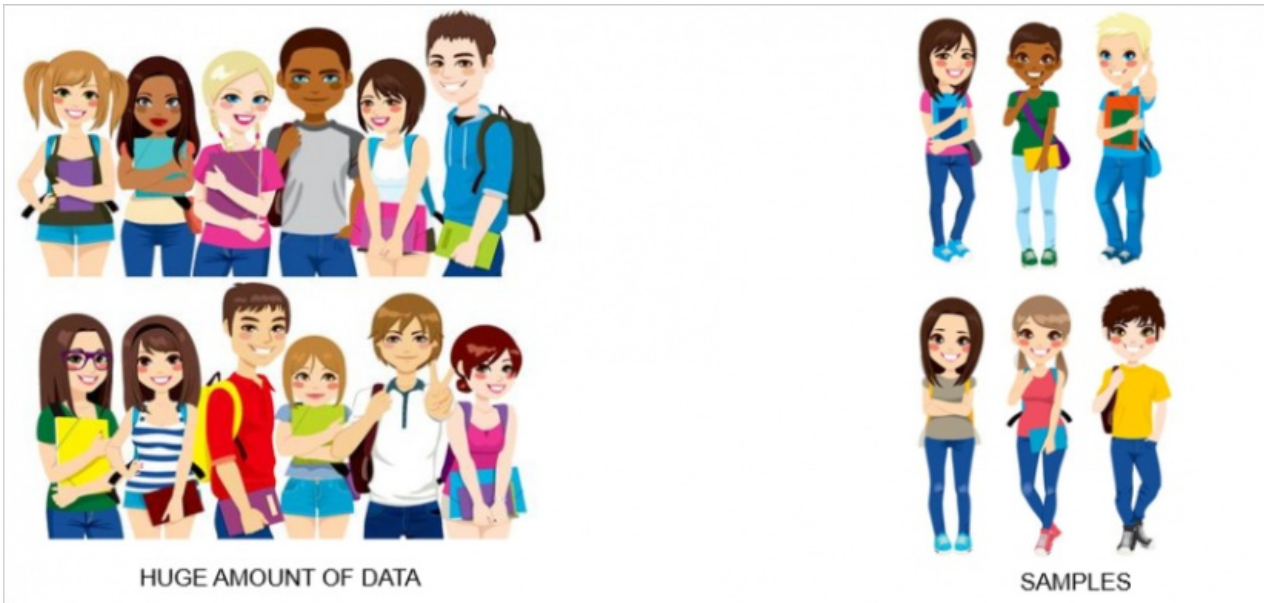
Real life scenario

Consider that there are 15 sections in the science department of a university and each section hosts around 100 students. Our task is to calculate the average weight of students in the science department. Sounds simple, right?
The approach I get from aspiring data scientists is to simply calculate the average:

- First, measure the weights of all the students in the science department
- Add all the weights
- Finally, divide the total sum of weights with a total number of students to get the average
-

But what if the size of the data is humongous? Does this approach make sense? Not really – measuring the weight of all the students will be a very tiresome and long process. So, what can we do instead? Let's look at an alternate approach.

- First, draw groups of students at random from the class. We will call this a sample. We'll draw multiple samples, each consisting of 30 students.



- Calculate the individual mean of these samples

- Calculate the mean of these sample means
- This value will give us the approximate mean weight of the students in the science department
- Additionally, the histogram of the sample mean weights of students will resemble a bell curve (or normal distribution)

Assumptions Behind the Central Limit Theorem

Before we dive into the implementation of the central limit theorem, it's important to understand the assumptions behind this technique:

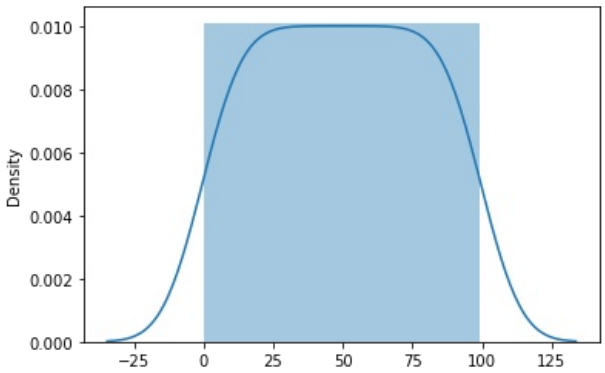
1. The data must follow the randomization condition. It must be sampled randomly
2. Samples should be independent of each other. One sample should not influence the other samples
3. Sample size should be not more than 10% of the population when sampling is done without replacement
4. The sample size should be sufficiently large. 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
5. Within a sample there will be no repetition but between sample there might be same point.

Naive implementation in python:

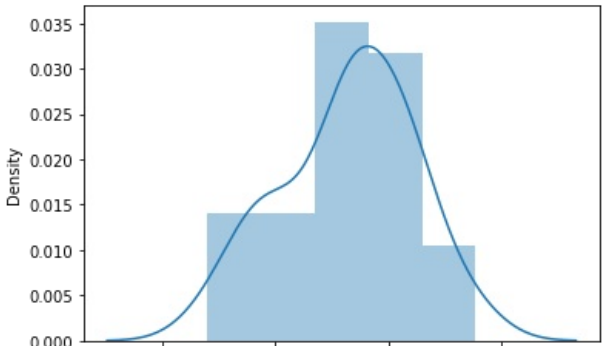
In [52]:
```
import random
import numpy as np
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

def naive_central_limit_theorm(data,sample_size):
    '''This function will perform central limit theorm on data'''
    sampling_distribution_of_sample_mean=[]
    sample_iteration=30

    for i in range(sample_iteration):
        sample=[]
        for j in range(sample_size):
            choice=random.choice(data)
            if choice not in sample:
                sample.append(choice)
            else:
                j=j-1
        mean=np.array(sample).mean()
        sampling_distribution_of_sample_mean.append(mean)
    sns.distplot(sampling_distribution_of_sample_mean)
data=range(100)
```

In [53]:
```
sns.distplot(data)
```

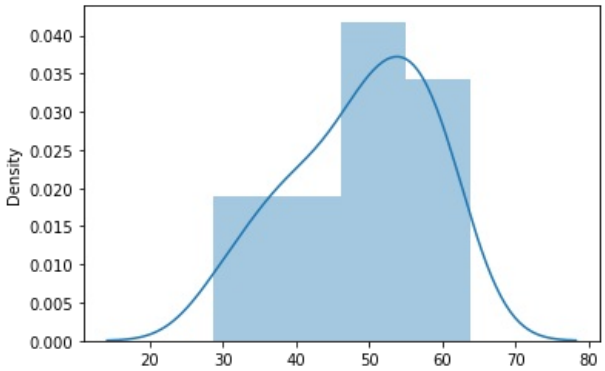
Out[53]:
```
<AxesSubplot:ylabel='Density'>
```



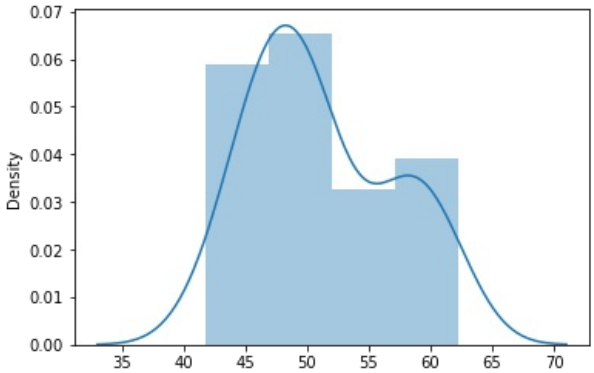
In [54]:
```
naive_central_limit_theorm(data,5)
```



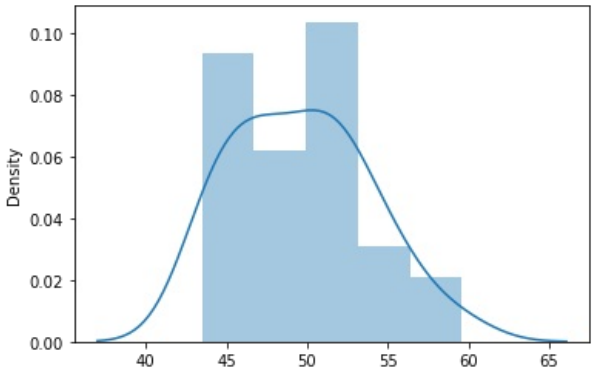
In [55]:
```
naive_central_limit_theorm(data,10)
```



In [56]:
```
naive_central_limit_theorm(data,20)
```



In [57]:
```
naive_central_limit_theorm(data,30)
```



Code Explanation: Here you can notice in first draw we took only 5 sample points then 10 then 20 and after that 30 and as we are increasing the sample size the distribution will more and more tends towards the normal distribution.

Quantile-Quantile(Q-Q) plot:

Q-Q plot are the graphical representation to measure that two sets of data come from the same distribution. In Q-Q plot we graphically analyze and compare two probablity distribution by plotting their quantiles each other. If the two distribution which we are comparing are exactly equal then the points on the Q-Q plot will perfectly lie on the same line.

How does it works:

Let us we have a random variable **X**.

X=x1 , x2 , x3 , x4 , x5 , x6 , x7 , x8 , ... , x500

Step 1 X=x1 , x2 , x3 , x4 , x5 , x6 , x7 , ... , x500
Sort all the xi's in ascending order

x1' , x2' , x3' , x4' , x5' , x6' , ... , x500'

Computer the percentile:

so, we have 500 variables
1st percentile: x5' : x(1)
2nd percentile: x10' : x(2)
3rd percentile: x15' : x(3)
4th percentile: x20' : x(4)
5th percentile: x25' : x(5)
.
.
.
.
.
.
.
100th percentile: x500' : x(100)

Step 2 Create a variable Y~N knows as the theoretical quantile
Sort all the yi's in ascending order

y1' , y2' , y3' , y4' , y5' , y6' , ... , y500'

Computer the percentile:

so, we have 500 variables
1st percentile: y5' : x(1)
2nd percentile: y10' : y(2)
3rd percentile: y15' : y(3)
4th percentile: y20' : y(4)
5th percentile: y25' : y(5)
.
.
.
.
.
.
.
100th percentile: y100' : y(100)

Step 3 Plot the Q-Q plot using all the percentile:
x(1),x(2),x(3),x(4),x(5),x(6),.....,x(100)
y(1),y(2),y(3),y(4),y(5),y(6),.....,y(100)

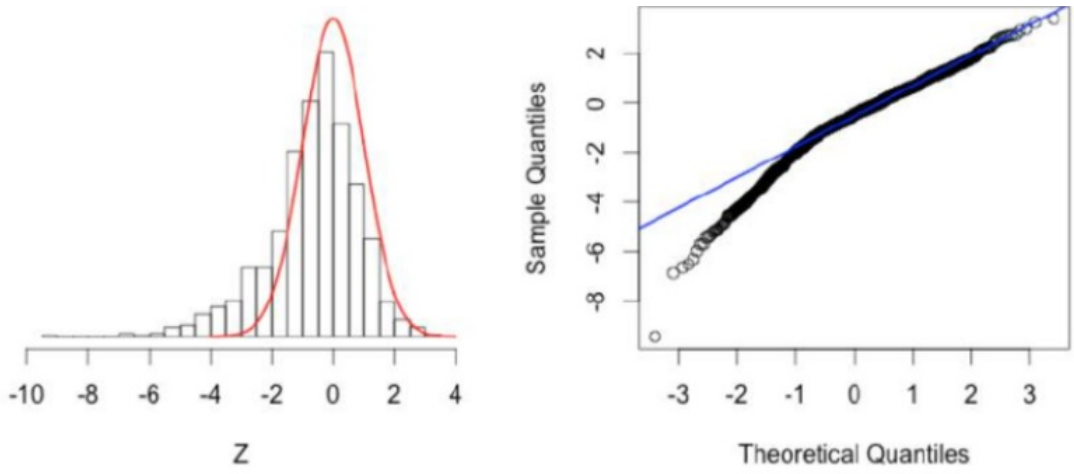
we will have 100 pair of (xn,yn)

Now we have to focus on the ends of the straight line. If the points at the ends of the curve formed should fall on the straight line.
If all the points plotted on the graph perfectly lies on a straight line then we can clearly say that this distribution is Normal distribution.

Skewed Q-Q plot

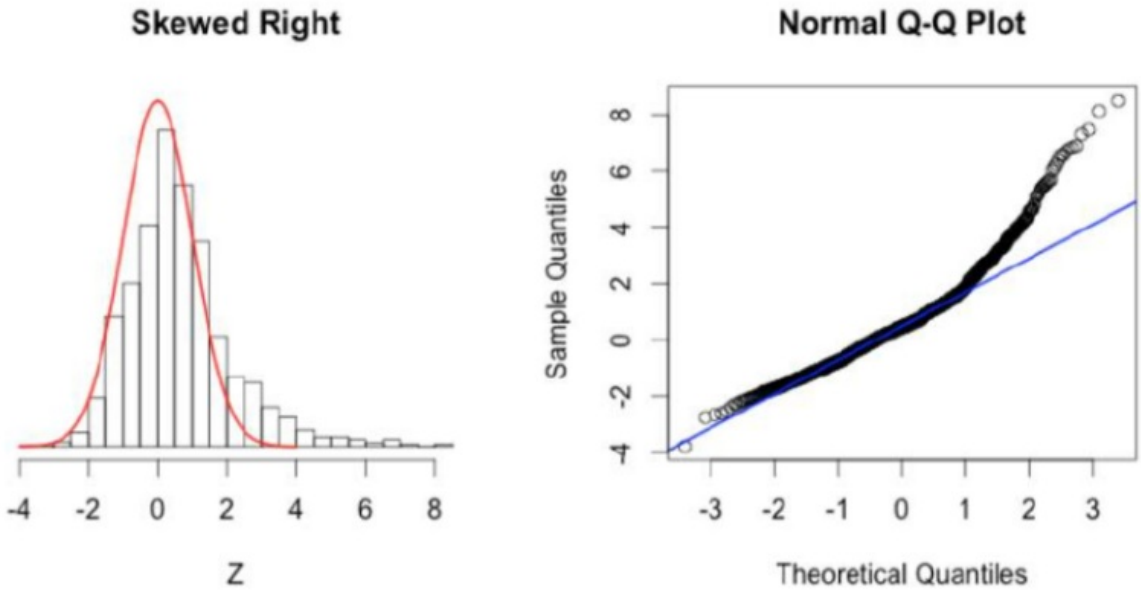
Q-Q plots are also used to find the skewnessof a distribution. When we plot theoratical quantiles on x - axis and the sample quantile whose distributio. We want to know on the Y-axis then we see a very pecular shape of a Normally distributed Q-Q plot for skewness.

- If the bottom end of the Q-Q plot deviates from the straight line but the upper end is not, then we can clearly say that the distribution has a longer tail to its left or simply it is left skewed.



Left Skewed Q-Q plot for Normal DIstribution

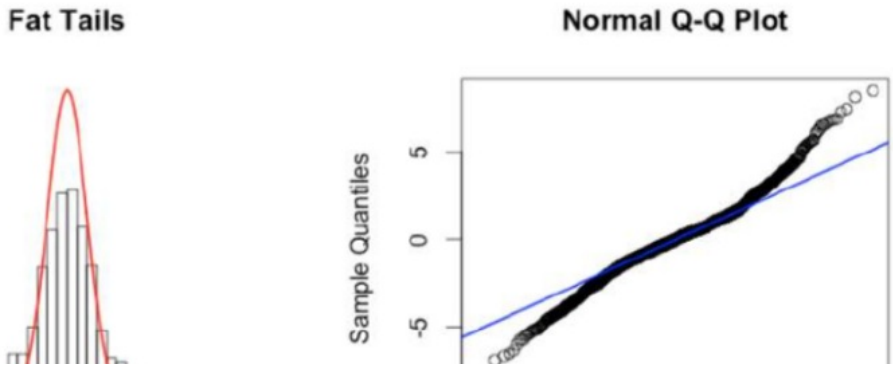
- But when we see the upper end of the Q-Q plot to deviate from the straight line then the curve has a longer tail to its right and it is right-skewed/Positively skewed.

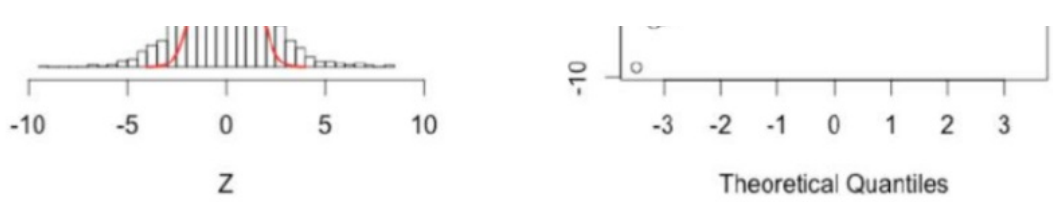


Right Skewed Q-Q plot for Normal DIstribution

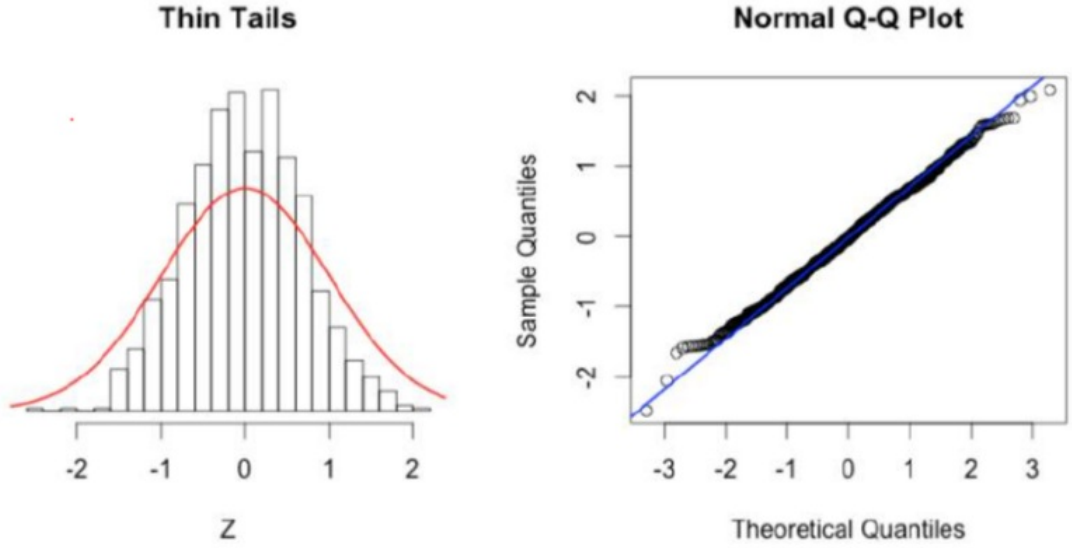
Tailed Q-Q plot

Similarly, we can talk about the kurtosis of the distribution by simply looking at its Q-Q plot. The distribution with a fat tail will have both the end of the Q-Q plot to deviate from the straight line and its center follows a straight line, whereas a thin tailed distribution will form a Q-Q plot with a very less or negotiable deviaions at the end thus making it a perfect fir for the Normal distribution.





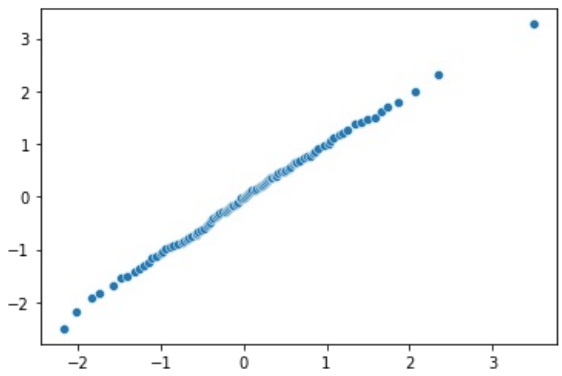
Fat-Tailed Q-Q plot for Normal DIstribution



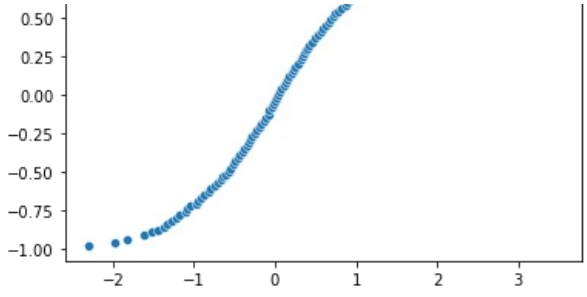
Thin-Tailed Q-Q plot for Normal DIstribution

Naive implementation in python

In [58]:
```
import numpy as np
import seaborn as sns

def naive_QQ_plot(first_distribution,second_distribution):
    '''This function will plot QQ plot on any two distribution'''
    percentile1=[]
    percentile2=[]

    for i in range(1,101):
        percentile1.append(np.percentile(first_distribution,i))
        percentile2.append(np.percentile(second_distribution,i))
    sns.scatterplot(percentile1,percentile2)

first_distribution=np.random.normal(loc=0,scale=1,size=1000)
second_distribution=np.random.normal(loc=0,scale=1,size=1000)
naive_QQ_plot(first_distribution,second_distribution)
```

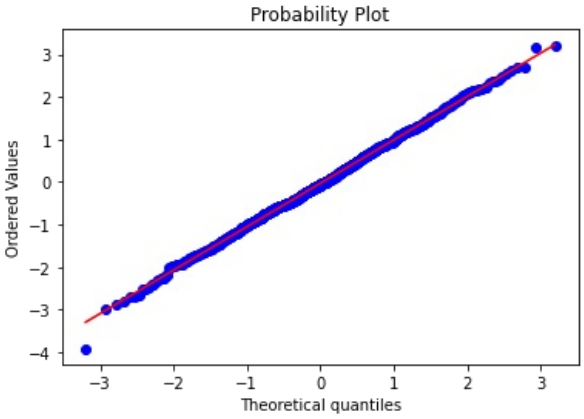


In [59]:
```
first_distribution=np.random.normal(loc=0,scale=1,size=1000)
second_distribution=np.random.uniform(low=-1, high=1, size=10000)
naive_QQ_plot(first_distribution,second_distribution)
```

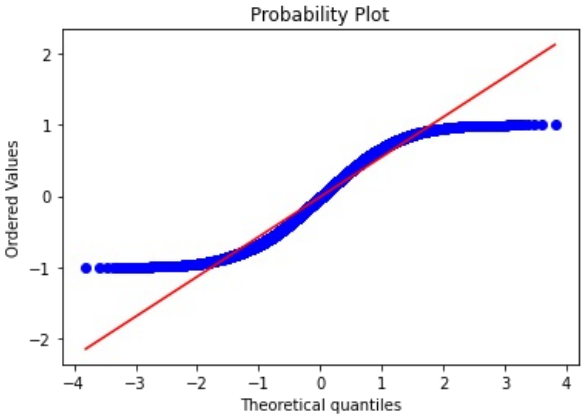


Implementation in python

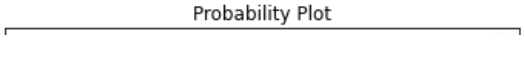
In [60]:
```
import numpy as np
import pylab
import scipy.stats as stats

std_normal=np.random.normal(loc=0,scale=1,size=1000)     #generate 1000 sample from normal distribution with samp
stats.probplot(std_normal,dist="norm",plot=pylab)
pylab.show()
```

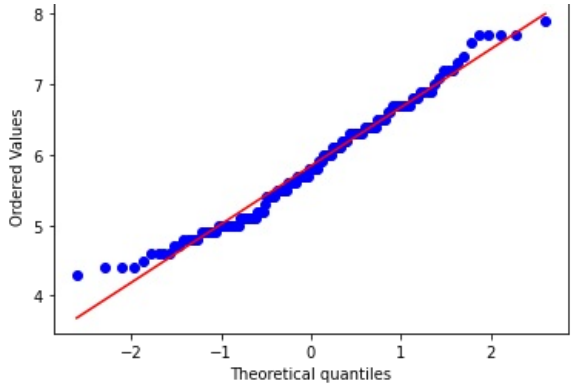


In [61]:
```
# generate 10000 sanples from N(20,5)
measurements = np.random.uniform(low=-1, high=1, size=10000)
stats.probplot(measurements, dist="norm", plot=pylab)
pylab.show()
```



In [62]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import pylab
import scipy.stats as stats
import warnings
warnings.filterwarnings('ignore')
'''downlaod iris.csv from https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/data/iris.csv'''
#Load Iris.csv into a pandas dataFrame.
iris = pd.read_csv("https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/data/iris.csv")
# generate 100 sanples from N(20,5)
measurements = iris['sepal_length']

stats.probplot(measurements, dist="norm", plot=pylab)
pylab.show()
```





Chebyshev's inequality:

We are familier with the Normal distribution concept and we know that if any random variable follow Normal distribution so we can make inference that within first standard deviation **[μ-σ<=x<=μ+σ] 68%** distribution lie and within second standard deviation **[μ-2σ<=x<=μ+2σ] 95%** distribution lie and within third distribution **[μ-3σ<=x<=μ+3σ] 99.7%** distribution lie.
But what if we don not know the distribution so how can i make inference about how much data lie in region.

Chebyshev's inequality is a theory describing the maximum number of extreme values in a probability distribution. It states that no more than a certain percentage of values (1/k^2) will be beyond a given distance (k standard deviations) from the distribution's average. The theorem is particularly useful because it can be applied to any probability distribution, even ones that violate normality, so long as the mean and variance are known. The equation is:

$$Pr(|X - \mu| \geq k \times \sigma) \leq \frac{1}{k^2}$$

In plain English, the formula says that the probability of a value (X) being more than k standard deviations (σ) away from the mean (μ) is less than or equal to 1/k^2. In that formula,k represents a distance away from the mean expressed in standard deviation units, where k can take any above-zero value.

As an example, let's calculate the probability for k=2.

$$Pr(|X - \mu| \geq k) \leq \frac{\sigma^2}{k^2}$$

Although, the inequality is valid for any real number k>0, only the case k>1 is practically useful.

As an example, let's calculate the probability for k=2.

$$Pr(|X - \mu| \geq 2 \times \sigma) \leq \frac{1}{4}$$

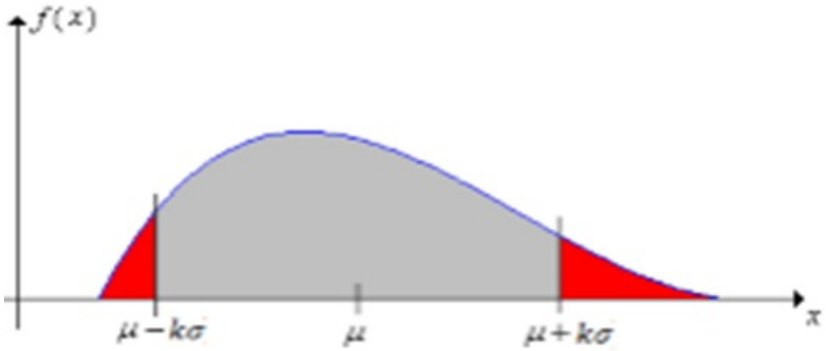
The answer is that no more than ¼ of values will be more than 2 standard deviations away from the distribution's mean. Note that this is a loose upper bound, but it is valid for any distribution.

Finding the reverse: Number of values within a range Chebyshev's inequality can also be used to find the reverse information. Knowing the percentage of values outside a given range also by definition communicates the percentage of values inside that range. percentage inside = 1 - percentage outside. In other words, the maximum number of values within k standard deviations of the mean will be 1 - 1/k^2.

In the previous example, we calculated that no more 1/4 of values will be more than 2 standard deviations away from the distribution's mean. To find the reverse, we calculate 1 - 1/k^2 = 1 - 1/4 = 3/4. In other words, no more than 3/4 of values will be within 2 standard deviations from the distribution's mean.

Condition on chebyshevs inequality

1. We have a finite mean(μ)
2. We have standard deviation (σ) non-zero and finite.



Formulae:

$$\text{Chebyshev's inequality}$$

$$\mathbb{P}(|X - \mu| \geq k\sigma) \leq \frac{1}{k^2}$$

Where X is a random variable, μ is an expected value of X, σ is a standard deviation of X and k > 0.

Mathematically solved: Let's see how to solve the chebyshev's inequality problem.

Qus: If we know the salary of individuals? and don't know the distribution only known mean=20k and standard deviation=10k. and somebody ask me what percentage of individuals have a salary in the range of [20k,60k]

Ans: P[20k,60k] = ?

mean=40k standard deviation=10k
Using Chebyshev's Inequality we can write the following probability:

P[40-2 *10* <= *X* <= *40*+2 10] > 1-1/2^2

```
> 1-1/4
> 3/4
```

In other words, chances of a person coming up inside salary range range of 20k to 60k times is at least 0.75.

In [63]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

def chebyshev_inequality(k):
    return 1 / k**2

k = np.arange(1,10)
p_cheb = np.zeros(len(k))

print('Chebyshev_Inequality for different k values:\n')

for i in range(len(k)):
    p_cheb[i] = chebyshev_inequality(k[i])
    print(f'Pr( |μ−σ| >= {k[i]}σ ) <= {p_cheb[i]}' )
```

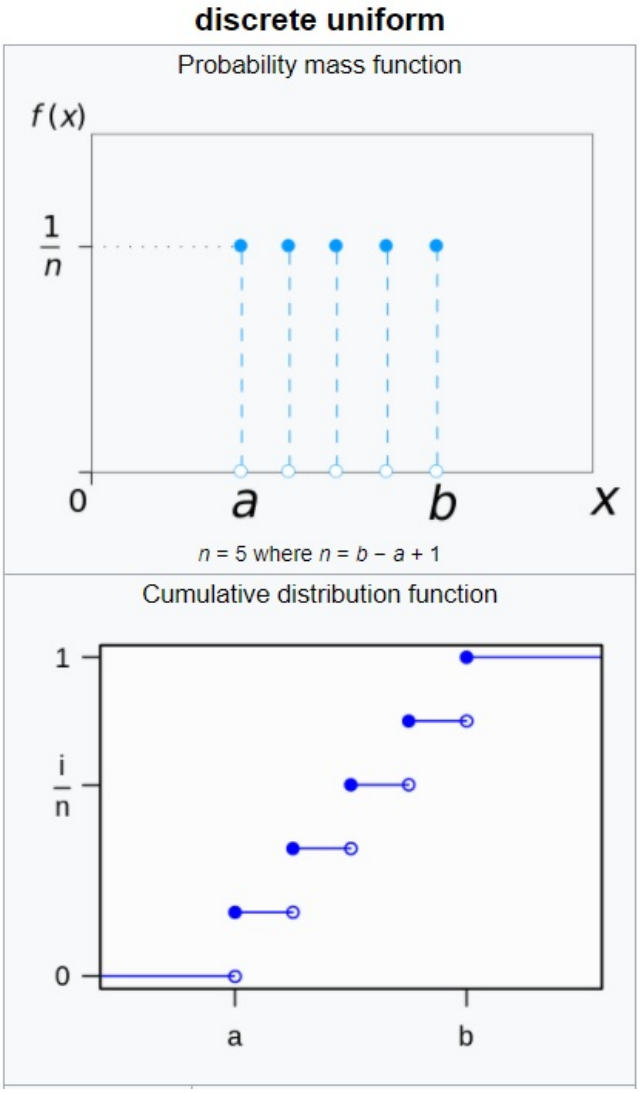
```
Chebyshev_Inequality for different k values:

Pr( |μ−σ| >= 1σ ) <= 1.0
Pr( |μ−σ| >= 2σ ) <= 0.25
Pr( |μ−σ| >= 3σ ) <= 0.1111111111111111
Pr( |μ−σ| >= 4σ ) <= 0.0625
```

```
Pr( |μ−σ| >= 5σ ) <= 0.04
Pr( |μ−σ| >= 6σ ) <= 0.027777777777777776
Pr( |μ−σ| >= 7σ ) <= 0.02040816326530612
Pr( |μ−σ| >= 8σ ) <= 0.015625
Pr( |μ−σ| >= 9σ ) <= 0.012345679012345678
```

Uniform Distribution:

- Perhaps one of the simplest and useful distribution is the uniform distribution uniform distribution is a distribution in which there equal probablities across all the values in the set.
- A uniform distribution, sometimes also known as a rectangular distribution, is a distribution that has a constant probablities
- The simplest probability distribution is the uniform distribution, which gives the same probability to any points of a set
   A uniform distribution is of two types:
  1. Discrete uniform distribution
  2. Continous uniform distribution

Discrete uniform distribution: The discrete uniform distribution is a symmetric probablities distribution. Where in a finite number of values are equally likely to be observed . Every one of n value has equal probablity 1/n.



Notation	$\mathcal{U}\{a, b\}$ or $\operatorname{unif}\{a, b\}$
Parameters	a, b integers with $b \geq a$ $n = b - a + 1$
Support	$k \in \{a, a+1, \ldots, b-1, b\}$
PMF	$\frac{1}{n}$
CDF	$\lfloor k \rfloor - a + 1$

	$\frac{}{n}$
Mean	$\frac{a+b}{2}$
Median	$\frac{a+b}{2}$
Mode	N/A
Variance	$\frac{(b-a+1)^2-1}{12}$
Skewness	0

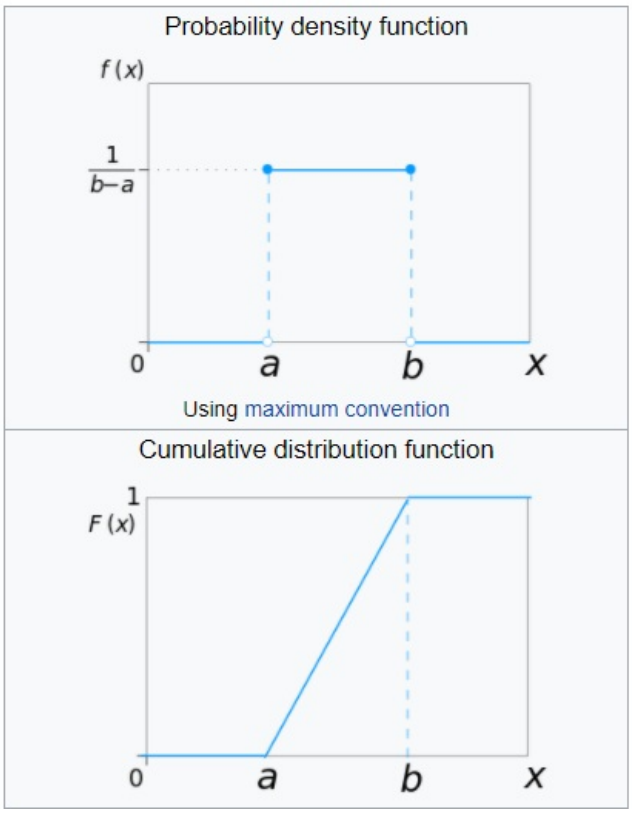
Example: Throwing a fair dice

So, the possible values are 1,2,3,4,5 and 6 and each time the die is thrown the probablity of a given score is 1/6.

p(X=1)=p(X=2)=p(X=3)=p(X=4)=p(X=5)=p(X=6)=1/6

Continous uniform distribution Describes an experiment where there is an arbitary outcome that lies between certain bounds are defined by the parameters, a and b, which are minimum and maximum values. The intervals can be either closed(e.g., [a,b]) or open (e.g.,(a,b)). The difference between the bounds define the interval length, all interval of same length on the distribution supports are equally probable.

Uniform



Notation	$\mathcal{U}(a,b)$ or $\text{unif}(a,b)$
Parameters	$-\infty < a < b < \infty$
Support	$x \in [a,b]$
PDF	$\begin{cases} \frac{1}{b-a} & \text{for } x \in [a,b] \\ 0 & \text{otherwise} \end{cases}$
CDF	$\begin{cases} 0 & \text{for } x < a \\ \frac{x-a}{b-a} & \text{for } x \in [a,b] \\ 1 & \text{for } x > b \end{cases}$
Mean	$\frac{1}{2}(a+b)$
Median	$\frac{1}{2}(a+b)$
Mode	any value in (a,b)
Variance	$\frac{1}{12}(b-a)^2$

Skewness	0

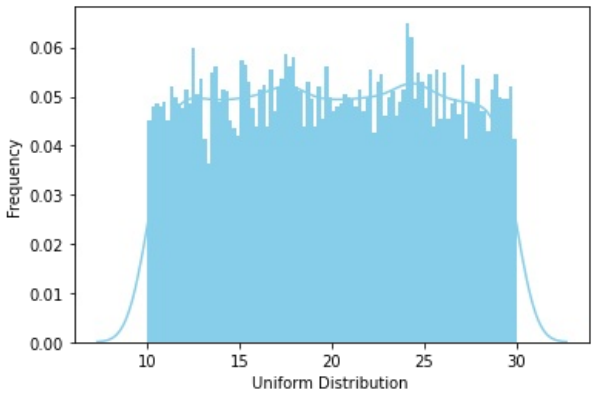
Example: Random number generator

So the every variable in between the interval has equal chance of happening

Implementation in python:

In [64]:
```
# import uniform distribution
from scipy.stats import uniform

data_uniform = uniform.rvs(size=10000, loc = 10, scale=20)

ax = sns.distplot(data_uniform,bins=100,kde=True,color='skyblue',hist_kws={"linewidth": 15,'alpha':1})
ax.set(xlabel='Uniform Distribution ', ylabel='Frequency')
```

Out[64]:
```
[Text(0.5, 0, 'Uniform Distribution '), Text(0, 0.5, 'Frequency')]
```



Bernoulli distribution:

The Bernoulli distribution is the discrete probablity distribution of a random variable which takes a binary, boolean outout

- 1 with probablity p (Success)
- 0 with probablity 1-p (Failure)

The idea is that, whenever you are running an experiment which might lead either to a success or to a failure. You can associate with your success(labelled with 1) a probablity with p, while your failure (labelled with 0) will have probablity (1-p)

Formulae

$$P(n) = \begin{cases} 1-p & \text{for } n = 0 \\ p & \text{for } n = 1, \end{cases}$$

The probablity of the success p is the parameter of the bernoulli distribution, and if a discrete random variable X follows that distribution, we write:

X~BE(p)

Example: Imagine your experiment consists of flipping a coin and you will win if the output is tail. Since the coin is fair, you know that the probablity of having "tail" is p=1/2. Hence, once set "tail=1" and "head=0". You can compute the probablity of success as follows:

p(X=1)=f(1)=p=1/2

Example:Imagine you are about to toss a dice, and you bet your money on number 1: hence number 1 will be your success (labelled with 1), while any other number will be a failure(labelled with 0). The probablity of success is 1/6. If you want to compute the probablity of failure. You will do like this:

p(X=0)=f(0)=1-p=5/6

Binomial Distribution:

This distribution describes the behaviour of the output of "n" random experiments, each having a Bernoulli distribution with probablity P.

Example: flipping a fair coin. We said that our experiment consisted of flipping that coin once. Let us now modify it a bit and say that we are going to flip that coin 5 times. Among these trials, we will have some successes(tail,labelled as 1) and some failure(head, labelled as 0). Each trial has probablity 1/2 of success and 1/2 of failure. We might be interested in knowing what is the probablity of obtaining a given number X of successes. How shall we proceed?

Let us visualize this experiment:

```
H  H  H  T  T
```

So, we flipped our coin 5 times and we lost and we lost in the first three trails, while we won in the last 2. Since we said that successes=tail=1 and failure=head=0, we can reframe it as follows:

```
0  0  0  1  1
```

Now, every trail is a bernoulli random variable, hence its probablity of occurence is P. If it is equal to 1, otherwise it is 0.Hence, if we want to compute the probablity of having the above situations(3 failure and 2 success). we will have something like that:

0(1-p) 0(1-p) 0(1-p) 1(p) 1(p)

and since trials are independent among each other:

```
p^2(1-p)^3
```

Generalizing this reasoning, if we had n trials with x successes:

0 0 0 0 0 0 0 . . . . . .0(n-x)times
1 1 1 1 1 1 1 1. . . . . .1(x times)

1-p 1-p 1-p 1-p 1-p 1-p. . . . . 1-p(n-x)times
p p p p p p p p. . . . . p(x)times

```
p^x(1-p)^n-x
```

Now a further concept need to be introduced. Indeed, so far we computed the probablity of having 2 successes exactly in the order shown above. Neverthless since we are interested in having a given number of success regardless of the order they are given to us, we need to take into account all the possible combination of having X successes

Namely, imagine we flip a coin 3 times and we want to compute the probablity of having 1 tail out of 3 trials. Hence, we will win in one of the following scenarious:

H H T
H T H
T H H
H T T
T H T
T T H

As you can see, there are three different combinations of outcomes which lead to a success. How can we incorporate this notion in our probablity function? The answer is the binomial coefficient is given by:

C(n,x)=n!/(x!(n-x)!)

Where n is the number of trials and X is the number of success of which we want to know the probablity of occurence.
So when we run n independent experiments, each having a bernoulli distribution with parameter p, and we want to know the probablity of having X success.

The probablity function will be:

$$P_n(x) = C(n,x)p^x q^{n-x}$$
$$= \frac{n!}{x!\,(n-x)!}p^x q^{n-x}$$

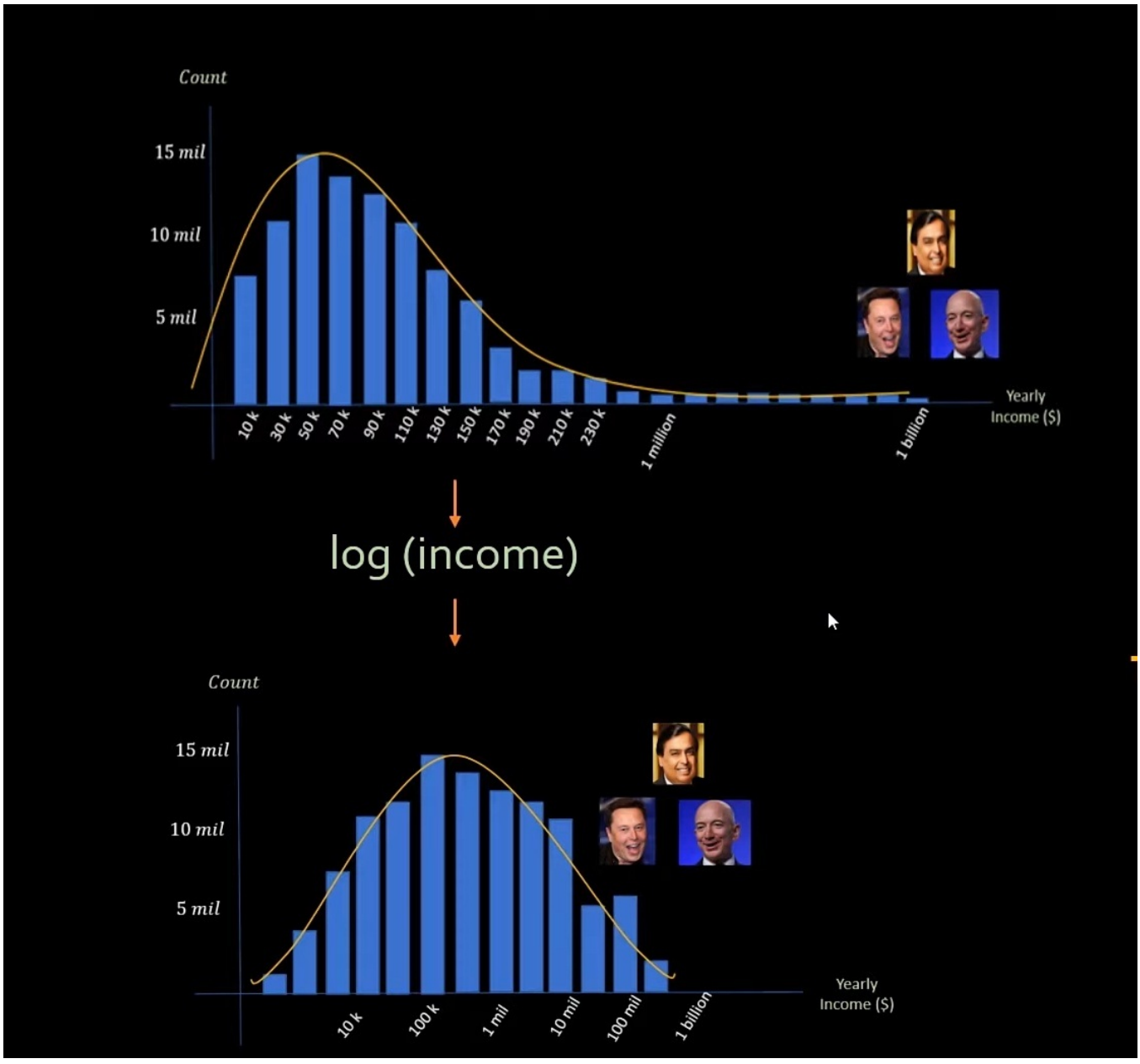
Log Normal Distribution:

The log Normal distribution is a continous probablity distribution of a random variable whose logarithm is normally distributed.

In simpler words, if X is a log Normal distribution

X~lognormal(μ,σ)
then y=ln(X) has a Normal distribution

If Y has a normal distribution, then the exponential function of Y
X=exp(Y)
has a Log Normal Distribution



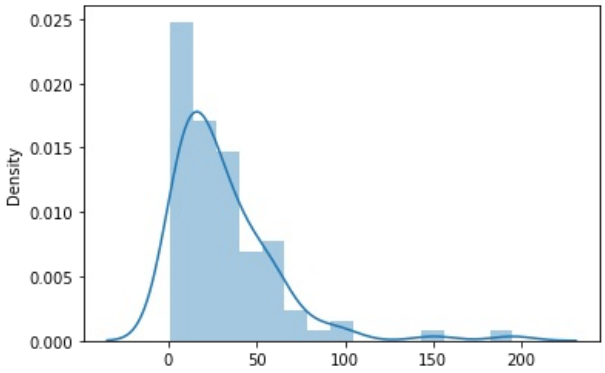
Occurence and application

- The length of comments posted in internet discussion forums.
- The user's dwell time on the article

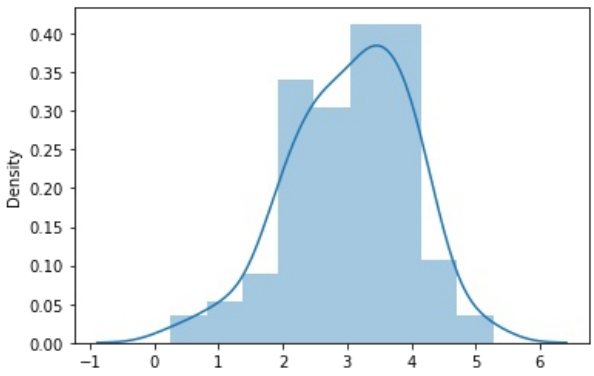
Implementation in python

In [65]:
```
import numpy as np
import seaborn as sns
import math

s=np.random.lognormal(3,1,100)
sns.distplot(s)
```

Out[65]:
```
<AxesSubplot:ylabel='Density'>
```

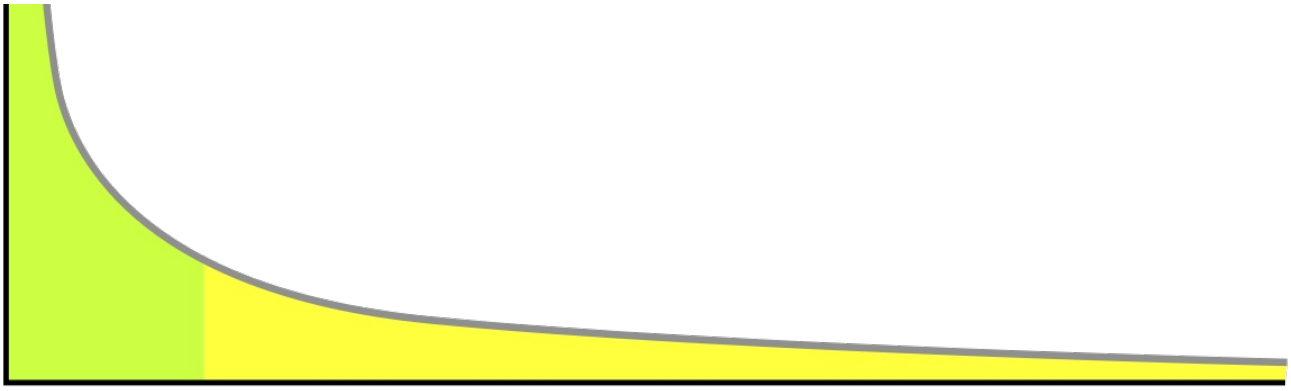


In [66]:
```
def log_to_normal_conversion(data):
    '''This function convert log normal distribution to normal distribution'''
    converted_values=[]
    for value in data:
        new_value=math.log(value)
        converted_values.append(new_value)
    sns.distplot(converted_values)

log_to_normal_conversion(s)
```



Power Law:

A power law is a functional relationship between two quantiles, where a relative change in one quantile result in a proportional relative change in other quantile, independent of initial size of those quantities. One quantities varies as a power of another.

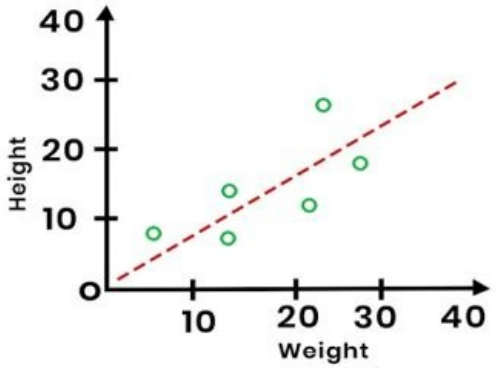


Power law follows 80-20 rule means the 80% of distribution found in first 20% of range and 20% of distribution found in last 80% of range

Co-variance:

We have already understand the concept of variance in starting of this notebook. Variance is a statistical measurement of the spread between numbers in a data set or we can say variance shows the relationship with itself. Co-variance term is defined from the variance term co-variance is the relationship between two variables. Let's understand what is co-variance.

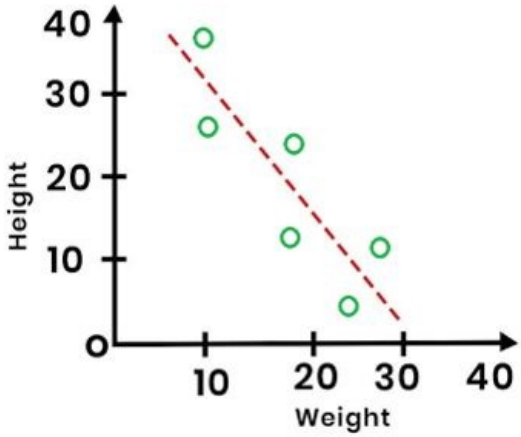
Here in the below plot you can see the relationship between height and weight.



Here, you can see the values of which have low height value generally have low weight value and who have high height value generally have high weight value.

This relationship low measurement for both height and weight for some person and high measurement for both height and weight for some person, can be summarized with the red line.

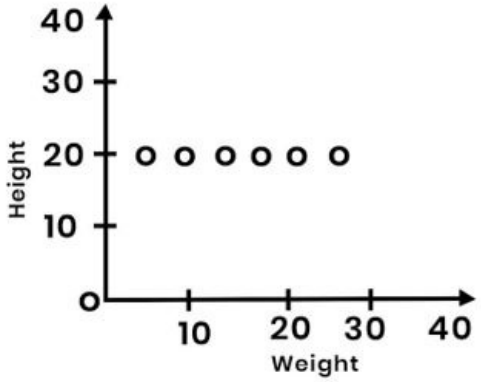
This line represent positive slope/positive trend.



Here, you can see the values of which have low height value generally have high weight value and who have high height value generally have low weight value.

This relationship can be summarized with the red line.

This line represent negative slope/negative trend.



Here, you can see every value of weight is paired with same value of height. and in another graph you can see every value of weight is paired with same value of height. So, there is no trend in data.

This is because we would not know if the same person should have a small value of height or relatively large or another value.

So, we can classified covariance into three relationship.

1. positive co-variance
2. Negative co-variance
3. No trend

Co-variance of positive relationship above is 116 is +ve, it means that the slope of the relationship between height and weight is +ve. In other words, when the co-variance value is +ve. We classify the trend as +ve.

Note: The co-variance value itself is not very easy to interpret and depends on the context.

For e.g., the covariance value doesn't tell us if the slope of the line representing the relationship is steap or not. More importantly the covariance value doesn't tell us if the points are relatively close to the dotted line or relatively far from the dotted line. It just tell us the relationship is +ve and same for the -ve slope.

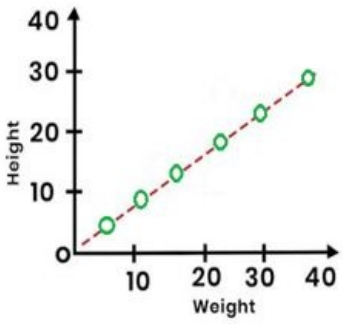
Now let's talk about why covariance value is hard to interpret.

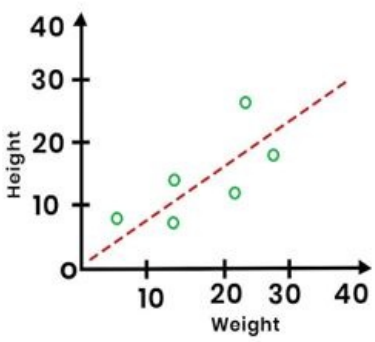
co-variance between height and height is co-variance(height,height)=variance(height)

now just multiply the data by 2. So, the relative trend doesn't change. In other word the only thing that changed was the scale that data is on.

But, we got the variance as 408, which is 4 times what we got before . Thus, we see that the co-variance value change even the relationship doesn't.

- In other words, co-variance value are sensitive to the scale of the data, and this makes them difficult to interpret.
- The sensitivity to scale also prevents the covariance value from telling us if the data are closed to the data dotted line that represent the relationship or far from it.





In this example, the covariance on the left, when each point is on the dotted line is 102, and the covariance on the right, when the data are relatively far from the dotted line is 381. So, in the right side case, when the data are far from the line, the co-variance is larger. Now just change the scale on the right hand side.

- covariance is used to measure relationship between two variables.
- covariance is a measure of how much two random variable vary together. It is similar to variance, but variance tells you how a single variable varies, and co-variance tells you how two variable vary together.
- Covariance is only dependent upon sign
- The covariance is the product of unit of two variables.
- The value of covariance lies between -∞ to ∞

Formulae:

$$cov_{x,y} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{N - 1}$$

$cov_{x,y}$ = covariance between variable a and y
x_i = data value of x
y_i = data value of y
$\bar{x}$ = mean of x
$\bar{y}$ = mean of y
N = number of data values

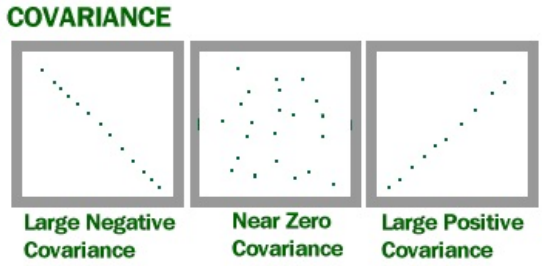
Types of Covariance:

Covariance can have both positive and negative values. Based on this, it has two types:

1. Positive Covariance
2. Negative Covariance

Positive Covariance
If the covariance for any two variables is positive, that means, both the variables move in the same direction. Here, the variables show similar behaviour. That means, if the values (greater or lesser) of one variable corresponds to the values of another variable, then they are said to be in positive covariance.

Negative Covariance
If the covariance for any two variables is negative, that means, both the variables move in the opposite direction. It is the opposite case of positive covariance, where greater values of one variable corresponds to lesser values of another variable and vice-versa.



Limitations:

1. If you change the unit of feature covariance will differ.

2. covariance will only tells about the direction not the strength.

Mathematically solved

Question: The table below describes the rate of economic growth (xi) and the rate of return on the S&P 500 (y_i). Using the covariance formula, determine whether economic growth and S&P 500 returns have a positive or inverse relationship. Before you compute the covariance, calculate the mean of x and y.

Economic Growth % (x_i)	**S&P 500 Returns % (y_i)**
2.1	8
2.5	12
4.0	14
3.6	10

x = 2.1, 2.5, 4.0, and 3.6 (economic growth)

y = 8, 12, 14, and 10 (S&P 500 returns)

Find $\bar{x}$ and $\bar{y}$.

Solution:

$\bar{x} = \frac{\sum x_i}{n}$

$\bar{x} = \frac{2.1+2.5+4+3.6}{4}$

$\bar{x} = \frac{12.2}{4}$

$\bar{x} = 3.1$

$\bar{y} = \frac{\sum x_i}{n}$

$\bar{y} = \frac{8+12+14+10}{4}$

$\bar{y} = \frac{44}{4}$

$\bar{y} = 11$

Now, substitute these values into the covariance formula to determine the relationship between economic growth and S&P 500 returns.

xi	yi	xi − $\bar{x}$	yi − $\bar{y}$
2.1	8	-1	-3
2.5	12	-0.6	1
4.0	14	0.9	3
3.6	10	0.5	-1

$$Cov(x,y) = \frac{(-1)(-3)+(-0.6)1+(0.9)3+(0.5)(-1)}{4-1} = \frac{3-0.6+2.7-0.5}{3} = \frac{4.6}{3} = 1.533$$

Note: If mean is corrupted then robust variance use where median into consideration.

Co-variance matrix:

Co-variance matrix is a square matrix giving the covariance between each pair of feature in a data. With the covariance we can calculate

entries of the covariance matrix, which is a square matrix given by Ci,j=σ(xi,xj) where C∈Rd×d and d describes the dimension or number of random variables of the data (e.g. the number of features like height, width, weight, …). Also the covariance matrix is symmetric since σ(xi,xj)=σ(xj,xi). The diagonal entries of the covariance matrix are the variances and the other entries are the covariances. For this reason the covariance matrix is sometimes called the variance-covariance matrix. The calculation for the covariance matrix can be also expressed as

$$C = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})(X_i - \bar{X})^T$$

Mathematically solved: Let's see how to calculate covariance matrix.

Example 1: Find the population covariance matrix for the following table.

Score	Age
68	29
60	26
58	30
40	35

Solution: The formula for population variance is $\frac{\sum_1^n (x_i-\mu)^2}{n}$.

$\mu_x = 56.5, n = 4$

$var(x) = [(68 - 56.5)^2 + (60 - 56.5)^2 + (58 - 56.5)^2 + (40 - 56.5)^2] / 4 = 104.75$

$\mu_y = 30, n = 4$

$var(y) = [(29 - 30)^2 + (26 - 30)^2 + (30 - 30)^2 + (35 - 30)^2] / 4 = 10.5$

$cov(x, y) = \frac{\sum_1^4 (x_i-\mu_x)(y_i-\mu_y)}{4}$

$cov(x, y) = -27$

The variance covariance matrix is given as follows:

$$\begin{bmatrix} 104.7 & -27 \\ -27 & 10.5 \end{bmatrix}.$$

Implementation in python:

In [67]:
```
# Python code to demonstrate the
# use of numpy.cov
import numpy as np

x = [68,60,58,40]

y = [29,26,30,35]

# find out covariance with respect columns
cov_mat = np.stack((x, y), axis = 0)

print(np.cov(cov_mat))
```

```
[[139.66666667 -36.        ]
 [-36.          14.        ]]
```

Code Explanation: Don't get confused why we are getting different values because in python implementation of co-variance we use n-1(unbiased variance) and in math we took only n(biased variance)

Pearson correlation coefficient(PCC):

PCC are used in statistics to measure how strong a linear relationship is between two variables.
The range of the possible results of PCC is (-1,1) where:

1. 0 indicates no correlation
2. 1 indicates a perfect positive correlation
3. -1 indicates a perfect negative correlation

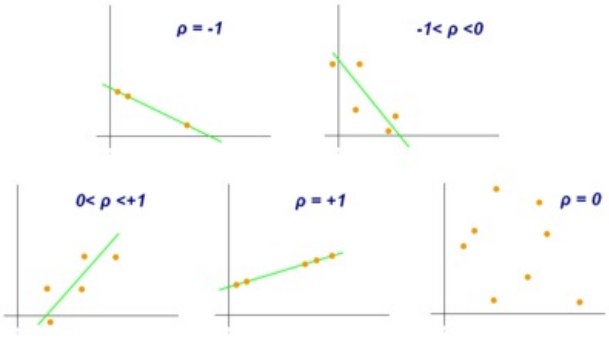
- The maximum correlation=1 when a straight line with a positive slope can go through the center of every data point.
- The minimum correlation=-1 when a straight line with a negative slope can go through the center of every data point.

Note: correlation doesn't depends on the scale of data.

Formulae:

$$r = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}}$$

r = correlation coefficient
x_i = values of the x-variable in a sample
$\bar{x}$ = mean of the values of the x-variable
y_i = values of the y-variable in a sample
$\bar{y}$ = mean of the values of the y-variable



Limitation:

The PCC only follows linear relationship if in case your data follows linear method then PCC works amazingly but for non linear structure PCC gets fails.

Question 1: Calculate the linear correlation coefficient for the following data. X = 4, 8 ,12, 16 and Y = 5, 10, 15, 20.

Solution:

Given variables are,

X = 4, 8 ,12, 16 and Y = 5, 10, 15, 20

For finding the linear coefficient of these data, we need to first construct a table for the required values.

x	y	x^2	y^2	XY

4	5	16	25	20
8	10	64	100	80
12	15	144	225	180
16	20	256	400	320
Σ x = 40	Σ y =50	480	750	600

According to the formula of linear correlation we have,

$$r(xy) = \frac{(4\times600)-(40\times50)}{\sqrt{4(480)-40^2}\sqrt{4(750)-50^2}}$$

$$r(xy) = \frac{2400-2000}{\sqrt{1920-1600}\sqrt{3000-2500}}$$

$$r(xy) = \frac{400}{\sqrt{320}\sqrt{500}}$$

$$r(xy) = \frac{400}{17.89\times22.36}$$

$$r(xy) = \frac{400}{400} = 1$$

Therefore, r(xy) = 1

Spearman Rank correlation Coefficient:

SRCC covers some of the limitations of PCC. It does not carry any assumption about the distribution of the data. SRCC is a test that is used to measure the degree of association between two variables by assigning rank to the values of each random variable by assigning rank to the value of each random variable and computing PCC out of it.

X Y Rank(X) Rank(Y) 121 56 6 7 124 34 8 6 101 12 4 2 96 32 3 4 231 14 10 3 123 35 7 5 129 7 9 1 111 76 5 8 78 120 1 10 91 101 2 9

Given two random variable X and Y. Compute rank of each random variable, such that the least value has Rank1. Then apply the PCC on Rank(X),Rank(Y) to compute SRCC

$$r_s = \rho_{\mathrm{rg}_X,\mathrm{rg}_Y} = \frac{\mathrm{cov}(\mathrm{rg}_X,\mathrm{rg}_Y)}{\sigma_{\mathrm{rg}_X}\sigma_{\mathrm{rg}_Y}},$$

where

ρ denotes the usual Pearson correlation coefficient, but applied to the rank variables,
$\mathrm{cov}(\mathrm{rg}_X,\mathrm{rg}_Y)$ is the covariance of the rank variables,
σ_{rg_X} and σ_{rg_Y} are the standard deviations of the rank variables.

SRCC range between -1 to +1 and works well with monotonically increasing and decreasing function.

Mathematically solved

Question: The following table provides data about the percentage of students who have free university meals and their CGPA scores. Calculate the Spearman's Rank

Correlation between the two and interpret the result.

State University	% of students having free meals	% of students scoring above 8.5 CGPA
Pune	14.4	54
Chennai	7.2	64
Delhi	27.5	44
Kanpur	33.8	32
Ahmedabad	38.0	37
Indore	15.9	68
Guwahati	4.9	62

Solution: Let us first assign the random variables to the required data –

X – % of students having free meals
Y – % of students scoring above 8.5 CGPA

Before proceeding with the calculation, we'll need to assign ranks to the data corresponding to each state university. We construct the table for the rank as below –

State University	$d_X = Ranks_X$	$d_Y = Ranks_Y$	$d = (d_X - d_Y)$	d^2
Pune	3	4	-1	1
Chennai	2	6	-4	16
Delhi	5	3	2	4

Kanpur	6	1	5	25
Ahmedabad	7	2	5	25
Indore	4	7	-3	9
Guwahati	1	5	-4	16
				$\Sigma d^2 = 96$

Now, using the formula(with n = 7 here) –

$$r_R = 1 - \frac{6\Sigma_i {d_i}^2}{n(n^2-1)}$$

$$= 1 - \frac{6.96}{7.(49-1)}$$

$$= 1 - \frac{576}{336}$$

$$= -0.714$$

Such a strong negative coefficient of correlation gives away an important implication – the universities with the highest percentage of students consuming free meals tend to have the least successful results (and vice-versa). Similarly, we can solve all other questions.

Feature scaling:

Feature scaling refers to some type of mathematical operations and transformation that we have to do on the data itself before we build complex model on such data. In feature scaling we have two types:

1. Feature normalization
2. Feature scaling

What is feature normalization:

Normalization is a scaling technique in which values are shifted and rescaled so that they end up ranging between 0 and 1. It is also known as Min-Max scaling.

Here's the formula for normalization:

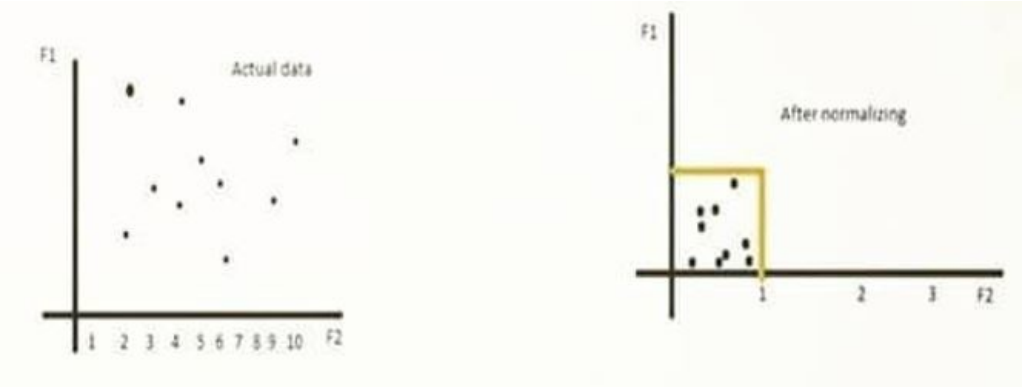
$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Here, Xmax and Xmin are the maximum and the minimum values of the feature respectively.

- When the value of X is the minimum value in the column, the numerator will be 0, and hence X' is 0

- On the other hand, when the value of X is the maximum value in the column, the numerator is equal to the denominator and thus the value of X' is 1
- If the value of X is between the minimum and the maximum value, then the value of X' is between 0 and 1

Geometric intuition:



after apply column normalization on any feature. All the datapoints get into unit cube. In column normalization your data can be anywhere in the N dimension space but after applying column normalization take your data from N dimension space and put it into unit cube.

In [68]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import warnings
warnings.filterwarnings('ignore')
'''downlaod iris.csv from https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/data/iris.csv'''
#Load Iris.csv into a pandas dataFrame.
iris = pd.read_csv("https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/data/iris.csv")
iris
```

Out[68]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

150 rows × 5 columns

In [69]:
```
from sklearn.preprocessing import MinMaxScaler
iris=iris[['sepal_length','sepal_width','petal_length','petal_width']]
norm=MinMaxScaler().fit(iris)
norm_data=norm.transform(iris)
norm_data
```

Out[69]:
```
array([[0.22222222, 0.625     , 0.06779661, 0.04166667],
       [0.16666667, 0.41666667, 0.06779661, 0.04166667],
       [0.11111111, 0.5       , 0.05084746, 0.04166667],
       [0.08333333, 0.45833333, 0.08474576, 0.04166667],
       [0.19444444, 0.66666667, 0.06779661, 0.04166667],
       [0.30555556, 0.79166667, 0.11864407, 0.125     ],
       [0.08333333, 0.58333333, 0.06779661, 0.08333333],
       [0.19444444, 0.58333333, 0.08474576, 0.04166667],
       [0.02777778, 0.375     , 0.06779661, 0.04166667],
       [0.16666667, 0.45833333, 0.08474576, 0.        ],
       [0.30555556, 0.70833333, 0.08474576, 0.04166667],
       [0.13888889, 0.58333333, 0.10169492, 0.04166667],
       [0.13888889, 0.41666667, 0.06779661, 0.        ],
       [0.        , 0.41666667, 0.01694915, 0.        ],
       [0.41666667, 0.83333333, 0.03389831, 0.04166667],
```

```
[0.38888889, 1.        , 0.08474576, 0.125     ],
[0.30555556, 0.79166667, 0.05084746, 0.125     ],
[0.22222222, 0.625     , 0.06779661, 0.08333333],
[0.38888889, 0.75      , 0.11864407, 0.08333333],
[0.22222222, 0.75      , 0.08474576, 0.08333333],
[0.30555556, 0.58333333, 0.11864407, 0.04166667],
[0.22222222, 0.70833333, 0.08474576, 0.125     ],
[0.08333333, 0.66666667, 0.        , 0.04166667],
[0.22222222, 0.54166667, 0.11864407, 0.16666667],
[0.13888889, 0.58333333, 0.15254237, 0.04166667],
[0.19444444, 0.41666667, 0.10169492, 0.04166667],
[0.19444444, 0.58333333, 0.10169492, 0.125     ],
[0.25      , 0.625     , 0.08474576, 0.04166667],
[0.25      , 0.58333333, 0.06779661, 0.04166667],
[0.11111111, 0.5       , 0.10169492, 0.04166667],
[0.13888889, 0.45833333, 0.10169492, 0.04166667],
[0.30555556, 0.58333333, 0.08474576, 0.125     ],
[0.25      , 0.875     , 0.08474576, 0.        ],
[0.33333333, 0.91666667, 0.06779661, 0.04166667],
[0.16666667, 0.45833333, 0.08474576, 0.        ],
[0.19444444, 0.5       , 0.03389831, 0.04166667],
[0.33333333, 0.625     , 0.05084746, 0.04166667],
[0.16666667, 0.45833333, 0.08474576, 0.        ],
[0.02777778, 0.41666667, 0.05084746, 0.04166667],
[0.22222222, 0.58333333, 0.08474576, 0.04166667],
[0.19444444, 0.625     , 0.05084746, 0.08333333],
[0.05555556, 0.125     , 0.05084746, 0.08333333],
[0.02777778, 0.5       , 0.05084746, 0.04166667],
[0.19444444, 0.625     , 0.10169492, 0.20833333],
[0.22222222, 0.75      , 0.15254237, 0.125     ],
[0.13888889, 0.41666667, 0.06779661, 0.08333333],
[0.22222222, 0.75      , 0.10169492, 0.04166667],
[0.08333333, 0.5       , 0.06779661, 0.04166667],
[0.27777778, 0.70833333, 0.08474576, 0.04166667],
[0.19444444, 0.54166667, 0.06779661, 0.04166667],
[0.75      , 0.5       , 0.62711864, 0.54166667],
[0.58333333, 0.5       , 0.59322034, 0.58333333],
[0.72222222, 0.45833333, 0.66101695, 0.58333333],
[0.33333333, 0.125     , 0.50847458, 0.5       ],
[0.61111111, 0.33333333, 0.61016949, 0.58333333],
[0.38888889, 0.33333333, 0.59322034, 0.5       ],
[0.55555556, 0.54166667, 0.62711864, 0.625     ],
[0.16666667, 0.16666667, 0.38983051, 0.375     ],
[0.63888889, 0.375     , 0.61016949, 0.5       ],
[0.25      , 0.29166667, 0.49152542, 0.54166667],
[0.19444444, 0.        , 0.42372881, 0.375     ],
[0.44444444, 0.41666667, 0.54237288, 0.58333333],
[0.47222222, 0.08333333, 0.50847458, 0.375     ],
[0.5       , 0.375     , 0.62711864, 0.54166667],
[0.36111111, 0.375     , 0.44067797, 0.5       ],
[0.66666667, 0.45833333, 0.57627119, 0.54166667],
[0.36111111, 0.41666667, 0.59322034, 0.58333333],
[0.41666667, 0.29166667, 0.52542373, 0.375     ],
[0.52777778, 0.08333333, 0.59322034, 0.58333333],
[0.36111111, 0.20833333, 0.49152542, 0.41666667],
[0.44444444, 0.5       , 0.6440678 , 0.70833333],
[0.5       , 0.33333333, 0.50847458, 0.5       ],
[0.55555556, 0.20833333, 0.66101695, 0.58333333],
[0.5       , 0.33333333, 0.62711864, 0.45833333],
[0.58333333, 0.375     , 0.55932203, 0.5       ],
[0.63888889, 0.41666667, 0.57627119, 0.54166667],
[0.69444444, 0.33333333, 0.6440678 , 0.54166667],
[0.66666667, 0.41666667, 0.6779661 , 0.66666667],
[0.47222222, 0.375     , 0.59322034, 0.58333333],
[0.38888889, 0.25      , 0.42372881, 0.375     ],
[0.33333333, 0.16666667, 0.47457627, 0.41666667],
[0.33333333, 0.16666667, 0.45762712, 0.375     ],
[0.41666667, 0.29166667, 0.49152542, 0.45833333],
[0.47222222, 0.29166667, 0.69491525, 0.625     ],
[0.30555556, 0.41666667, 0.59322034, 0.58333333],
[0.47222222, 0.58333333, 0.59322034, 0.625     ],
[0.66666667, 0.45833333, 0.62711864, 0.58333333],
[0.55555556, 0.125     , 0.57627119, 0.5       ],
[0.36111111, 0.41666667, 0.52542373, 0.5       ],
[0.33333333, 0.20833333, 0.50847458, 0.5       ],
[0.33333333, 0.25      , 0.57627119, 0.45833333],
[0.5       , 0.41666667, 0.61016949, 0.54166667],
[0.41666667, 0.25      , 0.50847458, 0.45833333],
[0.19444444, 0.125     , 0.38983051, 0.375     ],
[0.36111111, 0.29166667, 0.54237288, 0.5       ],
[0.38888889, 0.41666667, 0.54237288, 0.45833333],
[0.38888889, 0.375     , 0.54237288, 0.5       ],
[0.52777778, 0.375     , 0.55932203, 0.5       ],
```

```
       [0.22222222, 0.20833333, 0.33898305, 0.41666667],
       [0.38888889, 0.33333333, 0.52542373, 0.5       ],
       [0.55555556, 0.54166667, 0.84745763, 1.        ],
       [0.41666667, 0.29166667, 0.69491525, 0.75      ],
       [0.77777778, 0.41666667, 0.83050847, 0.83333333],
       [0.55555556, 0.375     , 0.77966102, 0.70833333],
       [0.61111111, 0.41666667, 0.81355932, 0.875     ],
       [0.91666667, 0.41666667, 0.94915254, 0.83333333],
       [0.16666667, 0.20833333, 0.59322034, 0.66666667],
       [0.83333333, 0.375     , 0.89830508, 0.70833333],
       [0.66666667, 0.20833333, 0.81355932, 0.70833333],
       [0.80555556, 0.66666667, 0.86440678, 1.        ],
       [0.61111111, 0.5       , 0.69491525, 0.79166667],
       [0.58333333, 0.29166667, 0.72881356, 0.75      ],
       [0.69444444, 0.41666667, 0.76271186, 0.83333333],
       [0.38888889, 0.20833333, 0.6779661 , 0.79166667],
       [0.41666667, 0.33333333, 0.69491525, 0.95833333],
       [0.58333333, 0.5       , 0.72881356, 0.91666667],
       [0.61111111, 0.41666667, 0.76271186, 0.70833333],
       [0.94444444, 0.75      , 0.96610169, 0.875     ],
       [0.94444444, 0.25      , 1.        , 0.91666667],
       [0.47222222, 0.08333333, 0.6779661 , 0.58333333],
       [0.72222222, 0.5       , 0.79661017, 0.91666667],
       [0.36111111, 0.33333333, 0.66101695, 0.79166667],
       [0.94444444, 0.33333333, 0.96610169, 0.79166667],
       [0.55555556, 0.29166667, 0.66101695, 0.70833333],
       [0.66666667, 0.54166667, 0.79661017, 0.83333333],
       [0.80555556, 0.5       , 0.84745763, 0.70833333],
       [0.52777778, 0.33333333, 0.6440678 , 0.70833333],
       [0.5       , 0.41666667, 0.66101695, 0.70833333],
       [0.58333333, 0.33333333, 0.77966102, 0.83333333],
       [0.80555556, 0.41666667, 0.81355932, 0.625     ],
       [0.86111111, 0.33333333, 0.86440678, 0.75      ],
       [1.        , 0.75      , 0.91525424, 0.79166667],
       [0.58333333, 0.33333333, 0.77966102, 0.875     ],
       [0.55555556, 0.33333333, 0.69491525, 0.58333333],
       [0.5       , 0.25      , 0.77966102, 0.54166667],
       [0.94444444, 0.41666667, 0.86440678, 0.91666667],
       [0.55555556, 0.58333333, 0.77966102, 0.95833333],
       [0.58333333, 0.45833333, 0.76271186, 0.70833333],
       [0.47222222, 0.41666667, 0.6440678 , 0.70833333],
       [0.72222222, 0.45833333, 0.74576271, 0.83333333],
       [0.66666667, 0.45833333, 0.77966102, 0.95833333],
       [0.72222222, 0.45833333, 0.69491525, 0.91666667],
       [0.41666667, 0.29166667, 0.69491525, 0.75      ],
       [0.69444444, 0.5       , 0.83050847, 0.91666667],
       [0.66666667, 0.54166667, 0.79661017, 1.        ],
       [0.66666667, 0.41666667, 0.71186441, 0.91666667],
       [0.55555556, 0.20833333, 0.6779661 , 0.75      ],
       [0.61111111, 0.41666667, 0.71186441, 0.79166667],
       [0.52777778, 0.58333333, 0.74576271, 0.91666667],
       [0.44444444, 0.41666667, 0.69491525, 0.70833333]])
```

How to Normalize a Dataset Without Converting Columns to Array?

In [70]:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from sklearn import preprocessing
import warnings
warnings.filterwarnings('ignore')
'''downlaod iris.csv from https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/data/iris.csv'''
#Load Iris.csv into a pandas dataFrame.
iris = pd.read_csv("https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/data/iris.csv")
iris=iris[['sepal_length','sepal_width','petal_length','petal_width']]
scaler = preprocessing.MinMaxScaler()
names = iris.columns
d = scaler.fit_transform(iris)
scaled_df = pd.DataFrame(d, columns=names)
scaled_df.head()
```

Out[70]:

	sepal_length	sepal_width	petal_length	petal_width
0	0.222222	0.625000	0.067797	0.041667
1	0.166667	0.416667	0.067797	0.041667
2	0.111111	0.500000	0.050847	0.041667
3	0.083333	0.458333	0.084746	0.041667
4	0.194444	0.666667	0.067797	0.041667

what is column standardization:

Standardization is another scaling technique where the values are centered around the mean with a unit standard deviation. This means that the mean of the attribute becomes zero and the resultant distribution has a unit standard deviation.

Here's the formula for standardization:

$$X' = \frac{X - \mu}{\sigma}$$

$$\mu$$

Feature scaling: Mu is the mean of the feature values andFeature scaling: Sigmais the standard deviation of the feature values. Note that in this case, the values are not restricted to a particular range.

In [71]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import warnings
warnings.filterwarnings('ignore')
'''downlaod iris.csv from https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/data/iris.csv'''
#Load Iris.csv into a pandas dataFrame.
iris = pd.read_csv("https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/data/iris.csv")
iris
```

Out[71]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

150 rows × 5 columns

In [72]:
```
from sklearn.preprocessing import StandardScaler
iris=iris[['sepal_length','sepal_width','petal_length','petal_width']]
norm=StandardScaler().fit(iris)
stan_data=norm.transform(iris)
stan_data
```

Out[72]:
```
array([[-9.00681170e-01,  1.03205722e+00, -1.34127240e+00,
        -1.31297673e+00],
       [-1.14301691e+00, -1.24957601e-01, -1.34127240e+00,
        -1.31297673e+00],
       [-1.38535265e+00,  3.37848329e-01, -1.39813811e+00,
        -1.31297673e+00],
       [-1.50652052e+00,  1.06445364e-01, -1.28440670e+00,
        -1.31297673e+00],
       [-1.02184904e+00,  1.26346019e+00, -1.34127240e+00,
        -1.31297673e+00],
       [-5.37177559e-01,  1.95766909e+00, -1.17067529e+00,
        -1.05003079e+00],
       [-1.50652052e+00,  8.00654259e-01, -1.34127240e+00,
        -1.18150376e+00],
       [-1.02184904e+00,  8.00654259e-01, -1.28440670e+00,
        -1.31297673e+00],
       [-1.74885626e+00, -3.56360566e-01, -1.34127240e+00,
        -1.31297673e+00],
       [-1.14301691e+00,  1.06445364e-01, -1.28440670e+00,
        -1.44444970e+00],
       [-5.37177559e-01,  1.49486315e+00, -1.28440670e+00,
```

```
 -1.31297673e+00],
[-1.26418478e+00,  8.00654259e-01, -1.22754100e+00,
 -1.31297673e+00],
[-1.26418478e+00, -1.24957601e-01, -1.34127240e+00,
 -1.44444970e+00],
[-1.87002413e+00, -1.24957601e-01, -1.51186952e+00,
 -1.44444970e+00],
[-5.25060772e-02,  2.18907205e+00, -1.45500381e+00,
 -1.31297673e+00],
[-1.73673948e-01,  3.11468391e+00, -1.28440670e+00,
 -1.05003079e+00],
[-5.37177559e-01,  1.95766909e+00, -1.39813811e+00,
 -1.05003079e+00],
[-9.00681170e-01,  1.03205722e+00, -1.34127240e+00,
 -1.18150376e+00],
[-1.73673948e-01,  1.72626612e+00, -1.17067529e+00,
 -1.18150376e+00],
[-9.00681170e-01,  1.72626612e+00, -1.28440670e+00,
 -1.18150376e+00],
[-5.37177559e-01,  8.00654259e-01, -1.17067529e+00,
 -1.31297673e+00],
[-9.00681170e-01,  1.49486315e+00, -1.28440670e+00,
 -1.05003079e+00],
[-1.50652052e+00,  1.26346019e+00, -1.56873522e+00,
 -1.31297673e+00],
[-9.00681170e-01,  5.69251294e-01, -1.17067529e+00,
 -9.18557817e-01],
[-1.26418478e+00,  8.00654259e-01, -1.05694388e+00,
 -1.31297673e+00],
[-1.02184904e+00, -1.24957601e-01, -1.22754100e+00,
 -1.31297673e+00],
[-1.02184904e+00,  8.00654259e-01, -1.22754100e+00,
 -1.05003079e+00],
[-7.79513300e-01,  1.03205722e+00, -1.28440670e+00,
 -1.31297673e+00],
[-7.79513300e-01,  8.00654259e-01, -1.34127240e+00,
 -1.31297673e+00],
[-1.38535265e+00,  3.37848329e-01, -1.22754100e+00,
 -1.31297673e+00],
[-1.26418478e+00,  1.06445364e-01, -1.22754100e+00,
 -1.31297673e+00],
[-5.37177559e-01,  8.00654259e-01, -1.28440670e+00,
 -1.05003079e+00],
[-7.79513300e-01,  2.42047502e+00, -1.28440670e+00,
 -1.44444970e+00],
[-4.16009689e-01,  2.65187798e+00, -1.34127240e+00,
 -1.31297673e+00],
[-1.14301691e+00,  1.06445364e-01, -1.28440670e+00,
 -1.44444970e+00],
[-1.02184904e+00,  3.37848329e-01, -1.45500381e+00,
 -1.31297673e+00],
[-4.16009689e-01,  1.03205722e+00, -1.39813811e+00,
 -1.31297673e+00],
[-1.14301691e+00,  1.06445364e-01, -1.28440670e+00,
 -1.44444970e+00],
[-1.74885626e+00, -1.24957601e-01, -1.39813811e+00,
 -1.31297673e+00],
[-9.00681170e-01,  8.00654259e-01, -1.28440670e+00,
 -1.31297673e+00],
[-1.02184904e+00,  1.03205722e+00, -1.39813811e+00,
 -1.18150376e+00],
[-1.62768839e+00, -1.74477836e+00, -1.39813811e+00,
 -1.18150376e+00],
[-1.74885626e+00,  3.37848329e-01, -1.39813811e+00,
 -1.31297673e+00],
[-1.02184904e+00,  1.03205722e+00, -1.22754100e+00,
 -7.87084847e-01],
[-9.00681170e-01,  1.72626612e+00, -1.05694388e+00,
 -1.05003079e+00],
[-1.26418478e+00, -1.24957601e-01, -1.34127240e+00,
 -1.18150376e+00],
[-9.00681170e-01,  1.72626612e+00, -1.22754100e+00,
 -1.31297673e+00],
[-1.50652052e+00,  3.37848329e-01, -1.34127240e+00,
 -1.31297673e+00],
[-6.58345429e-01,  1.49486315e+00, -1.28440670e+00,
 -1.31297673e+00],
[-1.02184904e+00,  5.69251294e-01, -1.34127240e+00,
 -1.31297673e+00],
[ 1.40150837e+00,  3.37848329e-01,  5.35295827e-01,
  2.64698913e-01],
[ 6.74501145e-01,  3.37848329e-01,  4.21564419e-01,
  3.96171883e-01],
```

```
[ 1.28034050e+00,  1.06445364e-01,  6.49027235e-01,
  3.96171883e-01],
[-4.16009689e-01, -1.74477836e+00,  1.37235899e-01,
  1.33225943e-01],
[ 7.95669016e-01, -5.87763531e-01,  4.78430123e-01,
  3.96171883e-01],
[-1.73673948e-01, -5.87763531e-01,  4.21564419e-01,
  1.33225943e-01],
[ 5.53333275e-01,  5.69251294e-01,  5.35295827e-01,
  5.27644853e-01],
[-1.14301691e+00, -1.51337539e+00, -2.60824029e-01,
 -2.61192967e-01],
[ 9.16836886e-01, -3.56360566e-01,  4.78430123e-01,
  1.33225943e-01],
[-7.79513300e-01, -8.19166497e-01,  8.03701950e-02,
  2.64698913e-01],
[-1.02184904e+00, -2.43898725e+00, -1.47092621e-01,
 -2.61192967e-01],
[ 6.86617933e-02, -1.24957601e-01,  2.50967307e-01,
  3.96171883e-01],
[ 1.89829664e-01, -1.97618132e+00,  1.37235899e-01,
 -2.61192967e-01],
[ 3.10997534e-01, -3.56360566e-01,  5.35295827e-01,
  2.64698913e-01],
[-2.94841818e-01, -3.56360566e-01, -9.02269170e-02,
  1.33225943e-01],
[ 1.03800476e+00,  1.06445364e-01,  3.64698715e-01,
  2.64698913e-01],
[-2.94841818e-01, -1.24957601e-01,  4.21564419e-01,
  3.96171883e-01],
[-5.25060772e-02, -8.19166497e-01,  1.94101603e-01,
 -2.61192967e-01],
[ 4.32165405e-01, -1.97618132e+00,  4.21564419e-01,
  3.96171883e-01],
[-2.94841818e-01, -1.28197243e+00,  8.03701950e-02,
 -1.29719997e-01],
[ 6.86617933e-02,  3.37848329e-01,  5.92161531e-01,
  7.90590793e-01],
[ 3.10997534e-01, -5.87763531e-01,  1.37235899e-01,
  1.33225943e-01],
[ 5.53333275e-01, -1.28197243e+00,  6.49027235e-01,
  3.96171883e-01],
[ 3.10997534e-01, -5.87763531e-01,  5.35295827e-01,
  1.75297293e-03],
[ 6.74501145e-01, -3.56360566e-01,  3.07833011e-01,
  1.33225943e-01],
[ 9.16836886e-01, -1.24957601e-01,  3.64698715e-01,
  2.64698913e-01],
[ 1.15917263e+00, -5.87763531e-01,  5.92161531e-01,
  2.64698913e-01],
[ 1.03800476e+00, -1.24957601e-01,  7.05892939e-01,
  6.59117823e-01],
[ 1.89829664e-01, -3.56360566e-01,  4.21564419e-01,
  3.96171883e-01],
[-1.73673948e-01, -1.05056946e+00, -1.47092621e-01,
 -2.61192967e-01],
[-4.16009689e-01, -1.51337539e+00,  2.35044910e-02,
 -1.29719997e-01],
[-4.16009689e-01, -1.51337539e+00, -3.33612130e-02,
 -2.61192967e-01],
[-5.25060772e-02, -8.19166497e-01,  8.03701950e-02,
  1.75297293e-03],
[ 1.89829664e-01, -8.19166497e-01,  7.62758643e-01,
  5.27644853e-01],
[-5.37177559e-01, -1.24957601e-01,  4.21564419e-01,
  3.96171883e-01],
[ 1.89829664e-01,  8.00654259e-01,  4.21564419e-01,
  5.27644853e-01],
[ 1.03800476e+00,  1.06445364e-01,  5.35295827e-01,
  3.96171883e-01],
[ 5.53333275e-01, -1.74477836e+00,  3.64698715e-01,
  1.33225943e-01],
[-2.94841818e-01, -1.24957601e-01,  1.94101603e-01,
  1.33225943e-01],
[-4.16009689e-01, -1.28197243e+00,  1.37235899e-01,
  1.33225943e-01],
[-4.16009689e-01, -1.05056946e+00,  3.64698715e-01,
  1.75297293e-03],
[ 3.10997534e-01, -1.24957601e-01,  4.78430123e-01,
  2.64698913e-01],
[-5.25060772e-02, -1.05056946e+00,  1.37235899e-01,
  1.75297293e-03],
[-1.02184904e+00, -1.74477836e+00, -2.60824029e-01,
```

```
 -2.61192967e-01],
[-2.94841818e-01, -8.19166497e-01,  2.50967307e-01,
  1.33225943e-01],
[-1.73673948e-01, -1.24957601e-01,  2.50967307e-01,
  1.75297293e-03],
[-1.73673948e-01, -3.56360566e-01,  2.50967307e-01,
  1.33225943e-01],
[ 4.32165405e-01, -3.56360566e-01,  3.07833011e-01,
  1.33225943e-01],
[-9.00681170e-01, -1.28197243e+00, -4.31421141e-01,
 -1.29719997e-01],
[-1.73673948e-01, -5.87763531e-01,  1.94101603e-01,
  1.33225943e-01],
[ 5.53333275e-01,  5.69251294e-01,  1.27454998e+00,
  1.71090158e+00],
[-5.25060772e-02, -8.19166497e-01,  7.62758643e-01,
  9.22063763e-01],
[ 1.52267624e+00, -1.24957601e-01,  1.21768427e+00,
  1.18500970e+00],
[ 5.53333275e-01, -3.56360566e-01,  1.04708716e+00,
  7.90590793e-01],
[ 7.95669016e-01, -1.24957601e-01,  1.16081857e+00,
  1.31648267e+00],
[ 2.12851559e+00, -1.24957601e-01,  1.61574420e+00,
  1.18500970e+00],
[-1.14301691e+00, -1.28197243e+00,  4.21564419e-01,
  6.59117823e-01],
[ 1.76501198e+00, -3.56360566e-01,  1.44514709e+00,
  7.90590793e-01],
[ 1.03800476e+00, -1.28197243e+00,  1.16081857e+00,
  7.90590793e-01],
[ 1.64384411e+00,  1.26346019e+00,  1.33141568e+00,
  1.71090158e+00],
[ 7.95669016e-01,  3.37848329e-01,  7.62758643e-01,
  1.05353673e+00],
[ 6.74501145e-01, -8.19166497e-01,  8.76490051e-01,
  9.22063763e-01],
[ 1.15917263e+00, -1.24957601e-01,  9.90221459e-01,
  1.18500970e+00],
[-1.73673948e-01, -1.28197243e+00,  7.05892939e-01,
  1.05353673e+00],
[-5.25060772e-02, -5.87763531e-01,  7.62758643e-01,
  1.57942861e+00],
[ 6.74501145e-01,  3.37848329e-01,  8.76490051e-01,
  1.44795564e+00],
[ 7.95669016e-01, -1.24957601e-01,  9.90221459e-01,
  7.90590793e-01],
[ 2.24968346e+00,  1.72626612e+00,  1.67260991e+00,
  1.31648267e+00],
[ 2.24968346e+00, -1.05056946e+00,  1.78634131e+00,
  1.44795564e+00],
[ 1.89829664e-01, -1.97618132e+00,  7.05892939e-01,
  3.96171883e-01],
[ 1.28034050e+00,  3.37848329e-01,  1.10395287e+00,
  1.44795564e+00],
[-2.94841818e-01, -5.87763531e-01,  6.49027235e-01,
  1.05353673e+00],
[ 2.24968346e+00, -5.87763531e-01,  1.67260991e+00,
  1.05353673e+00],
[ 5.53333275e-01, -8.19166497e-01,  6.49027235e-01,
  7.90590793e-01],
[ 1.03800476e+00,  5.69251294e-01,  1.10395287e+00,
  1.18500970e+00],
[ 1.64384411e+00,  3.37848329e-01,  1.27454998e+00,
  7.90590793e-01],
[ 4.32165405e-01, -5.87763531e-01,  5.92161531e-01,
  7.90590793e-01],
[ 3.10997534e-01, -1.24957601e-01,  6.49027235e-01,
  7.90590793e-01],
[ 6.74501145e-01, -5.87763531e-01,  1.04708716e+00,
  1.18500970e+00],
[ 1.64384411e+00, -1.24957601e-01,  1.16081857e+00,
  5.27644853e-01],
[ 1.88617985e+00, -5.87763531e-01,  1.33141568e+00,
  9.22063763e-01],
[ 2.49201920e+00,  1.72626612e+00,  1.50201279e+00,
  1.05353673e+00],
[ 6.74501145e-01, -5.87763531e-01,  1.04708716e+00,
  1.31648267e+00],
[ 5.53333275e-01, -5.87763531e-01,  7.62758643e-01,
  3.96171883e-01],
[ 3.10997534e-01, -1.05056946e+00,  1.04708716e+00,
  2.64698913e-01],
```

```
[ 2.24968346e+00, -1.24957601e-01,  1.33141568e+00,
  1.44795564e+00],
[ 5.53333275e-01,  8.00654259e-01,  1.04708716e+00,
  1.57942861e+00],
[ 6.74501145e-01,  1.06445364e-01,  9.90221459e-01,
  7.90590793e-01],
[ 1.89829664e-01, -1.24957601e-01,  5.92161531e-01,
  7.90590793e-01],
[ 1.28034050e+00,  1.06445364e-01,  9.33355755e-01,
  1.18500970e+00],
[ 1.03800476e+00,  1.06445364e-01,  1.04708716e+00,
  1.57942861e+00],
[ 1.28034050e+00,  1.06445364e-01,  7.62758643e-01,
  1.44795564e+00],
[-5.25060772e-02, -8.19166497e-01,  7.62758643e-01,
  9.22063763e-01],
[ 1.15917263e+00,  3.37848329e-01,  1.21768427e+00,
  1.44795564e+00],
[ 1.03800476e+00,  5.69251294e-01,  1.10395287e+00,
  1.71090158e+00],
[ 1.03800476e+00, -1.24957601e-01,  8.19624347e-01,
  1.44795564e+00],
[ 5.53333275e-01, -1.28197243e+00,  7.05892939e-01,
  9.22063763e-01],
[ 7.95669016e-01, -1.24957601e-01,  8.19624347e-01,
  1.05353673e+00],
[ 4.32165405e-01,  8.00654259e-01,  9.33355755e-01,
  1.44795564e+00],
[ 6.86617933e-02, -1.24957601e-01,  7.62758643e-01,
  7.90590793e-01]])
```

How to standardized a Dataset Without Converting Columns to Array?

In [73]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from sklearn import preprocessing
import warnings
warnings.filterwarnings('ignore')
'''downlaod iris.csv from https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/data/iris.csv'''
#Load Iris.csv into a pandas dataFrame.
iris = pd.read_csv("https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/data/iris.csv")
iris=iris[['sepal_length','sepal_width','petal_length','petal_width']]
scaler = preprocessing.StandardScaler()
names = iris.columns
d = scaler.fit_transform(iris)
scaled_df = pd.DataFrame(d, columns=names)
scaled_df.head()
```

Out[73]:

	sepal_length	sepal_width	petal_length	petal_width
0	-0.900681	1.032057	-1.341272	-1.312977
1	-1.143017	-0.124958	-1.341272	-1.312977
2	-1.385353	0.337848	-1.398138	-1.312977
3	-1.506521	0.106445	-1.284407	-1.312977
4	-1.021849	1.263460	-1.341272	-1.312977

Why we need feature scaling: The real world dataset includes features that highly vary in magnitudes, units and range. If an algorithm is not using feature scaling method then it can consider the value 4000 meter to be greater than 6 km but it's actually not true and in this case algorithm will give wrong predictions. So, we use feature scaling technique to bring all values to the same magnitude and thus, tackle this issue.