APPLIED ROOTS

AAIC Technologies Pvt Ltd

Data Analysis and Visualization

Table of Contents

1. INTRODUCTION TO PROBABILITY AND STATISTICS

INTRODUCTION TO PROBABILITY AND STATISTICS

RANDOM OR STOCHASTIC PROCESSES

Random or Stochastic processes in essence are processes whose outcomes are random within a fixed set or range of values. The explanations and examples that follow will further clarify this definition. The terms random and stochastic mean the same thing and can be used interchangeably.

PROBABILITY:

Probability can be defined as the measure of certainty(or uncertainty) that a certain event or outcome will occur given a certain stochastic or random process. It is represented numerically as a number between zero and one. The probabilities of zero and one both represent certainty. The former expresses the certainty of the event not occurring and the latter expresses the certainty of the event occurring. Intermediate values represent degrees of uncertainty. Probability theory is the branch of mathematics which deals with probability.

Consider the following example: The natural process that creates the weather is a stochastic process whose outcomes can be considered random within a certain set of outcomes. Let us say that the weather station predicts that there is a 70 percent probability that it will rain tomorrow. How is this value determined and what does it mean? Simplistically it could be said that this conclusion was arrived at by looking up historical meteorological data and determining the percentage of times it rained on that particular day of the year. Since it rained 70% of the time, we conclude that there is a 70% chance that it will rain again tomorrow. This probability is mathematically expressed as:

P(Rain) = 0.7

The sum of all the probabilities for all possible outcomes for a particular event (In this case the weather tomorrow) will always be equal to one. That is:

P(Rain) + P(No Rain) = 1 or P(Rain) + P(Sunny) + P(Cloudy) = 1

Similarly, say we want to mathematically represent the probability of getting heads after a coin toss. Since there are only two outcomes for a coin toss and each of these outcomes are equally likely (assuming it is a fair coin), we say there is a 50% probability of getting heads. This is represented as:

P(Heads) = P(Tails) = 0.5

STATISTICS

Statistics is the science of techniques & methodologies that are used for the collection, presentation and analysis of quantitative data for the sake of decision making. Quantitative data generally refers to numeric data that represent values (like temperature) or counts (like the number of people in the world).

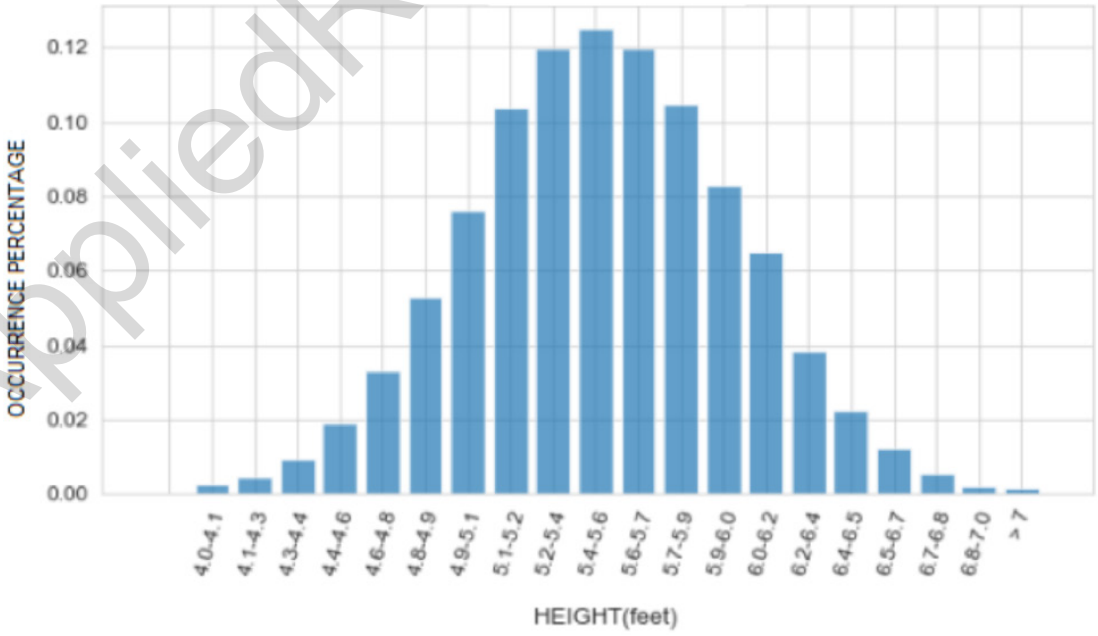
Statistics can be further sub classified into two main disciplines - Descriptive statistics for summarizing data and Inferential statistics for drawing conclusions from the data.

POPULATION AND SAMPLE:

Population refers to whole or complete data. For example, while studying the data on the heights of humans, the dataset consisting of the height measure of all the humans on the planet, would be called a Population.The dataset consisting of the height measures of a thousand randomly selected men and women from each country in the world would represent a Sample. It can be intuitively understood that such a subset could serve as a good approximation of the Population's characteristics and hence mathematical inferences derived from it could closely represent that of the Population.

PROBABILITY DISTRIBUTION:

Probability Distribution is a generic term used in statistics to refer to how the data is distributed among the set or range of values it contains. Let us say we have access to the "Human Heights" sample mentioned earlier. This data can then be represented as shown:



The data is subdivided or binned into suitable ranges and the proportions or percentage of data contained within each of these bins is visualized graphically. Now, these percentages express statistical fact, but more interestingly they also contain probabilistic information. It could be approximately inferred from the visualization above, that there is a 12.5% chance, the height of the next person we meet, would be in the range of 5.4 to 5.6 feet, or that there is a 2.3% chance that their height would be in the range of 6.4 to 6.5 feet.

The shape of the probability distribution is an attribute of the data being examined and gives us more insight into its nature. Data representing different population types exhibit different kinds of shapes. We shall be briefly exploring some of the commonly occurring probability distributions and their shapes in explanations that follow.

Generally distributions with shapes that exhibits the notions of centrality and symmetry like the one shown above (ie: there is maximum concentration of data points at a certain region and this concentration symmetrically reduces on both sides as we move away from this region of maximum concentration) are called Normal Distributions. It is one of the most commonly occurring distributions in nature and the most widely used distribution in statistics. We will describe this distribution in more depth in the explanations that follow.

RANDOM VARIABLES:

Random Variables are variables that represent the collection of **outcomes** of a stochastic process. For example, the collection of outcomes of a series of coin tosses is a random Variable. Here the possible set of outcomes are just two - Heads & Tails. If we map Heads to the number 1 and Tails to 0. Then the Random Variable could look something like [1, 1, 0, 1, 0, 1, 0, 0] for eight coin flips. The values of a Random Variable can change the next time it is recorded, but they can only contain a specific set of values.

Random Variables are mathematically defined by mapping the outcomes of the stochastic process to numbers. In the case of coin flips, we mathematically define the Random Variable as follows:

$$X = \begin{cases} 1 \text{ if outcome is Heads} \\ 0 \text{ if outcome is Tails} \end{cases}$$

Randon Variables (RVs) generally represented by capital letters

The actual Random Variable will be the byproduct of this definition applied to the outcomes of the stochastic process of the repetitive coin flips . For the example in consideration, it is mathematically expressed as: X = [1, 1, 0, 1, 0, 1, 0, 0].

Let us consider another example - The outcomes for a series of dice throws containing two 6 faced dies. Say we mathematically define a Random Variable as follows:

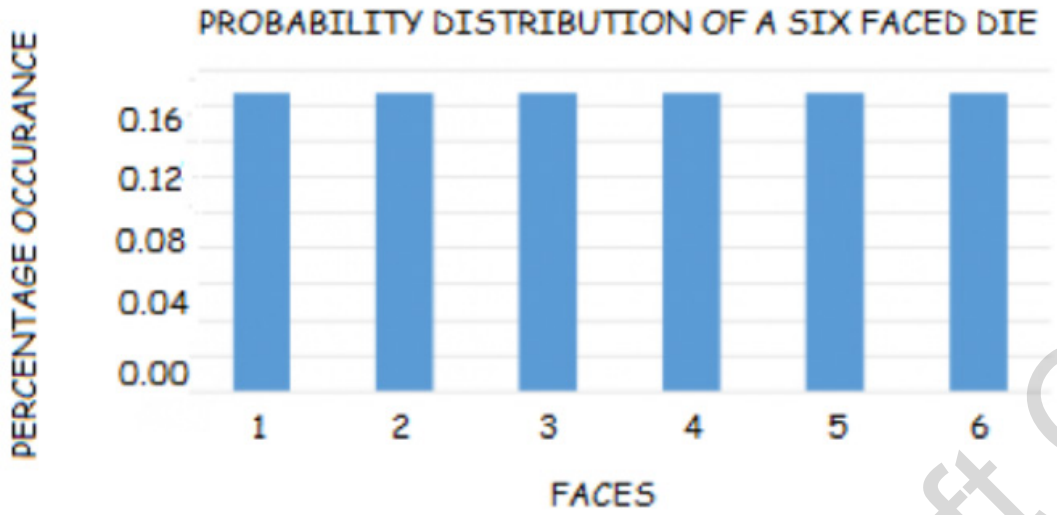
$$X = \begin{cases} 0 \text{ if } x1 + x2 \leq 4 \\ 1 \text{ if } x1 + x2 \leq 7 \ \& > 4 \\ 2 \text{ if } x1 + x2 \leq 10 \ \& > 7 \\ 3 \text{ if } x1 + x2 > 10 \end{cases}$$

where:
x1: Outcomes for die 1
x2: Outcomes for die 2

The set of all possible outcomes in this case would be: **{ 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 }**. Say we are considering 5 repetitions of dice throws. Then the collection of **actual outcomes**, due to the above mapping applied to this stochastic process could be the Random Variable
X = [3, 1, 2, 2, 0]

DISCRETE & CONTINUOUS RANDOM VARIABLES:

Random variables are classified into 2 basic categories based on their data type. Random variables that represent **discrete, non continuous data** are called Discrete Random Variables, whereas those that represent **continuous data** are called Continuous Random Variables. For example, the Random Variable that would represent the outcomes of ten thousand dice throws, would be a Discrete Random Variable, whereas the Random Variable representing the distribution of human heights, will be an example of a Continuous Random Variable. Note the difference between the two. Discrete random variables contain sets of discrete values like {0, 1} in case of coin flips and { 1, 2, 3, 4, 5, 6 } in case of dice throws, whereas continuous Random variables contain a range of continuous values. The heights of humans do not come in the form of fixed specific values, but instead they lie within a range of values like [4.2, 4.32, 5.75....]. Theoretically there are an infinite number of possible values that can exist between the first two values 4.2 & 4.32.



The probability distribution for a six faced die (discrete random variable) is shown above. The frequency of occurrence is uniform for all six sides, thus implying an equal probability of 1/6 for each of the faces to occur next. This type of distribution is called uniform distribution. It is another example of commonly occurring probability distributions.

DESCRIPTIVE STATISTICS:

MEASURES OF CENTRALITY - MEAN, MEDIAN & MODE:

A statistical measure of centrality is a number that summarises the typical value or "tendency" of a random variable or dataset. These measures give us a general indication as to where most values in a distribution fall.

The statistical mean refers to the average value of the Random Variable or data in question, Median refers to the middle value (or the average of the middle two values), of a Random Variable which has been sorted and Mode refers to that value in the Random Variable with the maximum frequency of occurrence. A dataset with no repetition will have no Mode.

$$\text{If } X = [x_1, x_2, x_3, \ldots x_i \ \ldots . x_n]$$

$$\text{Then mean of X, } \bar{x} = \frac{1}{n} \sum_i x_i$$

MEASURES OF DISPERSION:

MEAN ABSOLUTE DEVIATION, VARIANCE, STANDARD DEVIATION, MEDIAN ABSOLUTE DEVIATION & INTERQUARTILE RANGE:

A statistical measure of **dispersion** is a number or a range that summarises the scatter or spread of a random variable with respect to its mean value.

Mean Absolute deviation is the average of the absolute deviations of all the values of a Random Variable with respect to its mean.

Variance refers to the average squared distance of the values of a Random Variable with respect to its mean. The squared distance is used to eliminate negative values that can arise if just the difference of values were considered.

Standard deviation is simply the square root of the variance. Standard deviation modifies the Variance value to provide us a measure of dispersion that is of the same units as that of the Random Variable under examination.

Median Absolute Deviation is the median of the absolute differences of all the values of the Random Variable with respect to its mean.

$$\text{Mean Absolute Deviation, MD} = \frac{1}{n}\sum_{i} \text{abs}\left(\bar{x} - x_i\right)$$

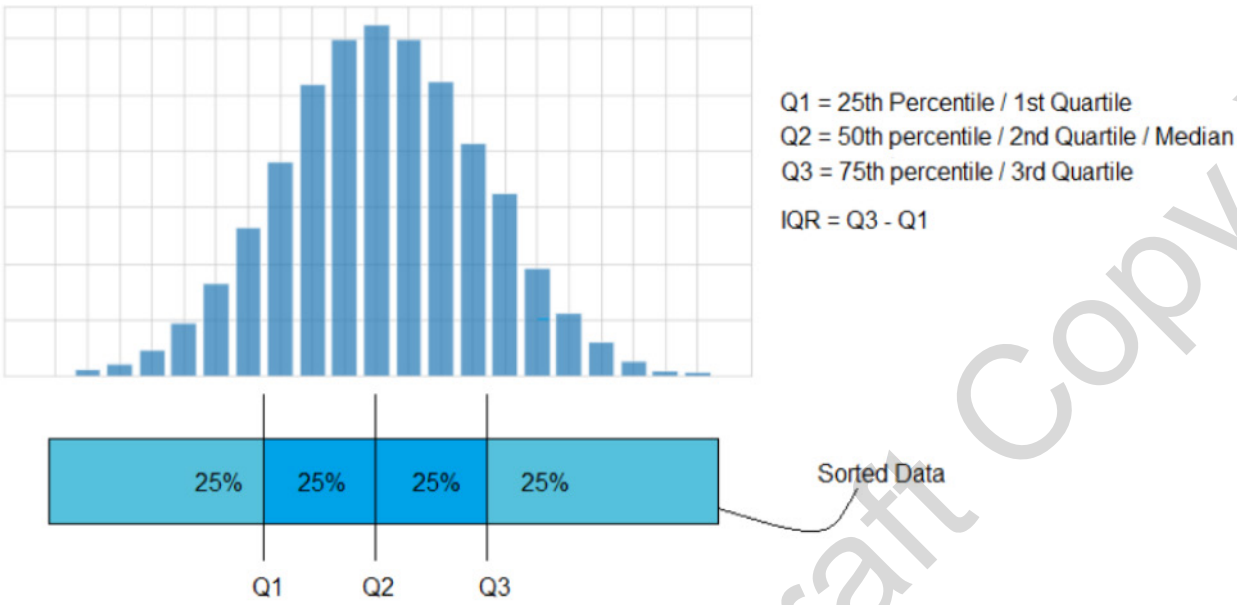
$$\text{Variance, } \sigma^2 = \frac{1}{n}\sum_{i} \left(\bar{x} - x_i\right)^2$$

$$\text{Standard Deviation} = \sigma$$

$$\text{Median Absolute Deviation, MAD} = \text{median}\left(\text{abs}\left(\bar{x} - x_i\right) \text{ for all i}\right)$$

The **InterQuartile Range (IQR)** is a measure of dispersion, based on dividing a dataset into quartiles. Quartiles divide a sorted dataset into four equal parts. The values that separate each of the parts are called the first, second, and third quartiles and they are denoted by Q1, Q2 and Q3 respectively. **IQR** is defined as the third quartile subtracted from the first quartile. It gives us the range of the midspread or the middle 50% of the dataset.

Distributions like the Normal Distribution are completely defined by the measures of mean and standard deviation as shall be seen in the topics that follow.



PARAMETER & STATISTIC:

In most cases it is impossible to have access to the data of an entire Population and it would be too computationally intensive to perform statistical analysis on such huge datasets. Therefore to gain insight about the population distribution, we randomly select from the population a suitably sized **sample** and perform our analysis on it. The statistical concepts of centrality, dispersion and shape (type of probability distribution) when used in context to a Population are called **Parameters** and when used in context to a Sample, are called **Statistics**. Most of statistics is about making inferences about the Populations parameters using information gained via the Sample's statistics.

PROBABILITY FUNCTIONS:

The mathematical characterization of a probability distribution is called a probability function. Given the distribution type (ex: Normal or Uniform or any other type of distribution) and its parameters (ex: Mean, Standard Deviation etc), these mathematical functions completely define the probability distribution of Random Variable they are associated with.

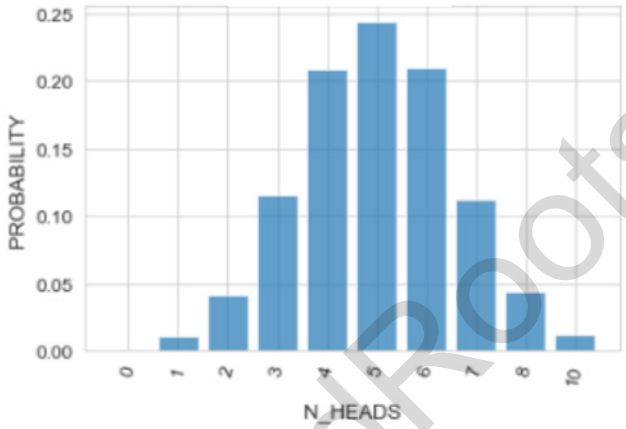
PROBABILITY MASS FUNCTION (PMF):

Probability functions for discrete random variables are called Probability Mass Functions. They mathematically define the probability of occurrence for each value or state within a discrete Random Variable.

Consider the Binomial distribution, which is a discrete probability distribution. The binomial distribution models situations where the following conditions are satisfied:

1. There are multiple trials associated with an outcome (ex: One outcome is the result of 10 coin flips).
2. Each outcome consists of the same number of trials.
3. There are only two possible results for each trial.(Heads or Tails)
4. The probability of occurrence of the desired result is equal for all trials. ie: P(Heads) is the same for all trials, similarly P(Tails) is also the same for all trials. P(Heads) and P(Tails) could be different, but they always add up to one.
5. Each result of each trial is independent of the outcome of the other trials.

If we have a sample of 1000 outcomes, where an outcome is the number of times heads occur in 10 coin flips (ie: 10 trials). Then this stochastic process would have a binomial distribution like the one shown below.



The Probability Mass Function for a binomial distribution is defined below:

$$P(X = x) = f(x) = \left(\frac{n!}{x!\,(n-x)!}\right) p^x (1-p)^{n-x}$$

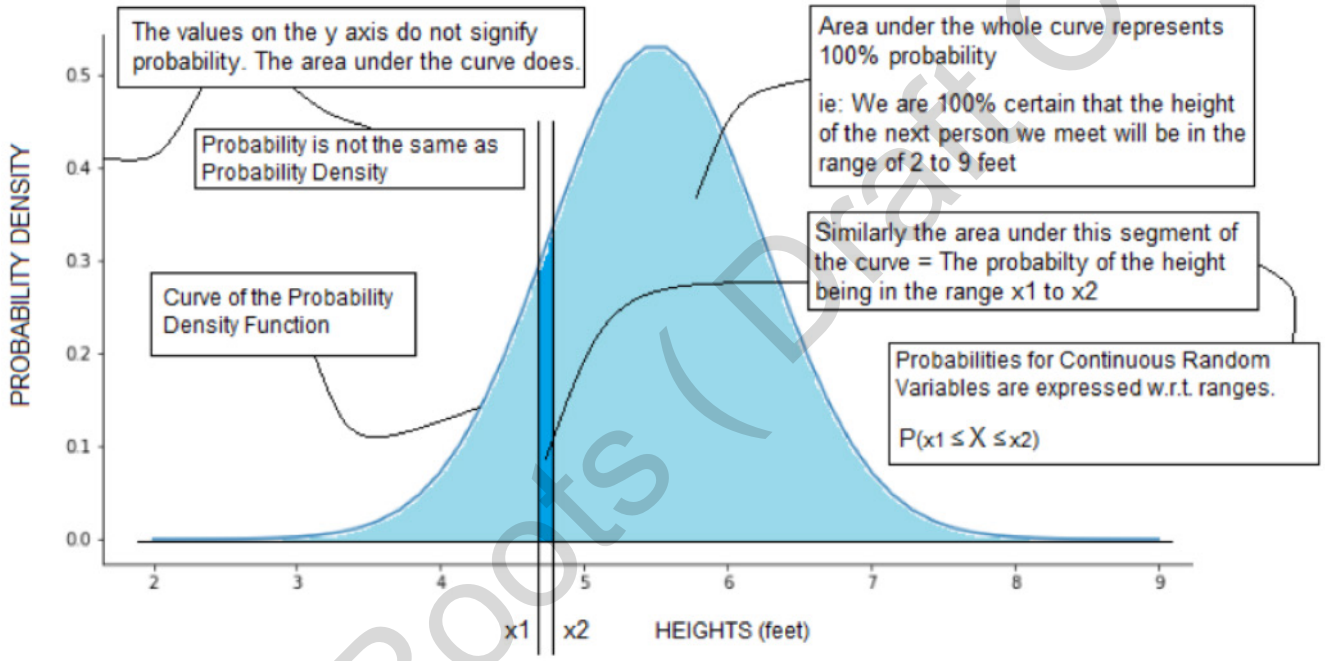
Where:
n = number of trials
x = number of occurrences of the desired result
p = probability of desired result for a single trial. (either result 1 or result 2)
n! = n(n-1)(n-2)(n-3)....(3)(2)(1)

In the example just mentioned, if the desired result is “seven heads per trial”, then the probability of that occurring would be obtained by substituting k = 7, n = 1000 and p = 0.5 in the function above.

PROBABILITY DENSITY FUNCTION (PDF):

The Probability Density Function is used to express the frequency distributions of continuous Random Variables. It maps the values of the Random Variable to their corresponding Probability Densities. Probability Density can have values greater than one, unlike probability.

The definition of what Probability Density is, is not important right now, what is important to understand is that the area under the curve defined by a PDF represents probability. The illustration below describes this concept using the "Human Heights" example.



The PDF is used to specify the probability of the random variable falling within a particular range of values, as opposed to taking on any one value. This probability is given by the integral of this variable's PDF over that range—that is, it is given by the area under the PDF between the lowest and greatest values of the range. The probability density function is nonnegative everywhere, and its integral over the entire space is equal to 1.

It can be seen that the shape of the PDF above mimics that of the binned Frequency Distribution of Human Heights shown earlier. Continuous Random Variables whose probabilities can be defined by the PDF curve shown above are said to follow the Normal Distribution.

The probability density function of a Normal Distribution is defined as:

$$\text{probability density (at x)} = f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

Where:

x = some value that belongs to the Random Variable in consideration

σ = Standard Deviation of the Random Variable

μ = Mean of the Random Variable

e = Euler's constant

The probability of a value in a normally distributed random variable, falling within the range [x1, x2] Is given by :

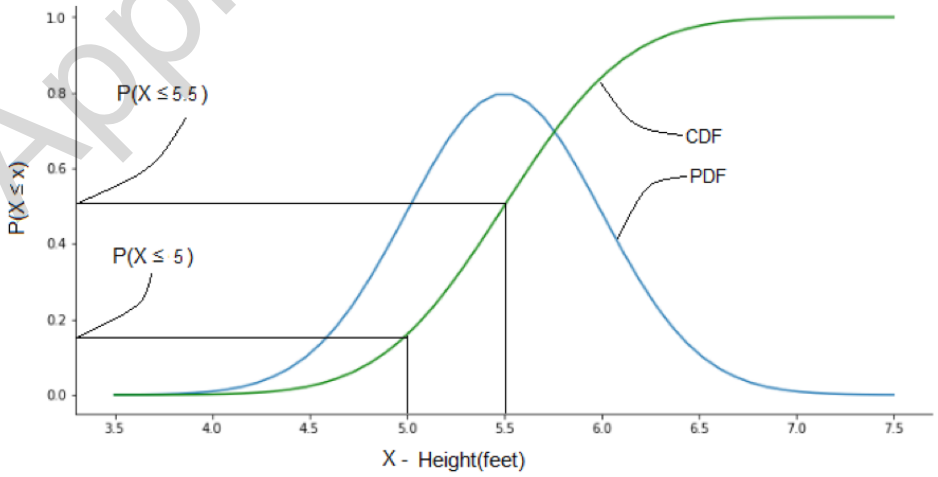
$$P(x1 \leq X \leq x2) = \text{Area under the curve between x1 \& x2} = \int_{x1}^{x2} f(x).dx$$

CUMULATIVE DENSITY FUNCTION (CDF):

The cumulative distribution function gives us the probability of any value in the random variable X being less than or equal to a particular value x. For example, say we want to know the probability for the height of the next person we come across, being less than or equal to 5 feet. We calculate this probability by counting the number of people in the Sample whose heights are less than or equal to 5 feet and dividing that with the count of the total number of people in the sample. The CDF is the function that maps all the values within a Random variable to their corresponding cumulative probabilities. Its is mathematically defined as:

CDF(x) = P(X ≤ x)

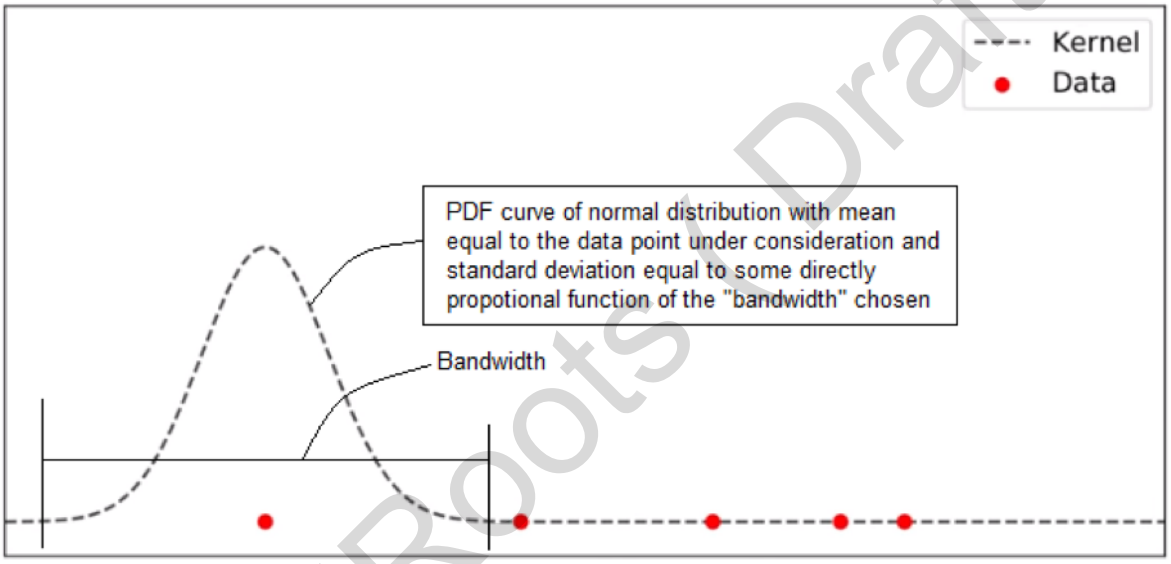
The advantage of the CDF is that it can be defined for any kind of random variable, discrete or continuous. The plot of the CDF for the "Heights" Random Variable, could like the one shown below:



From the plot above, we could say that the probability of the height being less than or equal to 5.5 feet is approximately 0.5 or 50%. The probability of the height being in the range 5 to 5.5 feet will be equal to P(X ≤ 5.5) minus P(X ≤ 5), that is approximately 0.5 minus 0.18, 0.32 or 32%.

KERNEL DENSITY ESTIMATION (KDE):

Sample datasets (Random Variables) can be incomplete and not represent the spectrum of the population accurately. They could contain larger proportions of certain parts of the distribution and thus lack smoothness. Many of the times the size of the dataset is suboptimal. Kernel density estimation is a statistical tool that allows us to create a smooth curve that approximates the PDF of a continuous Random Variable given these conditions.

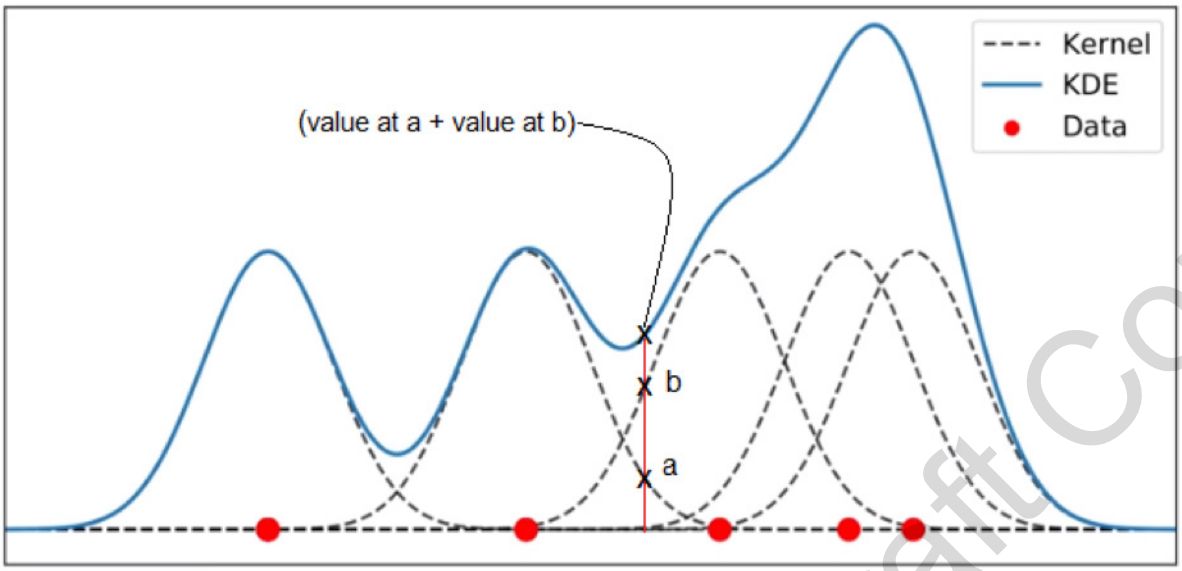


Consider a Continuous Random Variable of 5 data points as shown above. Say we want to approximate the PDF for this data. In KDE we do this by:

1. We replace each datapoint with the PDF of a normal distribution whose mean is equal to the value of the data point under consideration and whose standard deviation is a directly proportional function of the “band width” chosen. This PDF which is created for each individual data point is called a “Kernel”.

2. The band width is the range of values on either side of the data point which we want to consider, while creating such a kernel.

3. After each datapoint is replaced by a kernel, the PDF of the dataset as a whole is approximated by summing up the values of the individual kernels, at suitably chosen intervals, within the range of values that exist in the dataset. This step is illustrated in the image shown below.

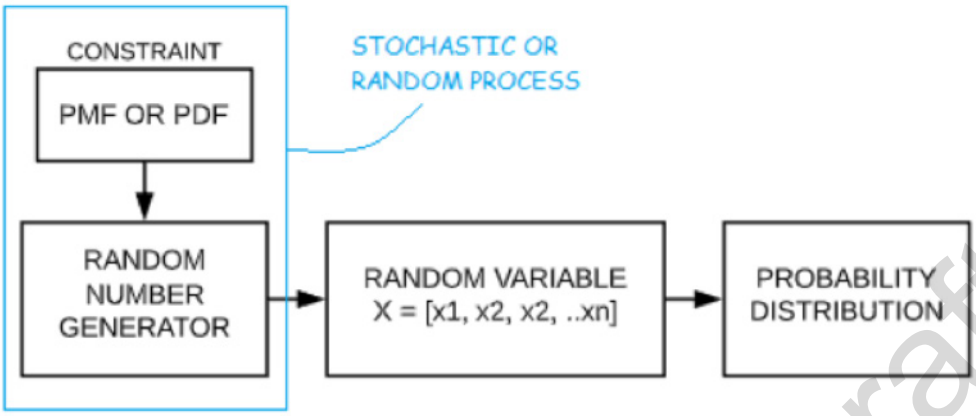


The “smoothness” of the PDF depends on the size of the bandwidth chosen. Larger bandwidths will make the PDF curve less responsive to the sparsity of the data. It is important to choose the right bandwidth to get a good approximation of the dataset’s PDF.

2. PROBABILITY DISTRIBUTIONS

PROBABILITY DISTRIBUTIONS

COMPUTATIONAL SIMULATION OF RANDOM OR STOCHASTIC PROCESSES:



Consider the image above. The Random Number Generator generates random numbers based on the constraints imposed on it. Say that we want to generate a Random Variable that exhibits a binomial distribution. We do this, by specifying the PMF of the binomial distribution, as the constraint/rule that the random number generator must obey.

So, depending on the PMF or PDF specified, we get Random Variables with various kinds of distributions.

The act of randomly sampling from a Population is analogous to collecting the outcomes of a random number generator that is constrained by the PDF or PMF of the population. For example, say we have a random number generator that is constrained to produce real numbers between 4 and 8(feet) as per the PDF of a normal distribution with a mean of 5.5 and a standard deviation of 0.5 and we collect 1000 numbers from this system, this would be almost the same as we randomly sampling 1000 heights from a real human population of the world.

The concept of computational simulation of stochastic processes will be demonstrated using the scipy package in python.

The code mainly consists of two kinds of functions:

1. Random Variable generators (Random number generators constrained by PDF or PMF)
2. Plotting functions

The reader is encouraged to experiment with these functions and observe the effects of changing the values of the parameters of these Random Variable generator functions.

EXAMPLES OF SOME COMMON TYPES OF DISTRIBUTIONS:

The plot of the PDF function (Random Variable VS Probability Density) is useful for describing the general shape of the data, but it does not provide any direct insight into the actual probabilities. It is much easier to get this information by plotting a "Binned Frequency Distribution" as shown in figure 1.1. By adjusting the granularity of the bins, we could derive quite a good approximation about the probability distribution of the data. This kind of plot could be considered as a continuous Random Variable's version of a PMF. Pairing this kind of plot together with the PDF plot while describing continuous data provides us a good description of the data.

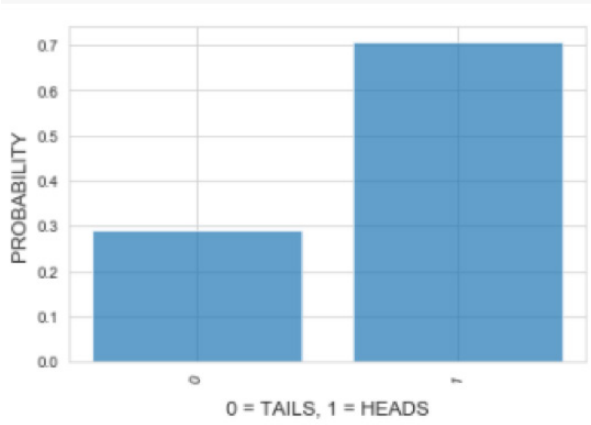
The PDF plots generated in the examples that follow, are actually an approximation of the Population's PDF. This approximation is achieved by applying the KDE algorithm on the sampled or simulated data.

THE BERNOULLI DISTRIBUTION:

A Bernoulli distribution is a discrete distribution that is used to represent or model stochastic processes that can have only two possible outcomes, where each outcome contains only a single trial. The outcomes are usually described in terms of 'success' or 'failure' with respect to the occurrence of one of the outcomes. Consider the coin toss example, which is an example of a Bernoulli distribution. Here we could describe the outcome of Heads as success and Tails as failure. If the probability of success is p, then the probability for failure will be 1-p. The Bernoulli distribution is the basic building block for other discrete distributions such as the binomial, hypergeometric and Poisson distributions. The probability distributions of multiple coin toss outcomes for an unbiased coin and a coin biased towards heads is shown below:

$$X = \begin{cases} 1 & \text{if outcome is Heads (or success)} \\ 0 & \text{if outcome is Tails (or failure)} \end{cases}$$

```
prob_success = 0.7
sample_size = 10000

random_var = fn_bermoulli_dist(prob_success, sample_size)

fn_plot_PMF(random_var, xlabel)
```



THE BINOMIAL DISTRIBUTION:

The binomial distribution has already been explained earlier, using the example of a sample of 1000 outcomes, where an outcome is the number of times heads occurs in 10 coin flips (ie: 10 trials). It can be seen that this distribution is basically an extension of the Bernoulli distribution, where the number of trials allowed for each outcome is now more than one. That is, a binomial event or trial is a sequence of bernoulli events.

To mathematically express that X is a Random Variable that follows a binomial distribution, we use the following notation: **X ~ B(n, p)**

The alphabet before the bracket signifies the distribution type (B is used to specifically signify Binomial) and the elements inside the brackets represent the parameters that define the distribution. Here "n" signifies the number of trials or events per outcome and p represents the probability of "success".

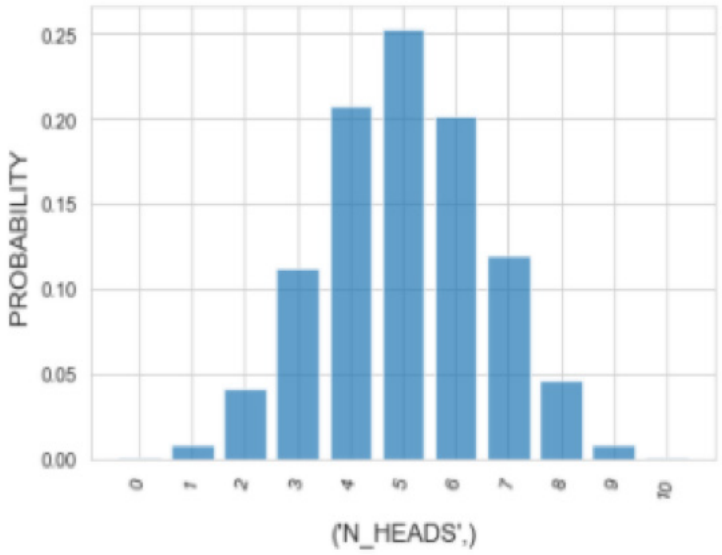
The PMF of a binomial distribution as shown earlier can now be understood in conjunction with the notation just described:

$$P(X = x) = \binom{n}{x} p^x (1-p)^{n-x}$$

where: $\binom{n}{x}$ is another way to represent $\left(\frac{n!}{x!(n-x)!}\right)$

Represents the number of ways x successes could be arranged among n trials.
ex: Number of ways 2 heads can occur if a coin were tossed 5 times.
ie: HHTTT, TTTHH, HTHTT, THTHT, etc

```
prob_success= 0.5
n_trials_per_outcome = 10
n_outcomes = 10000

random_var = fn_binomial_dist(prob_success, n_trials_per_outcome,
                              n_outcomes)
xlabel = 'N_HEADS'
fn_plot_PMF(random_var, xlabel)
```



THE HYPERGEOMETRIC DISTRIBUTION:

Imagine a bowl with equal numbers of white and black balls. Say one outcome is 10 repetitions (or trials) of the act of randomly drawing a ball and noting whether it is black and then putting it back into the bowl. The count of black balls for 1000 such outcomes will follow a binomial distribution.

Now in the example above, if the balls were drawn "without replacement", then the resulting distribution would be a Hypergeometric distribution. Without replacement means that the balls are not put back into the bowl once they have been drawn. This distribution is similar to the binomial distribution, but not the same, because the probability of success changes as the balls are removed. If the number of balls is large relative to the number of draws, the distribution will seem very similar to the binomial distribution because the chance of success changes less with each draw.

The hypergeometric distribution is mathematically represented as: **X ~ H(N, K, n)**

The PMF of a hypergeometric distribution is defined as shown below:

$$P(X = x) = \frac{\binom{K}{x}\binom{N-K}{n-x}}{\binom{N}{n}}$$

where

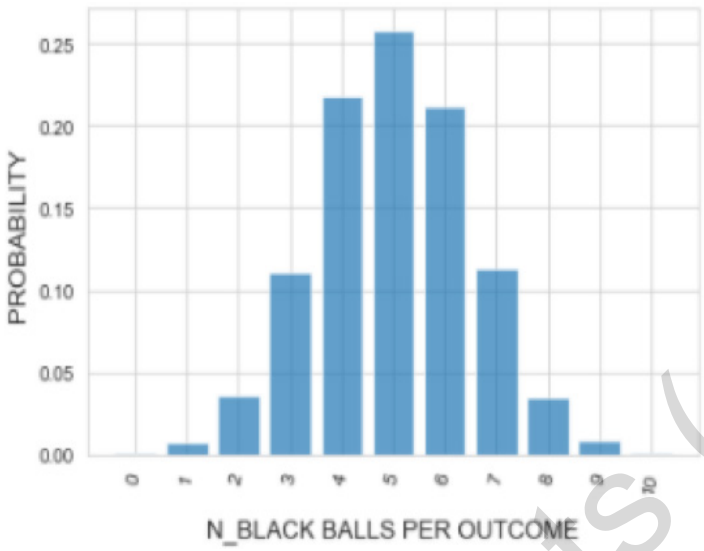
N is the population size

K is the number of success states in the population

n is the number of trials per outcome

x is the number of successes

```
pop_size, n_success_states_in_pop = 100, 50
n_trial_per_outcome, n_outcomes = 10, 10000
xlabel = 'N_BLACK BALLS PER OUTCOME'

random_var = fn_hyper_geometric_dist(pop_size, n_success_states_in_pop,
                                     n_trial_per_outcome, n_outcomes)
fn_plot_PMF(random_var, xlabel)
```



THE POISSON DISTRIBUTION:

The Poisson distribution deals with the frequency with which an event occurs in a specific interval of time, given that the probability of the event occurring is constant. It has the element of time associated with it.

Imagine that for the coin toss example used to describe the binomial distribution, the coins are tossed at a constant rate. Then the probability distribution of the number of heads per minute will be a Poisson distribution. Note that the probability of heads occurring is constant, ie: P(H) = 0.5.

The poisson distribution requires knowing how often the event occurs within a specific time period. The graph of a Poisson distribution plots the number of times an event occurs in a standard interval of time and the probability for each one.

Some other examples of stochastic processes following a Poisson distribution are:

- The number of male babies born per week in a large hospital
- The number of defective parts produced by an assembly line per week,etc.

The Poisson distribution is mathematically represented as: **$X \sim Po(\lambda)$**

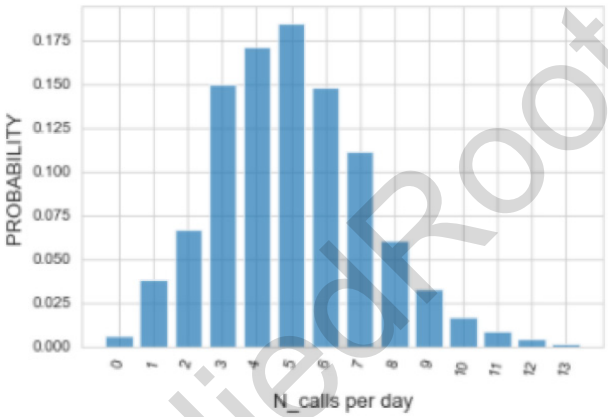
Where λ is the expected or average frequency of success events and is called the rate parameter.

The PMF of a Poisson distribution is defined as shown below:

$$P(X = x) = \frac{e^{-\lambda}\lambda^{x}}{x!}$$ where x = 0, 1, 2, 3.....

The Poisson distribution is used to model situations where the general rate of occurrence of the event in question is known and we now want to know the probability of the same event occurring at a different rate of occurrence. For example consider a call center which gets an average of 5 calls per hour. Say we now want to know what is the probability for getting 10 calls per hour. This information can be obtained using the Poisson distribution.

```
1  expected_occurrence_rate = 5 # Calls per hour
2  sample_size = 1000
3
4  random_var = fn_poisson_dist(expected_occurrence_rate, sample_size)
5  xlabel = 'N_calls per day'
6  fn_plot_PMF(random_var, xlabel)
```



THE EXPONENTIAL DISTRIBUTION:

The exponential distribution is closely related to the Poisson distribution. If the Poisson distribution represents or models the number of successful events per unit time, then the exponential distribution models the time period between 2 consecutive successful events that follow a Poisson distribution. Since the time periods are continuous, this distribution is a continuous distribution.

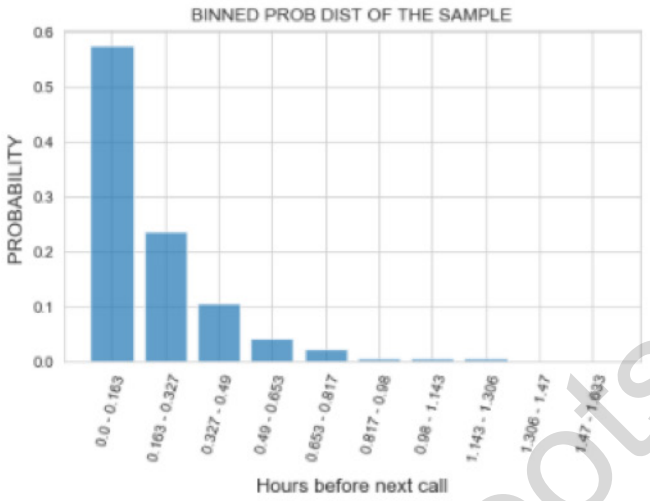
The exponential distribution is mathematically expressed as:

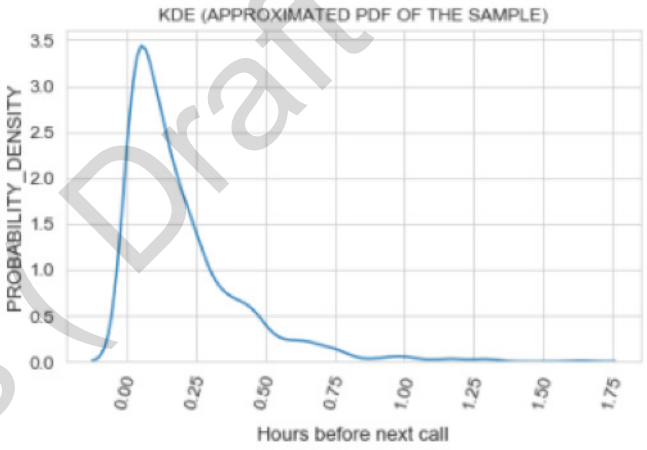
X ~ E(β) where β = 1 / β

The PDF of an exponential distribution is defined as shown below:

$$P(X > x) = e^{-x\lambda} \quad \text{where } x \text{ is } > 0$$

```
1  expected_occurrence_rate = 5 # Calls per hour
2  sample_size = 1000
3
4  random_var = fn_exponential_dist(expected_occurrence_rate, sample_size)
5  xlabel = 'Hours before next call'
6
7  fn_plot_PDF(random_var, xlabel)
```





THE NORMAL OR GAUSSIAN DISTRIBUTION:

Normal Distributions are used to represent stochastic processes, whose outcomes are continuous and exhibit "centrality". The concentration of the frequency distribution is maximum near the central or mean value and it reduces symmetrically as we move away from it.

The normal distribution is mathematically represented as:
$X \sim N(\mu, \sigma)$

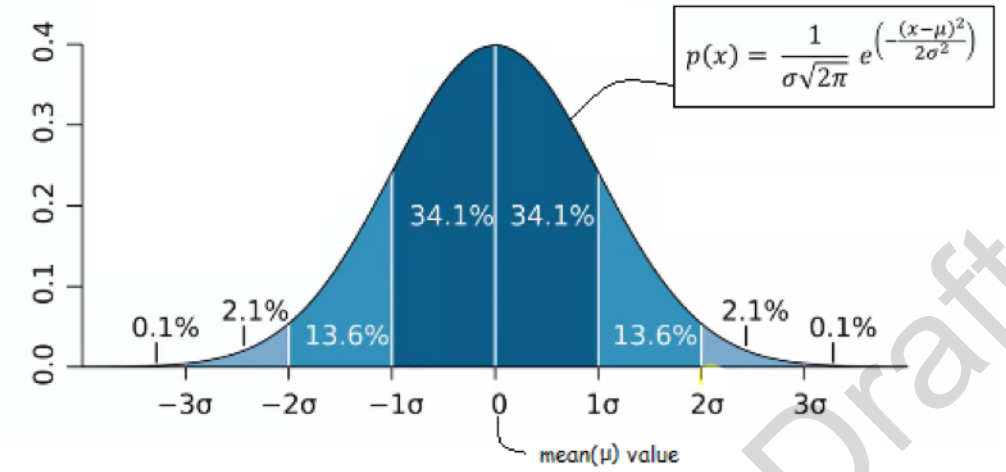
Where:
μ is the mean value, also as the location parameter
σ is the standard deviation, also known as the scale parameter

As explained earlier, the PDF of a normal distribution is defined as:

$$P(X = x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

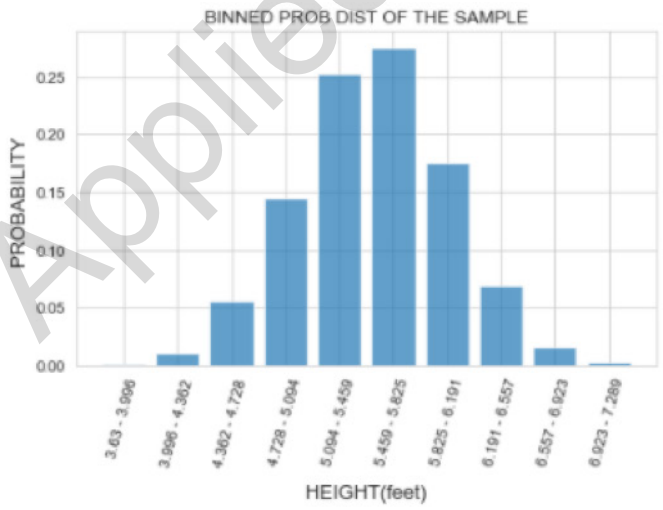
The values of a normal distribution are always distributed as shown below. Approximately 68% of all values lie

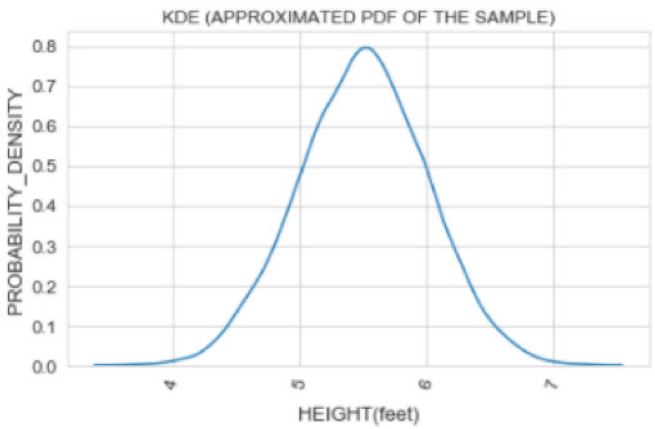
within one standard deviation of the mean, approximately 95.5% of all values lie within two standard deviations of the mean and so on.



Shown below is the Heights of humans modelled by a Normal distribution of mean 5.5 and standard deviation of 0.5:

```
mu, sd, sample_size = 5.5, 0.5, 10000

random_var = fn_normal_dist(mu, sd, sample_size)
xlabel =  'HEIGHT(feet)'

fn_plot_PDF(random_var, xlabel)
```





THE LOG NORMAL DISTRIBUTION:

A random variable X is said to be lognormally distributed if log of X is normally distributed. Random variables which are log-normally distributed can contain only positive real values. This distribution is used to model Random Variables that are asymmetric around their mean values and are heavy tailed on one side as shown in the plot below.

The lognormal distribution is mathematical represented as:
ln(X) ~ N(μ, σ)

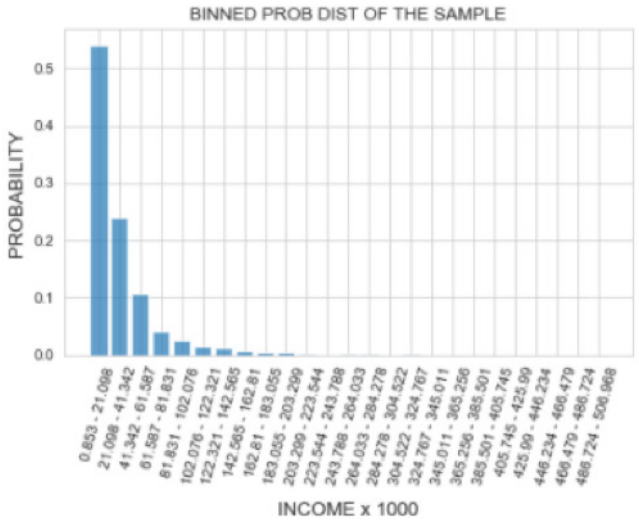
The PDF of a lognormal distribution is defined as:

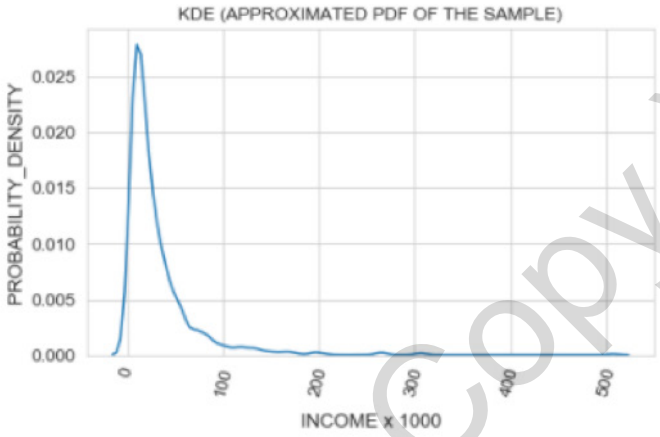
$$P(X = x) = \frac{1}{\sigma x \sqrt{2\pi}} e^{-\frac{(\ln x - \mu)^2}{2\sigma^2}}$$

Some other examples of Random Variables that are lognormally distributed are:
- Distribution of Incomes
- Size of human settlements
- Values of market stock
- Weights of humans, etc.

Consider the distribution of incomes of all the people in a country. Let's say that most of the people have an income level in the range of around 3000 units of currency. As the income level decreases, the corresponding percentage of people decreases and reaches the minimum at zero income. On the other side of the spectrum, as the income increases beyond 3000 units, the corresponding percentage of people decreases at a slower rate and keeps stretching into a long tail representing a small percentage of people who have extremely high incomes.

```
mu, sd = 3, 1 # mean and standard deviation
sample_size = 30000

random_var = fn_lognormal_dist(mu, sd, sample_size)
xlabel =  'INCOME x 1000'

fn_plot_PDF(random_var, xlabel, n_bins = 25)
```





TRANSFORMATION OF SAMPLED DATA TO GAUSSIAN:

Data transformation is the replacement of a Random Variable by a function of that Random Variable: for example, replacing a Random Variable X by the square root of X or the logarithm of X. Data transformations can be used to modify the shape of the distribution of the Random Variable.

A normal or Gaussian distribution is regarded as ideal and it is often assumed by many statistical methods. This is because it has unique properties that make it very handy to work with. Its defining parameters, the mean and standard deviation are very intuitive and easy to understand. It is symmetric and the distribution of data with respect to its standard deviation is always the same (see Fig:1.12). Hence it is often the practice that data is transformed to normal whenever possible.

Very often sampled data may be almost normal and hence it may be beneficial to transform it to an ideal normal distribution. Sometimes the data might have a non-normal distribution which could be transformed into a Normal distribution given the appropriate Transformation function.

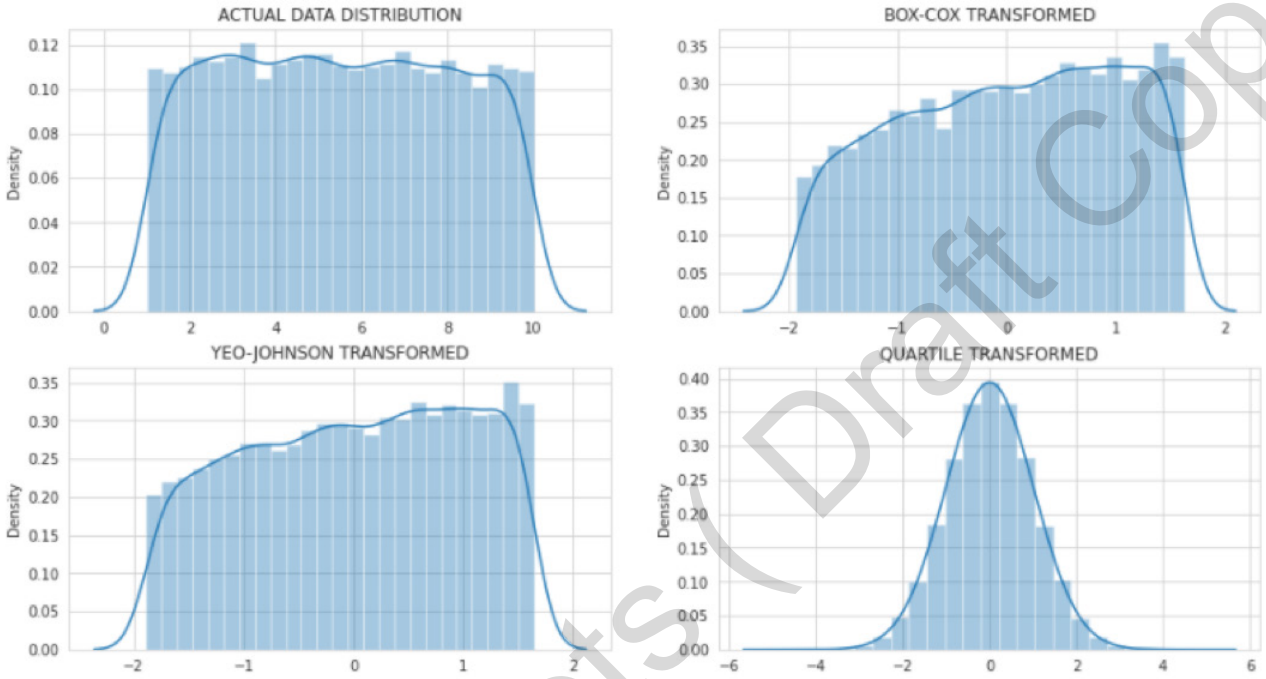
Three of the commonly used Transformation techniques are:

1. The Box Cox Transform
2. The Yeo Johnson Transform
3. The Quantile Transform

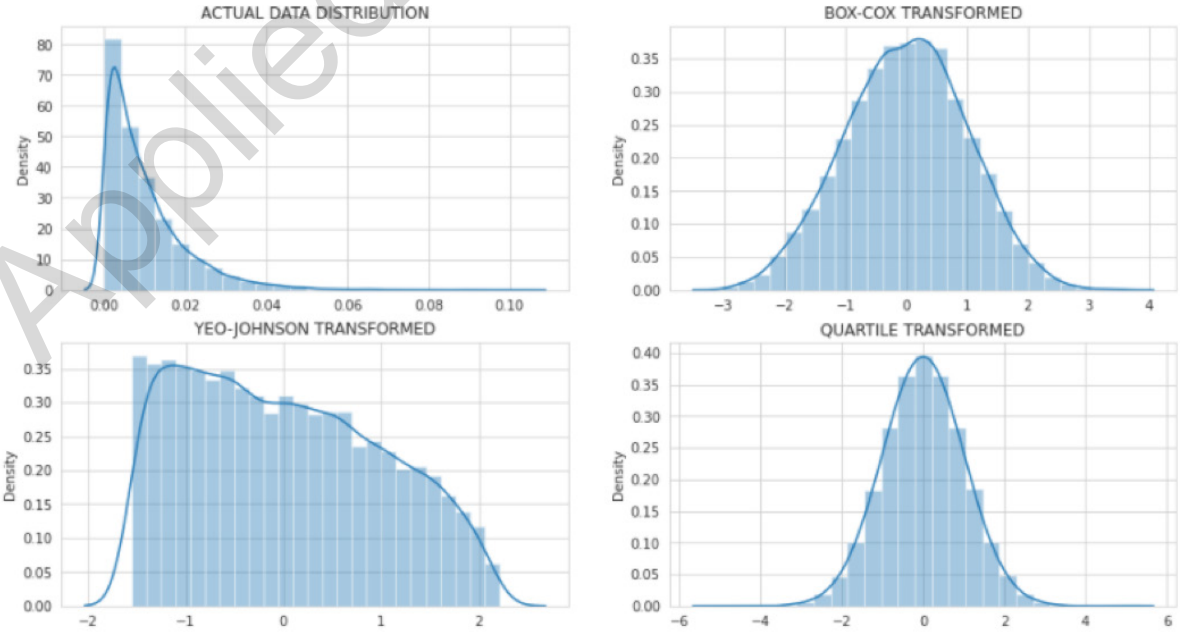
The Box Cox transform can only be applied to non negative Random Variables. Each of these techniques use different transformation criteria and each have limitations within which they work optimally. It is good practice to try all three and choose the one that works best.

Below are some examples of these transformations using the scikit learn package in python:

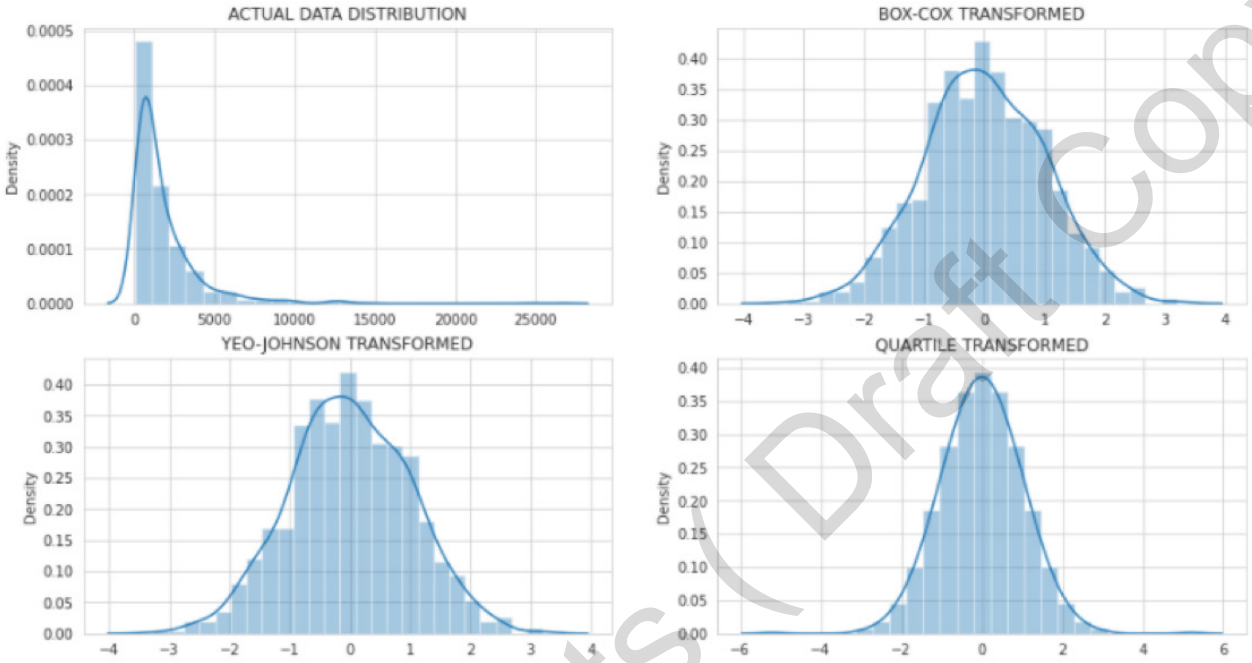
```
min_val, max_val, sample_size = 1, 10, 10000
random_var = fn_uniform(min_val, max_val, sample_size, discrete = False)

fn_transform_to_normal(random_var)
```




```
expected_occurrence_rate, sample_size = 100, 10000
random_var = fn_exponential_dist(expected_occurrence_rate, sample_size)

fn_transform_to_normal(random_var)
```



```
1  mu, sd, sample_size = 7, 1, 10000
2  random_var = fn_lognormal_dist(mu, sd, sample_size)
3
4  fn_transform_to_normal(random_var)
```



3. EXPLORATORY DATA ANALYSIS (EDA)

EXPLORATORY DATA ANALYSIS (EDA)

As the name suggests, exploratory data analysis (EDA) deals with the initial analysis of data. EDA is basically more exploratory in nature and not concerned with any form of inference or prediction. It can be classified into 2 main categories: Univariate and Multivariate data analysis

We will use the "Heights, weights & gender" dataset shown below as an example to explain the basics of EDA..

HEIGHTS & WEIGHTS DATASET

The "Heights, weights & gender" dataset shown below contains 300 records of heights, weights and gender adult humans.

```
df = pd.read_csv('hts_wts_gender.csv')

df.head()
```

	gender	height	weight
0	female	5.362289	60.988907
1	female	5.230310	60.982442
2	male	5.234430	69.533642
3	male	5.865660	79.757386
4	male	5.664446	83.816351

UNIVARIATE DATA AND MULTIVARIATE DATA:

In the dataset just introduced above, the collection of all the heights in the dataset forms one Random Variable, similarly the collection of all the weights forms another Random Variable. All data can be classified into 2 main categories, based on the number of characteristics or attributes used, to describe the data. Datasets that contain only one Random Variable are said to be Univariate and those that contain more than one Random Variable are said to be Multivariate.

UNIVARIATE DATA ANALYSIS:

All the example data explored prior to the heights, weights & gender' dataset involved univariate data. Univariate data analysis is the simplest form of data analysis - it deals with only one random variable at a time. Univariate data analysis doesn't concern itself with the relationships that the Random Variable being examined might have, with any other Random Variable. It basically deals with exploring and defining the frequency distribution of the Random Variable by:

1. Summarizing its maximum, minimum, mean, median, mode, standard deviation and inter quartile range.
2. Using visualization techniques like Histograms and other variations of it.
3. Estimating its PMF/PDF if possible.

Most of task 1 described above is easily achieved using the Pandas library as shown below.

```
df.describe()
```

	height	weight
count	300.000000	300.000000
mean	5.533202	73.305333
std	0.304623	14.130191
min	4.815568	43.321817
25%	5.310336	62.272749
50%	5.536692	72.194269
75%	5.776933	84.677926
max	6.353281	106.516913

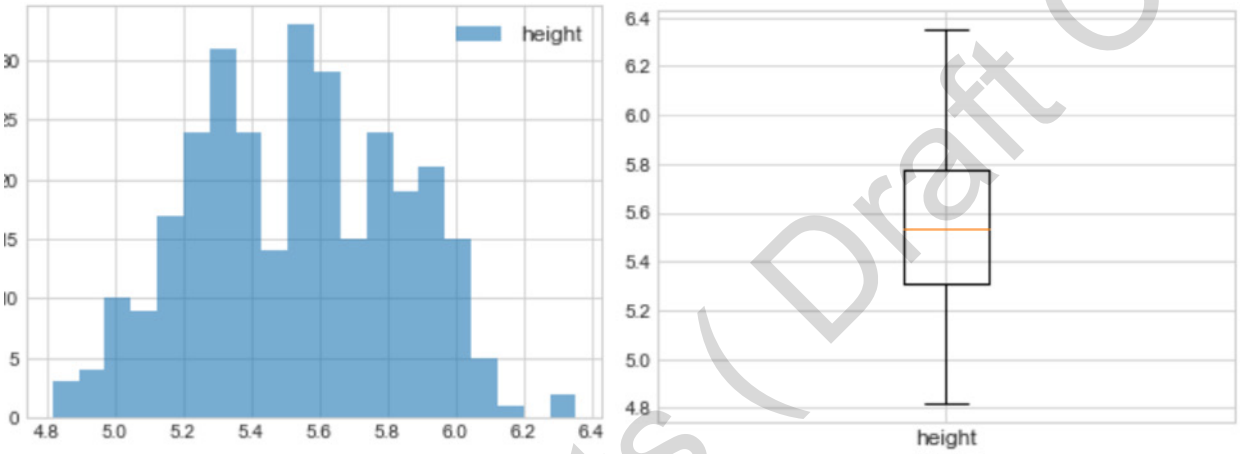
HISTOGRAMS:

Shown above is the function **fn_distribution**. It takes in a Random Variable (or list of random variables) as its argument and plots its histogram and box distribution plots.

```
def fn_distribution(list0_rvs, list0_rv_names = None):

    if list0_rv_names == None:
        list0_rv_names = [str(i) for i in range(len(list0_rvs))]

    kwargs = dict(alpha = 0.6, bins = 20)
    plt.figure(figsize = (12, 4))

    plt.subplot(1,2,1)
    for idx, rv in enumerate(list0_rvs):
        plt.hist(rv, label = list0_rv_names[idx],  **kwargs)
        plt.legend(fontsize = 12)

    plt.subplot(1,2,2)
    plt.boxplot(list0_rvs, labels = list0_rv_names)
    plt.xticks(fontsize = 12)
```

Consider the random variable "**heights**" in the DataFrame df containing heights, weights and gender data. We can understand its distribution by visualizing it using a histogram and box plot as shown below.

```
rv_ = df.height.values
list0_rvs = [rv_]
list0_rv_names = ['height']

fn_distribution(list0_rvs, list0_rv_names=list0_rv_names)
```

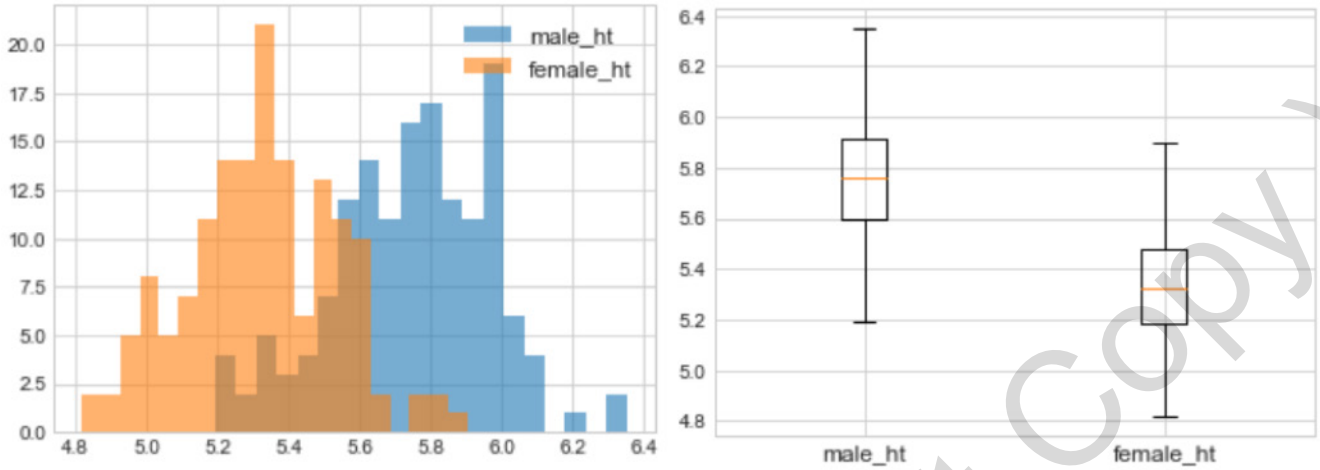


From the left plot shown above, we see that male_heights seems to follow a normal distribution. From the right plot we see that the middle 50% of the heights approximately lies within the range 5.3 to 5.8 and the mean male height is around 5.5.

We can get a deeper insight by separately visualizing the categories that form this distribution. This is demonstrated in the code shown below.

```
df_male = df[df.gender == 'male']
df_female = df[df.gender == 'female']

rv1 = df_male.height
rv2 = df_female.height
```

```
list0_rvs = [rv1, rv2]
list0_rv_names = ['male_ht', 'female_ht']

fn_distribution(list0_rvs, list0_rv_names=list0_rv_names)
```



MULTIVARIATE DATA ANALYSIS:

Multivariate data analysis deals with data containing two or more Random Variables. It is mainly concerned with detecting and defining the relationships that might exist between these Random Variables.

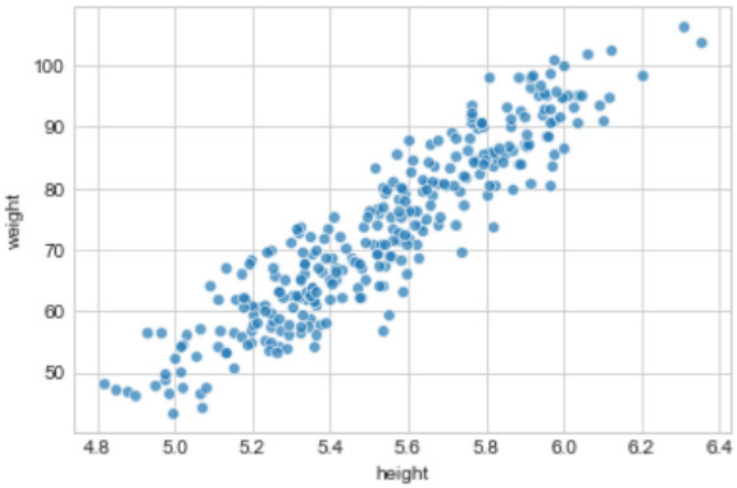
VISUALIZING BIVARIATE DATA:

Bivariate Data specifically refers to Multivariate data with 2 Random Variables. In univariate analysis performed earlier, we analysed single random variables without any other consideration. But one can get deeper insight into the data as a whole, by comparing the distributions of related random variables. The code below demonstrates this. We use the same function fn_distribution to do this.

.Just creating a scatter plot of one variable against another can instantly give us a visualization of the correlation between two Random Variables.

```
def fn_scatter(df, x_col, y_col, cat_col = None):
    import seaborn as sns
    sns.scatterplot(x = x_col, y = y_col,
                    data = df, hue = cat_col, alpha = 0.7)
    plt.show()
```

```
df_ =  df.sample(300)
x_col =  'height'
y_col =  'weight'

fn_scatter(df_, x_col, y_col, cat_col = None)
```




```
fn_scatter(df_, x_col, y_col, cat_col = 'gender')
```

From the plots above, we can say that the height and weight random variables are positively correlated

(or in other words weight values generally increase as height increases).

4. CORRELATION

CORRELATION

Correlation is the technical term used to describe the joint variability between Random Variables (Random Variables that vary together are said to be correlated). Correlation is said to exist between 2 Random Variables if:

1. The greater values of one variable generally correspond with the greater values of the other variable, and the same correspondence holds true for the lesser values .

2. The greater values of one variable generally correspond to the lesser values of the other and the same correspondence holds true for the lesser values.

In the first case the correlation is said to be positive and in the latter case, it is said to be negative.

Correlation does not imply Causation. Just because 2 Random Variables are correlated it does not mean one of them is the cause of the other.

MEASURES OF CORRELATION:

1. COVARIANCE

Covariance is a measure of the correlation between two random variables. Covariance is mathematically defined as shown below:

$$\text{Cov}(X,Y) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})$$

where: $\overline{x}$ is the mean of X &
$\overline{y}$ is the mean of Y

It is the average of the product of the differences of each random variable with their own mean. Covariance can be expressed in python code as shown below:

```
def fn_covariance(rv1, rv2):

    rv1 = np.array([*rv1])
    rv2 = np.array([*rv2])

    diff_1 = rv1 - rv1.mean()
    diff_2 = rv2 - rv2.mean()

    cov = (diff_1 * diff_2).mean()
    return cov
```

```
rv1_, rv2_ = df.height, df.weight

fn_covariance(rv1_, rv2_)
```

```
3.9692032634754755
```

The outcome of the code above shows that there is a positive correlation observed between the heights & weights variable in the dataset, but the value of 3.969 does not provide any information about the degree of correlation because we have no information about the upper bound.

Consider the Random Variables shown in the code below, observe the covariances for various combinations of the two Random Variables:

```
rv1_= [5, 4, 3, 2, 1]
rv2_ = [500, 400, 300, 200, 100]

fn_covariance(rv1_, rv2_)
```

```
200.0
```

```
fn_covariance(rv1_, rv1_)
```

```
2.0
```

```
fn_covariance(rv1_, rv1_[::-1])
```

```
-2.0
```

```
fn_covariance(rv2_, rv2_)
```

```
20000.0
```

Covariance berween rv1_ & rv2_

Covariance of rv1_ with itself

Covariance of rv1_, with its reversed self

Covariance of rv2_ with itself

It can be seen from the results above that covariance has utility in detecting correlation, and telling whether the correlation is positive or negative, but it does not provide interpretability in terms of degree of correlation.

2. PEARSON'S CORRELATION COEFFICIENT(PCC):

The Pearson's Correlation Coefficient is a measure of correlation which overcomes the limitation of Covariance as presented earlier. It is simply the Covariance, normalized by the product of the standard deviations of the 2 Random Variables being examined.

$$Corr(X, Y) = Cov(X, Y) / \sigma_x \sigma_y$$

Covariance normalized by the product of the standard deviations of individual random variables

The values of the PCC lie within the interval [-1, 1]. Due to this property (ie: having well defined upper & lower bounds), the magnitude of the PCC is indicative of the degree of correlation between the Random Variables. Zero magnitude signifies no correlation, 0.5 implies medium and a magnitude of one signifies maximum correlation. The values can be positive or negative signifying positive and negative correlation respectively.

PCC can be expressed in python code as shown below:

```
def fn_PCC(rv1, rv2):

    covariance = fn_covariance(rv1, rv2)

    rv1 = np.array([*rv1])
    rv2 = np.array([*rv2])

    std_dev1 = rv1.std()
    std_dev2 = rv2.std()

    PCC = covariance/(std_dev1 * std_dev2)

    return PCC
```

Consider the following code, where we compute the PCC for different combinations of the Random Variables:

```
rv1_, rv2_ = df.height.values, df.weight.values

fn_PCC(rv1_, rv2_)
```

```
0.9252157489817202
```

```
fn_PCC(rv1_, rv1_)
```

```
0.9999999999999999
```

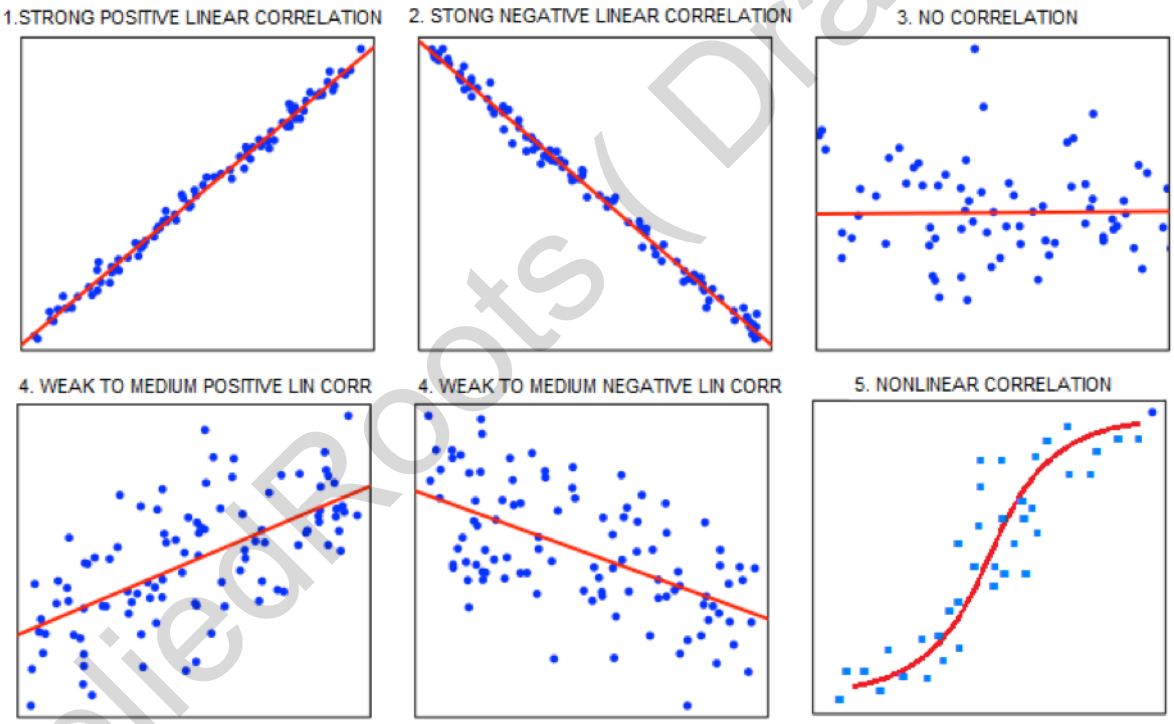
```
fn_PCC(rv2_, rv2_)
```

```
1.0
```

The results above show:

1. Good positive correlation between the heights & weights Random Variables.
2. Maximum correlation between heights and itself
3. Maximum correlation between weights and itself .

PCC assumes that the correlation between the Random Variables if any, is linear, ie: The plot of the two Random Variables in question, can generally be “fit” using a straight line. Consider the examples in the image below. PCC will not be effective when applied to a pair of Random Variables that produce a plot like the one shown as number 5. For Random Variables such as these, another correlation coefficient called the Spearman’s Ranked Correlation Coefficient is used.



3. SPEARMAN’S RANKED CORRELATION COEFFICIENT (SRCC):

SRCC uses “ranked scores” of the Random Variables in conjunction with PCC to determine correlation. In other words, it describes the linear correlation between the ranks of the Random Variables. Doing this has the effect of detecting if any monotonic non decreasing correlation exists between two variables. The correlation is considered high when the variables have a similar rank order, and low when variables have a dissimilar rank order.

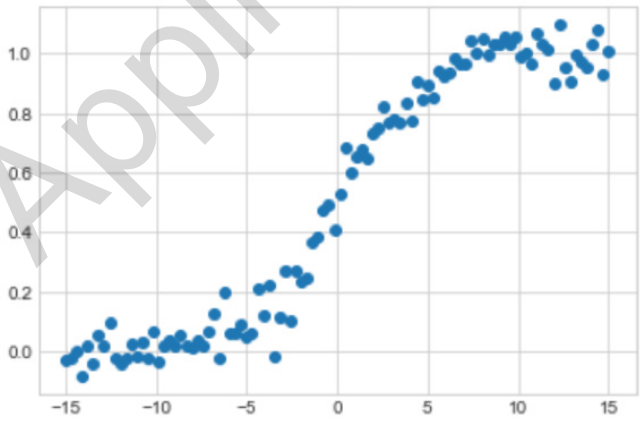
A monotonic non decreasing correlation is such that, for any 2 Random Variables X & Y, if x2 is greater than x1, then y2 will be greater than or equal to y1. Plot 5 shown above is an example of this kind of correlation. The SRCC is best suited for data with monotonic non decreasing correlation and may not fare well with other kinds of non linear correlations. The values of Spearman's correlation lie within the interval [-1, 1] and have the same interpretations as PCC

Below is SRCC implemented in python code:.

```
def fn_SRCC(rv1, rv2):

    from scipy import stats

    rv1 = np.array([*rv1])
    rv2 = np.array([*rv2])

    # ouput corresponding rank of each element of the RV
    rank_rv_1 = stats.rankdata(rv1, method='ordinal')
    rank_rv_2 = stats.rankdata(rv2, method='ordinal')

    SRCC = fn_PCC(rank_rv_1, rank_rv_2)
    return SRCC
```

Shown below is an example of two Random Variables, having a monotonic non decreasing correlation and the SRCC computed for the same:

```
rv1_ = np.linspace(-15, 15, 100)

# MONOTONOUS NON DECREASING RELATION TO random_var_1:
rv2_ = 1 / (1 + np.exp(-rv1_ * 0.5)) + np.random.normal(0, 0.05, 100)

plt.scatter(rv1_, rv2_)
plt.show()
```




```
fn_SRCC(rv1_, rv2_)
```

```
0.9443384338433843
```

4. Correlation Between Numeric And Categorical Random Variables (ANOVA):

Consider the height random Variable in the heights, weights and gender dataset, say the variance of heights is some value **"var_heights"**. If we separate the heights random variable into male and female, the variance in each group will be smaller, which suggests that there is a relationship between height and gender. The ANOVA (Analysis of Variance) test in essence uses this principle to detect correlation.

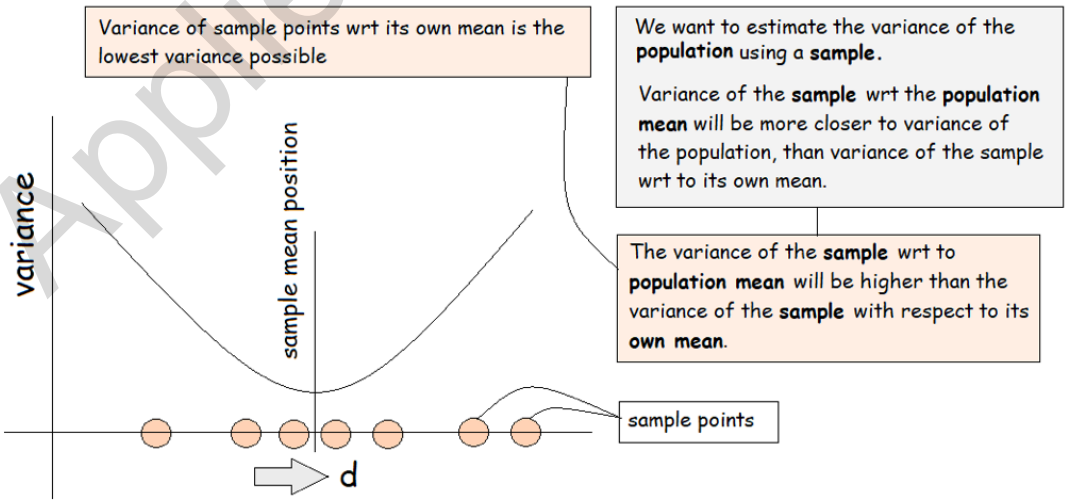
Variance & Degrees Of Freedom:

Degrees of freedom can be thought of as the number of values that are free to vary as you estimate a population parameter (ie: Population mean & variance). It indicates how much independent information goes into computation.

Say, we want to compute the mean of a sample containing 'n' numbers. Each of the n values are independent of each other and so have the freedom to be any value. Say we know the values of n-1 numbers and only the last value is unknown. The knowledge of all other values does not constrain the value of the nth number in any way. In other words we need to know the values of all n samples to compute the mean.

Now say we know the mean of that same sample. Knowing the mean creates a constraint on the value of nth number. In other words if we know the value of the mean and also the values of n-1 numbers in the sample, the value of the nth number becomes "fixed". Therefore while computing the variance of the sample, assuming we already know the mean of the sample, we need to know the values of only n-1 numbers and not all n values. In general the information that goes into the computing of any population parameter that utilizes the mean (ex: variance), will have only n-1 degrees of freedom.

Degrees of freedom is used in cases where we want to estimate the variance of the **population** using a **sample** from the population. Consider the information contained in the image below.



As can be understood from the image shown above:

1. The variance of the sample with respect to the population's mean will generally always be larger than the variance of the sample with respect to its own mean.
2. Therefore if one wants to get a more accurate estimation of the population variance one has to adjust for this increase in variance.

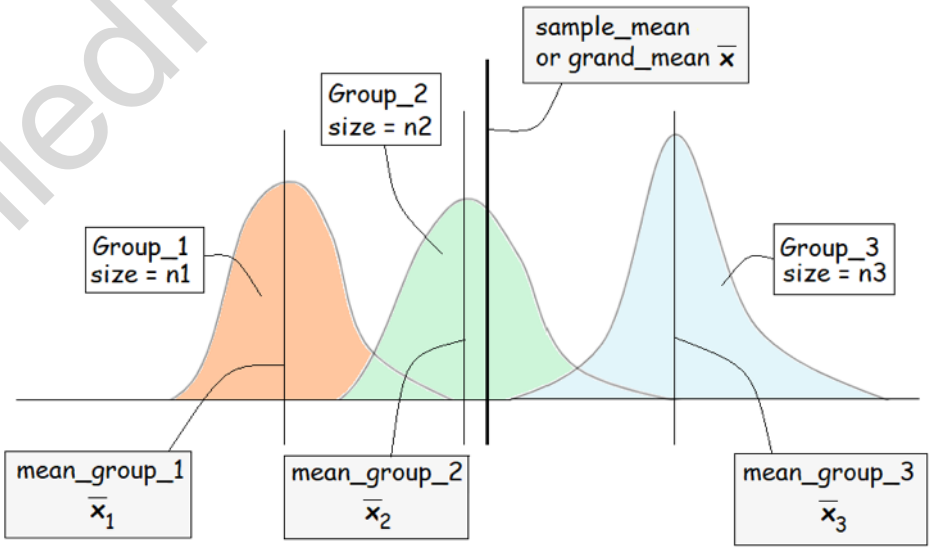
This is accomplished by dividing the sum of the squared distances from the sample mean, with the degrees of freedom (dof), instead of dividing it with the number of samples. This is demonstrated in the image below.

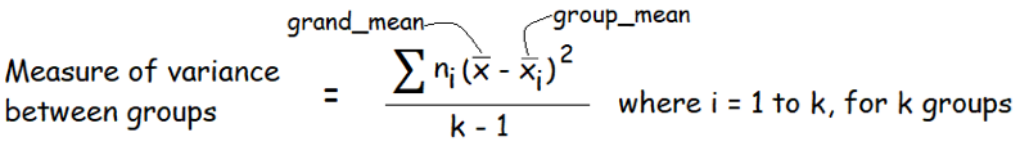
```
sum of squared distances = sum([(sample_mean - point)**2 for point in sample])
variance of sample = sum of squared distances/sample_size
Estimate of Population variance = sum of squared distances/dof
                  degrees of freedom = sample_size - 1
```

Anova F Statistic:

To understand the F statistic one has to understand the following two measures:

1. Measure of the variance in the data due to the variances between each group. This is the variance of the sample's population if each point in the sample were located at the same location as the mean of the group it belonged to.





The function shown below computes the measure shown in the image above.

```
def fn_var_between(sample_mean, list0_group_means, list0_group_sizes):

    iterator = zip(list0_group_means, list0_group_sizes)
    sq_dists = sum([((mean_ - sample_mean)**2)*n for mean_, n in iterator])
    var_between = sq_dists/(len(list0_group_means) - 1)

    return var_between
```

2. Measure of the variance in the data due to the individual variances within each group. This is the average of the variances of each group.

The function shown below computes this measure

```
def fn_var_within(list0_group_rvs):

    def fn_variance(rv):
        rv = np.array([*rv])
        mean_ = rv.mean()
        sq_dists = ((rv - mean_)**2).sum()
        var = sq_dists/(len(rv)-1)
        return var

    var = 0
    for rv_ in list0_group_rvs:
        variance = fn_variance(rv_)
        var+=variance

    avg_var = var/len(list0_group_rvs)
    return avg_var
```

The anova F_statistic is the ratio of the “between” variance measure with the “within” variance measure. This ratio will be close to one if there is no correlation between the categorical random variable (ex: gender) and the numerical random variable (ex: height).

The code shown below uses the two functions shown above to compute the F statistics correlation measure between the heights and gender random variables in the heights, weights and gender dataset.

```
rv = df.height
sample_mean = rv.mean()
list0_group_rvs = [df[df.gender == cat].height.values for cat in df.gender.unique()]
list0_group_means = [i.mean() for i in list0_cat_rvs]
list0_group_sizes = [i.size for i in list0_cat_rvs]


var_between = fn_var_between(sample_mean, list0_group_means, list0_group_sizes)
var_within = fn_var_within(list0_group_rvs)

fstat = var_between/var_within
fstat
```

```
280.8749615267523
```

As can be seen from the output of the code above the F statistic is 280.87. This implies that there is correlation between heights and gender random variables.

The function shown below creates random groups/categories within a random variable and then computes the F_statistic. It does this for **n_iter** number of times and returns the mean F_statistic.

```
def fn_no_correlation(rv, sample_size, n_groups, n_iters = 100):

    list0_fstats = []
    for iter in range(n_iters):

        list0_group_rvs = [df.sample(sample_size).height.values for cat in range(n_groups)]
        list0_group_means = [i.mean() for i in list0_group_rvs]
        list0_group_sizes = [i.size for i in list0_group_rvs]

        var_between = fn_var_between(sample_mean, list0_group_means, list0_group_sizes)
        var_within = fn_var_within(list0_group_rvs)
        list0_fstats.append(var_between/var_within)

    return sum(list0_fstats)/len(list0_fstats)
```

Shown in the code below, we sample 2 groups randomly from the heights random variable to represent the male and female genders and we compute the F statistic for 1000 such sampling and check the value of the F statistic.

```
rv_ = df.height
sample_size = int(len(rv)/2)
n_samples = 2

fstat_random = fn_no_correlation(rv_, sample_size, n_samples, n_iters = 1000)
fstat_random
```

```
0.9883614887219287
```

As can be seen from the output of the code above, the mean f_statistic is close to one.

5. INFERENTIAL STATISTICS

INFERENTIAL STATISTICS

JOINT & CONDITIONAL PROBABILITIES:

In the Heights & Weights dataset, each value in one Random Variable is uniquely paired with a value from the other Random variable, thus describing each observation in the dataset. In other words, each pair of height & weight describes a specific person and so the position of each Random Variable is fixed with respect to the other. The same logic applies to all datasets containing more than 2 variables.

Joint probability refers to the probability of two events/values occurring simultaneously The Heights & Weights dataset contains continuous random variables, joint probability is more easily understood using discrete random variables. For example, consider the outcome of two six faced dies thrown together. In this case, the outcomes of each dice form the random variables (say: **outcome_dice1 & outcome_dice2**). Each observation/row consists of a pair of outcomes of values between 1 and 6. This is demonstrated in the code shown below.

```
n_throws = 1000

data = np.random.randint(1, 7, [n_throws, 2])
df_dice = pd.DataFrame(data, columns = ['outcome_dice1', 'outcome_dice2'])
df_dice.sample(5)
```

	outcome_dice1	outcome_dice2
986	4	5
616	1	5
19	2	5
123	5	6
200	3	4

Then the probability of both dice showing the number 6 is a joint probability and it is represented as:

P(die 1 = 6, die 2 = 6) or P(6, 6) or generically: P(event_A, event_B).

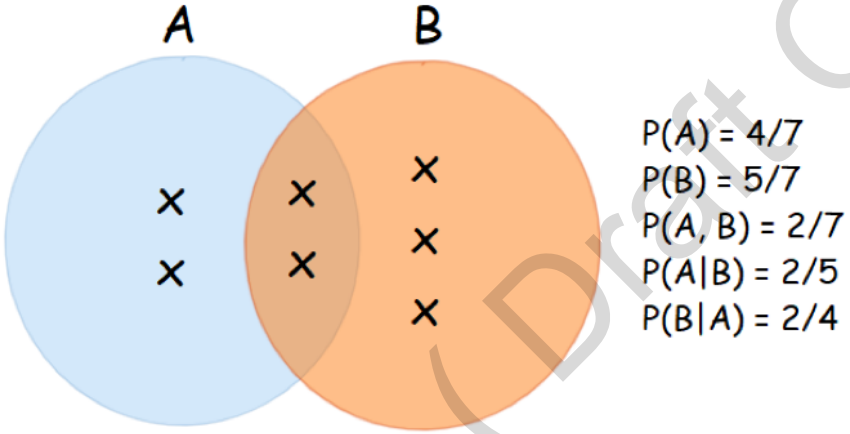
Conditional Probability refers to the probability of an event occurring, given the prior occurrence of another event. For example, the probability that **outcome_dice1 = 5**, given that **outcome_dice2 = 1** Here we are "conditioning" the probability of one event (outcome_dice1 = 5) based on the existence/occurrence of some other event/condition (outcome_dice2 = 1). Conditional probability is as shown below.

P(outcome_dice1 = 5 | outcome_dice2 = 1)

In the same same lieu, the probability of a person to have black hair given that they are Indian would be represented as:

P(Black_Hair | Indian)

Joint and conditional probabilities can be intuitively understood by examining the image below.



INDEPENDENT AND DEPENDENT EVENTS:

Two events are dependent if the outcome of the first event affects the outcome of the second event, so that the probability of the second event is changed. Example : Suppose we have 5 black balls and 5 white balls in a bag. We pull out one ball, which may be black or white. What is the probability that the second ball will be black? It depends on the outcome of the first draw. If the first ball was black, then the bag is left with 4 black balls out of 9 so the probability of drawing a black ball on the second draw is 4/9 . But if the first ball we pull out is white, then there are still 5 black balls in the bag and the probability of pulling a black ball out of the bag is 5/9.

Independent events are events whose outcomes are not influenced by the outcome of another event. A coin toss is an example of an independent event. The probability of heads remains 50% irrespective of the outcomes of the previous tosses.

An event A is said to be independent of B if:

P(A | B) = P(A) ie: The "condition" B has no effect on event A

CONDITIONAL INDEPENDENCE:

Consider the following situation: A box contains 5 balls. They could be a mixture of black and white balls or they could be all black. Say that we draw 2 balls one after the other. Let the possibility that :

1. The group of balls in the box are all black be "Event A".
2. The group of balls in the box being a mixture of black & white balls be "Event B"
3. The first ball drawn is black be "Event C"
4. The second ball drawn is black be "Event D".

Now the probability of D given event B is dependent on event C, but the probability of event D given event A will be independent of event C. That is:

P(D | C, A) = P(D | A)

The probability of event D given the joint occurrence of events C & A is the same as the probability of event D given just the occurrence of event A. The Occurrence of event A makes event D independent of event C. Here event D is said to be conditionally independent of event C.

MUTUALLY EXCLUSIVE EVENTS:

Two events A and B are said to be mutually exclusive events if only one of them can occur at any given moment. For example, given a coin toss, the events of heads and tails are mutually exclusive or in other words it is not possible for them to occur simultaneously. That is:

P(Heads, Tails | one coin) = 0 ie: The joint probability of heads & tails given a single coin is zero.

SUM AND PRODUCT RULES:

Sum Rule 1:

When two events, A & B, are mutually exclusive, the probability that A or B will occur is:
P(A or B) = P(A) + P(B)

Sum Rule 2:

When two events, A & B, are non-mutually exclusive, the probability that A or B will occur is:
P(A or B) = P(A) + P(B) - P(A, B)

Product Rule 1:

When two events A & B are independent events, the probability that A & B will occur is:
P(A, B) = P(A) x P(B)

Product Rule 2:

When two events A & B are dependent events, the probability that A & B will occur is:
P(A, B) = P(A) x P(B|A) = P(B) x P(A|B)

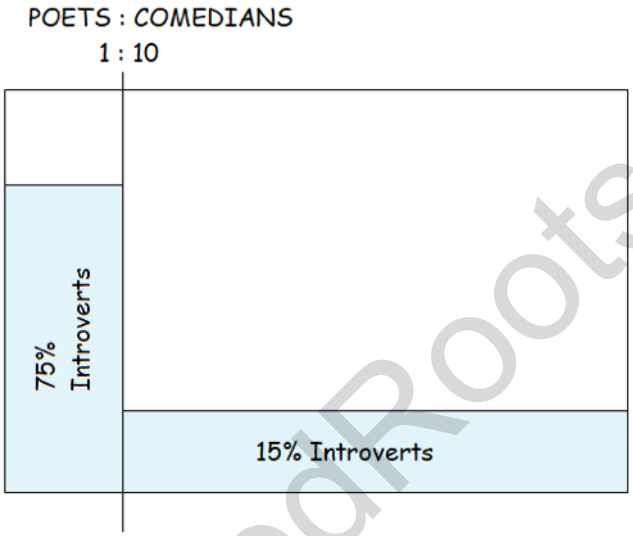
BAYES RULE:

The Bayes rule is a reframing of the product rule for dependent events just shown above. Shown below is the Bayes rule.

P(A|B) = P(B|A) x P(A) / P(B)

Consider the following example: Jill belongs to a "population" of poets and comedians. The ratio of poets to comedians is 1:10. It is known that 75% of poets and 15% of comedians are introverts. Jill is an introvert, what is the probability of her being a poet?

The above question can be visualized and understood as shown below:



1. P(poet) = 1/11 = 0.091
2. P(comedian) = 10/11 = 0.91
3. P(introvert|poet) = 0.75

Assuming size of population = 100
Total n_poets = 100 x 0.091 = 9.1
Total n_poet_introverts = 9.1 x 0.75 = 6.825

Total n_comedians = 100 x 0.91 = 91
Total n_comedian_introverts = 91 x 0.15 = 13.65

From the calculations shown above we can find the probability of a person in the population being an introvert as shown below:

Total n_introverts = Total n_poet_introvert + Total n_comedian_introverts
Total n_introverts = 20.475

P(introvert) = Total n_introverts/Population size = 20.475/100 = 0.205

Therefore the probability of an introverted person being a poet is:

P(poet|introvert) = P(introvert|poet) x P(poet)/P(introvert)
P(poet|introvert) = 0.75 x 0.091/0.205 = 0.333 (ie: 33.3%)

Therefore the probability of Jill being a poet is 33.3%

6. HYPOTHESIS TESTING

HYPOTHESIS TESTING

THE LAW OF LARGE NUMBERS:

The law of large numbers states that as the size (number of observations) of a Sample increases, its mean gets closer to the average of the whole population. In other words - the larger the size of the sample, the more representative it is of the Population it has been drawn from.

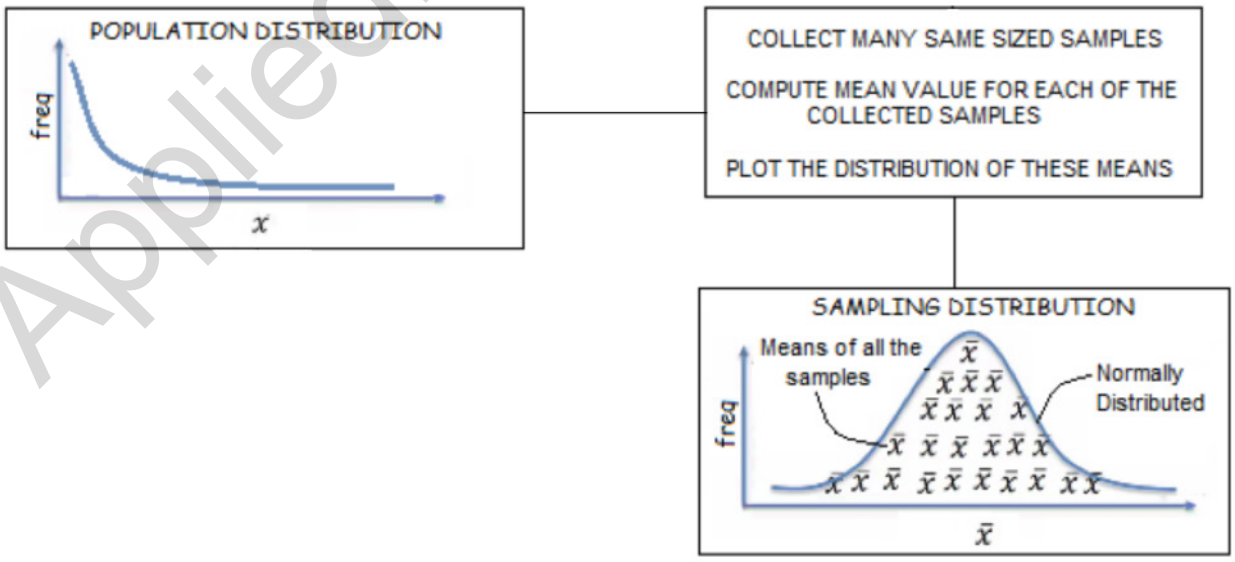
For example consider the coin toss example, we know that the expected mean is 0.5 (ie:P(H) = P(T) = 0.5). This may not be apparent in a sample consisting of 6 observations, we could have 4 heads and 2 tails. But as the number of tosses increases, the mean of the observations will tend towards 0.5.

THE CENTRAL LIMIT THEOREM (CLT):

While the Law of Large Numbers describes and deals with the "degree of similarity" of a single Sample to its parent Population, the Central Limit Theorem deals with multiple Samples and the distribution of the statistics derived from each sample.

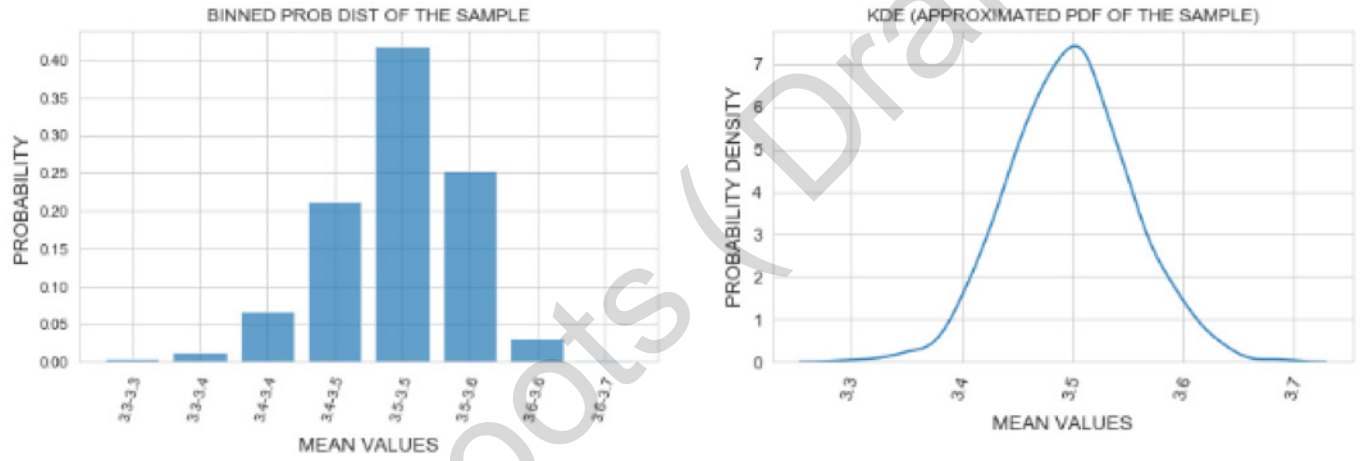
The Central Limit Theorem states that, given enough samples, the distribution of the means of the samples will be normally distributed, irrespective of the distribution of the parent population.

The Central Limit Theorem is the basis for many of the statistical tests we will be using. For example, we could make judgements or inferences about the similarity of two populations by analysing the distributions of the means of the samples collected from them.



Consider the simulation of a six faced die shown below. Here we collect 1000 samples, where each sample contains 1000 dice throws and we compute the mean of all of those 1000 samples. It can be seen that the PDF of the means, generally follows a normal distribution as stated by the Central Limit Theorem.

```
listO_means = []
min_val, max_val, sample_size, n_samples = 1, 6, 1000, 1000

for i in range(n_samples):
    random_var = fn_uniform(min_val, max_val, sample_size, discrete = True) #-- 6 faced die
    mean_val = np.array(random_var).mean()
    listO_means.append(mean_val)

xlabel, n_bins = 'MEAN VALUES', 10
fn_plot_PDF(listO_means, xlabel, n_bins)
```



SAMPLING DISTRIBUTION AND STANDARD ERROR:

Sampling Distribution is the statistical terminology used to denote the distribution of some statistic (ex: Mean) derived from multiple samples. In the case of the example shown above, the Sampling Distribution is the distribution of the means of the multiple samples of dice throws. The Sampling Distribution helps us gain more insight into a Population's distribution, by summarizing the statistics gained from examining multiple samples, instead of relying on a single sample.

The Standard Error is the standard deviation of a Sampling Distribution, it reflects the extent to which the measured statistic changes from sample to sample.

Note that the Sampling Distribution does not represent any sample, but instead it is representative of multiple samples, which were drawn from the same Population. The size of the Sampling Distribution will by definition, always be the same as the number of samples. As the size of the samples increase, the Standard Error decreases and the Sampling Distribution becomes more narrow.

HYPOTHESIS TESTING:

As mentioned earlier most of Statistics is about making inferences about the Population's parameters based on the statistics derived from the Sample. This method of making inferences about the population overcomes our inherent inability to perform measurements that are inclusive of the entire population. But this method also introduces a level of uncertainty since the Sample is in essence, an approximation of the Population and hence all statistical information gleaned from the Sample is a "Hypothesis" about the Population. Hypothesis testing is a Statistical methodology or framework that helps us test the validity of a new or alternate hypotheses with respect to the previous or "neutral" hypothesis.

For example consider the case of a 6 faced die. Say that after a certain amount of experience of using a particular die, we begin to doubt that it is biased. We want to know how probable it is that our doubts could be true. We do this using the Hypothesis Testing framework.

NULL & ALTERNATIVE HYPOTHESES:

The previous or "neutral" or "obvious" hypothesis is called the Null hypothesis. Generally the Null Hypothesis is the opposite of the "new claim" (ie: Alternative Hypothesis) being made and it has the theme of: "There is no difference" or "There is no change".

In the case of the six face die example, the Null Hypothesis would be that the die is unbiased. The Alternative Hypothesis would be that the die is biased.

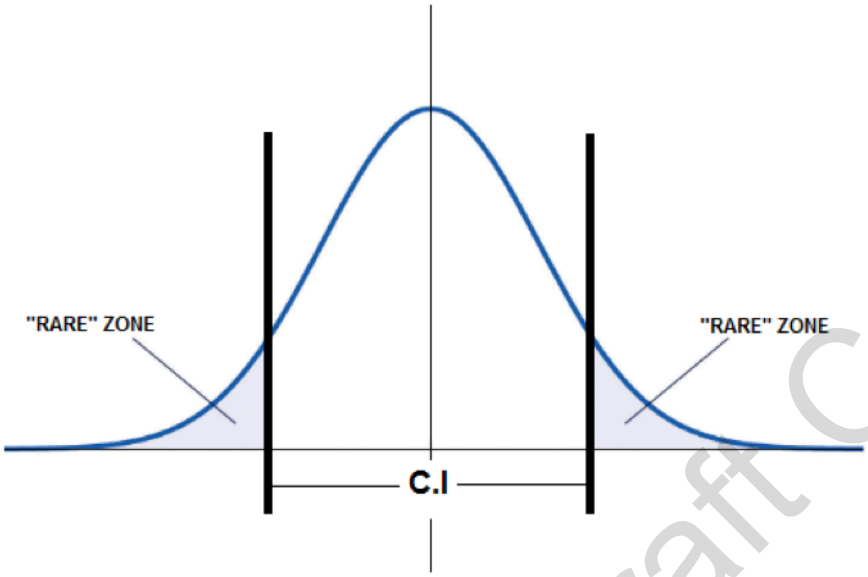
The Null Hypothesis is denoted as H0 & the Alternative Hypothesis as H1. The meaning of the Null & Alternative Hypotheses will become clearer in the examples that follow.

CONFIDENCE INTERVALS (CI):

Confidence intervals are intervals within the range of values of a Random Variable that correspond to a specified probability of occurrence. For example, we know that for a normally distributed Random Variable, approximately 95% of all values fall within 2 standard deviations on either side of the mean value. The interval that corresponds to this range is called the 95% Confidence Interval. In other words we can be 95% confident that, if we randomly pick any value from this distribution, it will lie within the 95% Confidence Interval. In the same vein we could define a 50% Confidence Interval or a 10% Confidence Interval.

SIGNIFICANCE LEVEL:

Significance level is a way to define the level of confidence we desire to have while trying to decide whether to accept or reject the Null Hypothesis. Its value is complementary to the confidence interval. That is: significance level = 1 minus the C.I value.



Consider the image above. The “Rare” zones represent zones of lesser probability of occurrence, as compared to the area enclosed within the Confidence Interval (CI). In other words, events from the “Rare zone” rarely occur when compared to the events within the CI. The significance level defines the threshold, beyond which an event would be considered as a rare occurrence. Therefore, a significance level of 5% would imply that all events which lie outside the 95% Confidence Interval would be considered as rare unlikely chance occurrences. The 5% significance level

(ie:95% confidence interval) is the most widely chosen significance level.

Suppose we have the Sampling Distributions of the means of the weights of a particular species of mice and say that we have the weight of a single mouse of an unknown species. We want to know how probable it is that this mouse belongs to the same species of mice mentioned earlier. We do this by measuring the weight of the rat and comparing it with the Sampling Distribution we have. If the weight of the mouse lies outside the 95% confidence interval, then it becomes quite probable that the mouse is not of the same species.

Hypothesis Testing involves the following steps:

1. We define the Null & Alternative Hypothesis
2. We choose an appropriate Significance Level for rejecting the Null Hypothesis
3. We sample from the Population or computationally simulate the stochastic process of sampling from the population.
4. We test for “Significance” on the samples just obtained, based on which we either reject or confirm the Null hypothesis.

Note that Hypothesis Testing does not prove which hypothesis is correct. It only says that the Alternative Hypothesis becomes "more probable" if the Null Hypothesis can be rejected.

TEST STATISTIC:

A test statistic is the observation based on which the Alternative Hypothesis is formulated. The Alternative Hypothesis is usually created to serve purposes with the following themes :

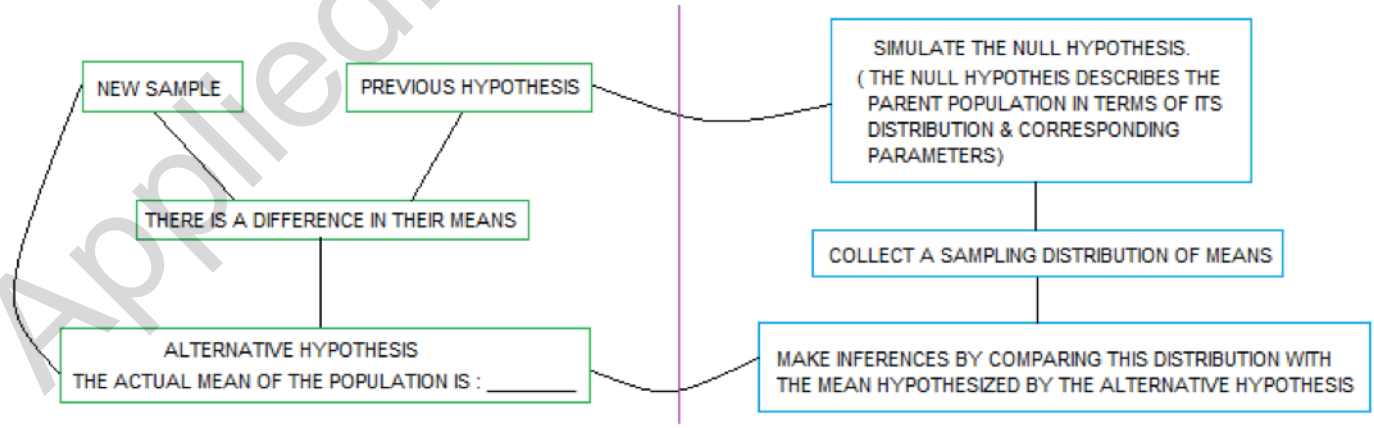
1. Detect Change: The mean of a newly derived sample is found to be different from the previously "hypothesized" mean of the same population and we want to know if this difference is significant or not. Has the mean actually changed?

2. Compare samples : We have 2 samples and we want to know if they belong to the same population or not.

In the first case, we have a single Sample and its mean value differs from the mean value suggested by the previous hypothesis. We want to know if this observed difference is Significant or not. Here the mean of this newly derived single sample is the Test Statistic.

In cases like these, we have access to the "Null Hypothesis version" of the Parent Distribution, because the Null Hypothesis directly or indirectly describes its Parent population in terms of its distribution type & corresponding parameters. This makes it possible for us to computationally simulate the Null Hypothesis and hence the Test Statistic can be compared with the outcomes of this simulation. For example, we know what the distribution of an unbiased 6 faced die is and so it can be simulated and samples can be drawn from it.

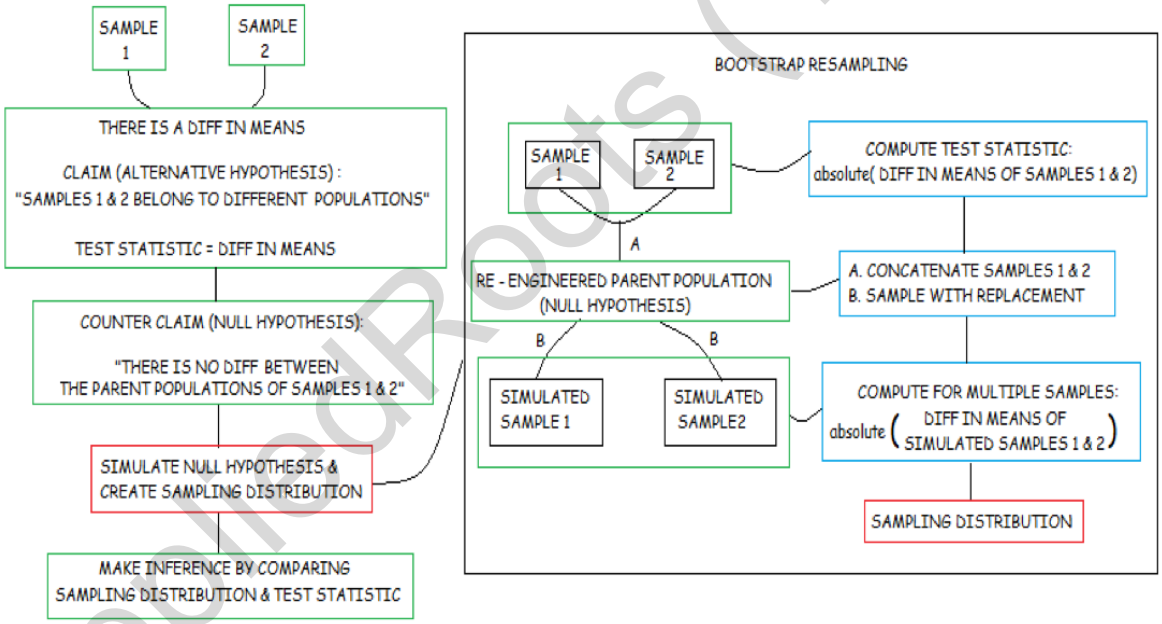
The image shown below illustrates the above explanation:



In the second case we have two comparable samples. For example: Heights of Men and Heights of women or the outcomes of 10000 dice throws from 2 different 6 faced dies and others of the kind. The means of these 2 groups (ie: samples) are observed to be different. We want to infer if this difference is Significant or not. Here the Test Statistic would be the difference in means between the two samples.

In cases like these, we do not have direct access to a Parent Distribution. To overcome this, we reverse engineer what the Parent Distribution, as described by the Null Hypothesis, would be like, by using the 2 samples at hand. In statistical parlance, this reverse engineering technique is called Bootstrap Resampling.

Once the Parent Population of the Null Hypothesis is reverse engineered, we draw multiple pairs of samples from it and compute the absolute difference in means for each pair and plot the resulting Sampling Distribution of these differences in means. The position of the Test Statistic with respect to this Sampling Distribution, will tell us if we should consider the Test Statistic as a "rare" occurrence and hence the difference between the means of the 2 original samples as significant. The image below illustrates the above explanations:



BOOTSTRAP RESAMPLING:

In this technique, we concatenate the 2 samples that we want to test for significance. New samples are then drawn "with replacement" from this resulting concatenation. "With Replacement" means that, every value drawn from this concatenated dataset, is immediately replaced, keeping the dataset unchanged. Thus, this dataset can be indefinitely sampled from, just like sampling from a real Population.

P VALUE:

P value expresses the probability of the Test Statistic being observed, given the Null Hypothesis. In other words, it is an indicator of how certain we can be that the Test Statistic was derived from a sample that belongs to the Null Hypothesis. The lesser this probability, the more "rare" the chances of observing the Test Statistic, while sampling from the Null Hypothesis distribution.

P value = P(test_statistic | Null Hypothesis)

In Hypothesis testing, one of the ways to reject the Null Hypothesis is to check if the P Value of the Test Statistic is lesser than the specified Significance Level. If so, it is rejected.

P Values has been a cause for much confusion among beginners in the field of statistics, due to improper interpretations. It is a much better practice in the beginning, to visualize (ie: plot) the position of the Test Statistic with respect to the Sampling Distribution of the Null Hypothesis to get a clearer picture.

Examples of Hypothesis Testing:

1. Six Faced Die - Theme 1:

Say that after a certain amount of experience of using a particular 6 faced die, we begin to doubt that it is biased. We want to know how probable it is that our doubts could be true. We implement the following Hypothesis Test for this purpose:

1. We collect one sample of 1000 dice throws of the suspected dice and measure its mean. This value becomes the Test Statistic.
2. The Null hypothesis will be that the die is not biased.
3. The Alternative Hypothesis will be that the die is biased.
4. We choose a Significance Level (say 5%) to quantify what we would consider as a "significant difference".
5. We then simulate an unbiased die and collect 1000 samples, each containing 1000 dice throws. We then compute the mean for each of these samples.
6. We then plot the Sampling Distribution of the simulated unbiased die along with the Test Statistic obtained from the "real" die which we are investigating.

The result of the above hypothesis test implemented in python code, is shown below. The code contains the following main functions:

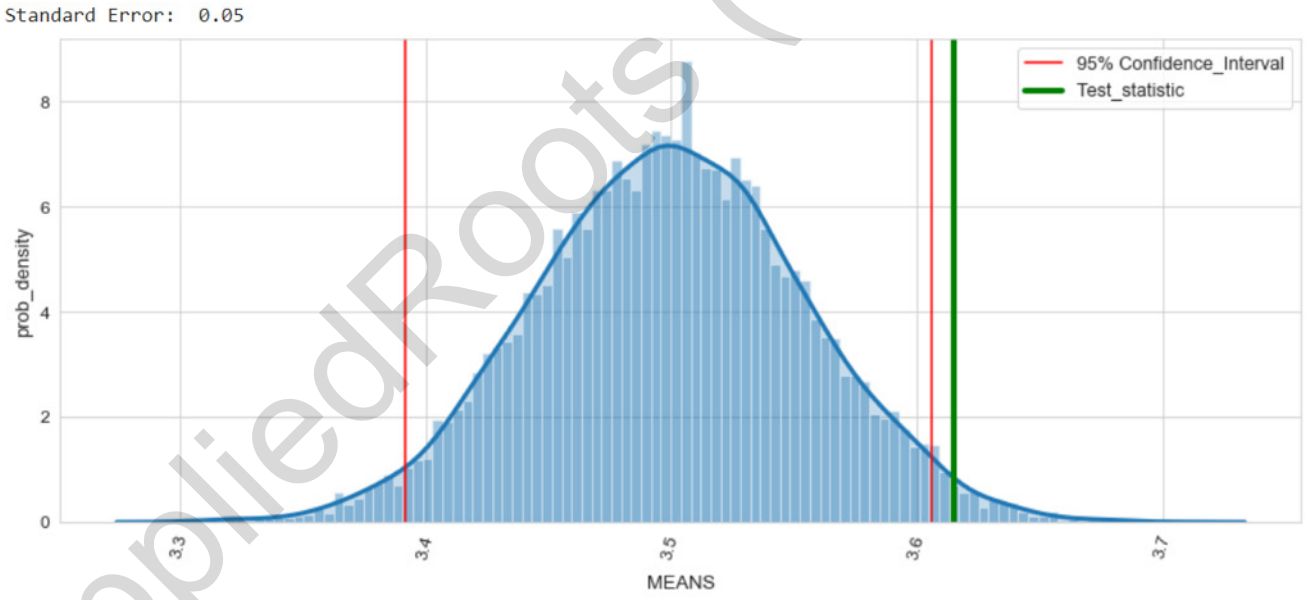
1. fn_test_statistic_biased_die: This implements step number 1 mentioned above.
2. fn_sampling_distribution_unbaised_die: This implements step 5 mentioned above.
3. fn_plot_CI: This implements step 6 mentioned above.

```
bias, baised_face = 0.21, 6 # Die is 21% baised towards face 6
                            # Unbiased die would have a probability of
                            # 1/6 (0.16) for each face

test_statistic = fn_test_statistic_biased_die(bias, baised_face)
test_statistic
```

```
3.615
```

```
n_samples = 10000
sampling_dist = fn_sampling_distribution_unbiased_die(n_samples, sample_size = 1000, n_faces = 6)

xlabel = 'MEANS'
n_tails = 2

fn_plot_CI(sampling_dist, test_statistic, xlabel, n_tails, significance_level = 5)
```

```
Standard Error:  0.05
```



It can be seen from the above plot, that the Test Statistic lies outside the 95% CI of the Null Hypothesis distribution. The Null hypothesis can therefore be rejected, making it quite probable that the Alternative Hypothesis is true.

2. PREGNANCY LENGTH OF FIRST BORN BABIES AND OTHER BABIES - THEME 2:

Imagine that there is a claim made that the pregnancy lengths of first born babies are longer than those for other babies. We want to know if this claim could be true. We do this by examining the records of a Hospital that specializes in childbirth. Below is the dataset collected from the hospital, the pregnancy length is measured in weeks:

```
df = pd.read_csv('df_preg_lgs.csv')
df.sample(10)
```

	preg_lg	wt_baby	type
7192	35	5.0000	others
6950	39	7.3750	others
2778	39	6.8750	firsts
1670	39	9.1250	firsts
7688	38	7.3750	others
2251	37	6.0000	firsts
7302	37	8.8125	others
8935	37	5.0625	others
7557	39	10.0000	others
6652	38	5.2500	others

```
df.describe()
```

	preg_lg	wt_baby
count	9038.000000	9038.000000
mean	38.570591	7.265628
std	2.687198	1.408293
min	0.000000	0.125000
25%	39.000000	6.500000
50%	39.000000	7.375000
75%	39.000000	8.125000
max	50.000000	15.437500

We implement the following Hypothesis Test to test the claim:

1. We define the Null & Alternative Hypothesis. In this case the Alternative Hypothesis would be that there is a significant difference in the pregnancy lengths of first born babies & the other babies a. The Null Hypothesis would be the counterclaim, that there is no difference.

2. We then segregate the two groups and then measure the difference in their means. This becomes our Test Statistic.

3. We then define the Significance Level.

4. We then create a Sampling Distribution of the difference in means, by reengineering the Null Hypothesis distribution, using the Bootstrap Sampling Technique discussed earlier. That is - We concatenate the 2 groups and draw multiple pairs of samples from it, without replacement. The absolute difference in means of these pairs of samples becomes the Sampling Distribution

5. 5. We plot the Sampling Distribution and compare the position of the Test Statistic with respect to it to check for significance.

6. 6. If the Test Statistic does not lie within the Confidence Interval defined by the Significance level chosen, we reject the Null Hypothesis and consider the Alternative Hypothesis to be quite probable.

The result of the above hypothesis test implemented in python code, is shown below. The code contains the following main functions:

1. fn_bootstrap_sampling_distribution: Implements step 4 mentioned above.
2. fn_plot_CI: Implements step 5 mentioned above.
3. fn_p_val: Computes P Value.

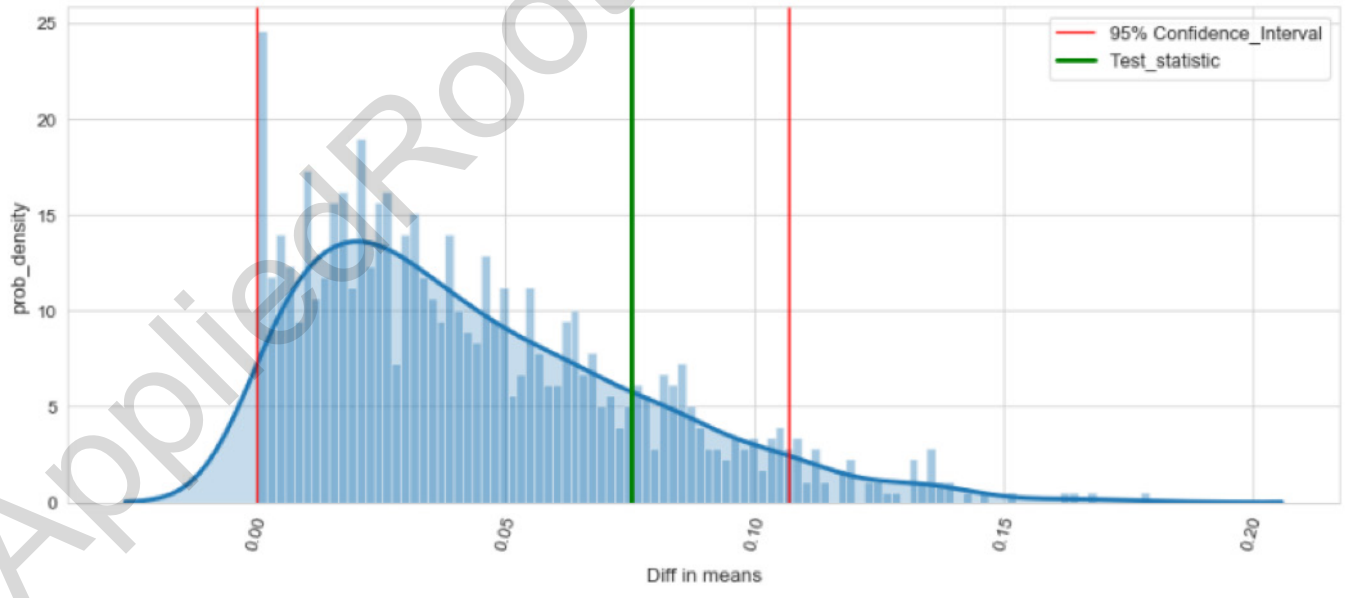
```
group1 = df[df.type == 'firsts'].preg_lg.values
group2 = df[df.type == 'others'].preg_lg.values
```

```
test_statistic = abs(group1.mean() - group2.mean())
test_statistic
```

```
0.07511149297508268
```

```
n_samples = 1000
sampling_dist = fn_bootstrap_sampling_distribution(group1, group2, n_samples, stat_type = 'mean')

xlabel = 'Diff in means'
n_tails = 1

fn_plot_CI(sampling_dist, test_statistic, xlabel, n_tails, significance_level = 5)
```

```
Standard Error:  0.03
```




```
p_val = fn_p_val(sampling_dist, test_statistic)
p_val
```

```
0.191
```

It can be seen that the Test Statistic lies within the defined Confidence Interval and hence the Null Hypothesis cannot be rejected. The P value also reflects this fact, since it is greater than 0.05 (5%).

Note that the Sampling Distributions for themes 1 & 2 are not the same, due to the different nature of the Test Statistic in each case. Theme 1 has a Sampling Distribution that is two tailed (a Normal Distribution), whereas, theme 2 has a one tailed distribution as shown above (There can be no values less than zero because the distribution represents the absolute difference between mean values).

Statistical Power:

The measure of the ability of a Hypothesis Test to detect a difference between the samples, when the difference truly exists(ie: We know that the Alternative Hypothesis is actually True), is called the Statistical Power of the test. It is defined as the probability that the Hypothesis Test will reject the Null Hypothesis given that the Alternative Hypothesis is true:

Statistical Power = P(Rejecting H0 | H1 is true)

Statistical power mainly depends on the following three factors:

1. The Significance Level chosen.
2. The sample size used.
3. The actual magnitude of the difference between the samples.

We have control over the first two factors mentioned above and hence they must be carefully chosen while trying to optimize the Statistical Power of a Hypothesis Test.

7. LINEAR ALGEBRA FOR MACHINE LEARNING

LINEAR ALGEBRA FOR MACHINE LEARNING

Linear Algebra is the mathematics of vectors and vector spaces. It primarily involves concepts such as:

1. Representing interactions between vectors (ie: vector operations).
2. The manipulation of vector spaces (ie: Transformations) so as to make it possible to have representations of the same vectors in alternate vector spaces.

Linear Algebra makes it possible for us to abstract mathematically into hyperdimensional spaces (ie: Spaces that are more than 3 dimensional). It makes it possible for us to perform meaningful computations on high dimensional data by extrapolating our comprehensions about basic geometrical concepts such as space, distance, perpendicularity, collinearity and others, into the hyperdimensional spaces. This ability to generalize basic geometrical concepts over hyperdimensional spaces is used exhaustively in Machine Learning.

SCALARS, VECTORS, FEATURES AND FEATURE SPACES:

Vectors and Vector Spaces (or Feature Spaces) form the most fundamental building blocks of Linear Algebra.

A Vector is basically an ordered sequence of values, that is usually representative of some " object". Each value contained within the vector numerically quantifies a particular feature/attribute of the object.

"Scalar" is the Linear Algebra Terminology for single values, and so a vector could also be defined as an ordered sequence of scalars.

The object that a vector represents, can be thought of as existing at a particular location within an imaginary space called the feature space. The dimensionality(ie: number of axes) of this space is equal to the number of values contained within the vector (ie: size of the vector) and the location of the object in this space is defined by the magnitude of the values the vector contains.

In other words, the imaginary spaces that vectors exist in are called Vector Spaces or Feature Spaces. Each axis of a Feature Space represents a particular feature that is descriptive of the objects it represents. This is demonstrated in the example below.

Feature Spaces can best be understood by visualizing 3 dimensional datasets. Consider the synthetic dataset created as shown below. The dataset consists of 7 "objects", described using 3 features.

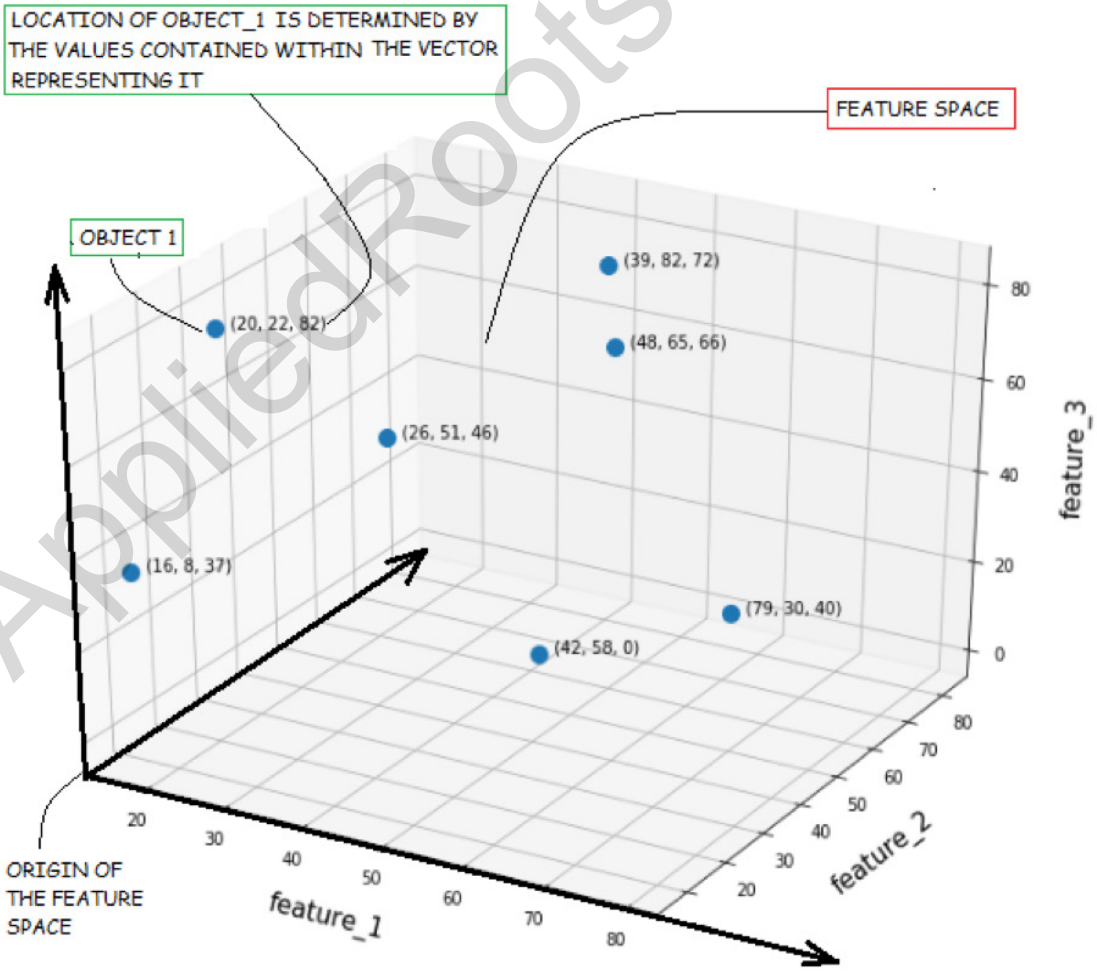
```
n_objects = 7
objects = range(1, n_objects + 1)

cols = dict(objects = objects,
            feature_1 = np.random.randint(100, size = n_objects),
            feature_2 = np.random.randint(100, size = n_objects),
            feature_3 = np.random.randint(100, size = n_objects))

df = pd.DataFrame().assign(**cols)

df.head(5)
```

	objects	feature_1	feature_2	feature_3
0	1	20	22	82
1	2	42	58	0
2	3	48	65	66
3	4	26	51	46
4	5	16	8	37

VECTOR REPRESENTING "OBJECT 1"

CORRESPONDS TO A SPECIFIC LOCATION IN THE FEATURE SPACE

```
feature_1, feature_2, feature_3 = df.feature_1, df.feature_2, df.feature_3

fn_plot_3D(feature_1, feature_2, feature_3)
```



Consider the Heights & weights dataset. The objects contained within this dataset are humans and the features/attributes used to describe these objects are “height” & “weight”. Here, each human can be represented by a vector containing 2 values in a fixed sequence/order. If we choose the order to be height first and weight second, then all humans contained in this dataset can be described by vectors with this fixed sequence.

Each of these objects (ie: humans) in the dataset could now be imagined to exist at a particular location, in a 2 dimensional feature space, whose axes represent height & weight. This same logic is extended to datasets containing objects that have more than 3 features.

Sets

A set is basically a collection of unique values(ie: No value is repeated within the collection). For example, the numbers 1, 2, 3, 4 & 5 when considered collectively, form a single set of size 5, written as:

S = {1, 2, 3, 4, 5}

Note that sets are specifically represented by “curly brackets” to distinguish them from normal lists and the alphabet representing the set is in bold capitals.

One of the most unique properties of sets is the fact that they can represent unbounded (infinite at one or both ends) mathematical archetypes. For example, we could use a set to represent all real numbers that exists, another set to represent all prime numbers and yet another set to represent numbers common to both these sets - All three have no limits bounding them.

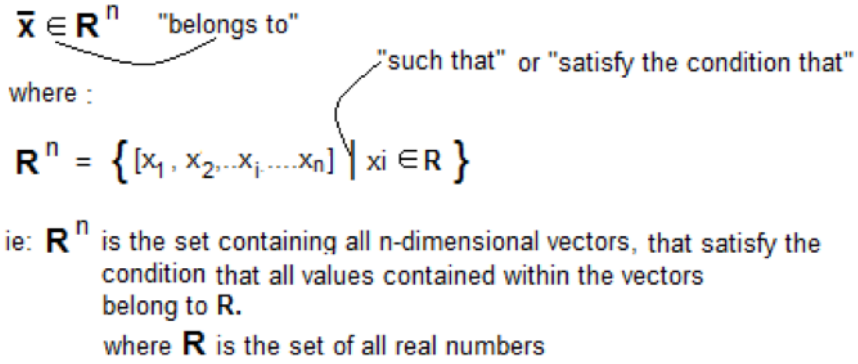
MATHEMATICAL REPRESENTATION & DEFINITION OF VECTORS:

Vectors are mathematically represented as shown below. The alphabet representing the vector is in bold small letters and it has a “dash” crowning it:

$$\bar{\boldsymbol{x}} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

Elements of the vector are represented as a "column" within a square bracket.

Vectors are mathematically defined by expressing them in terms of the group or set they belong to, for example:



VECTOR OPERATIONS:

Amongst the set of all vector operations that exist, the ones that are most relevant for the purposes of Machine Learning are:

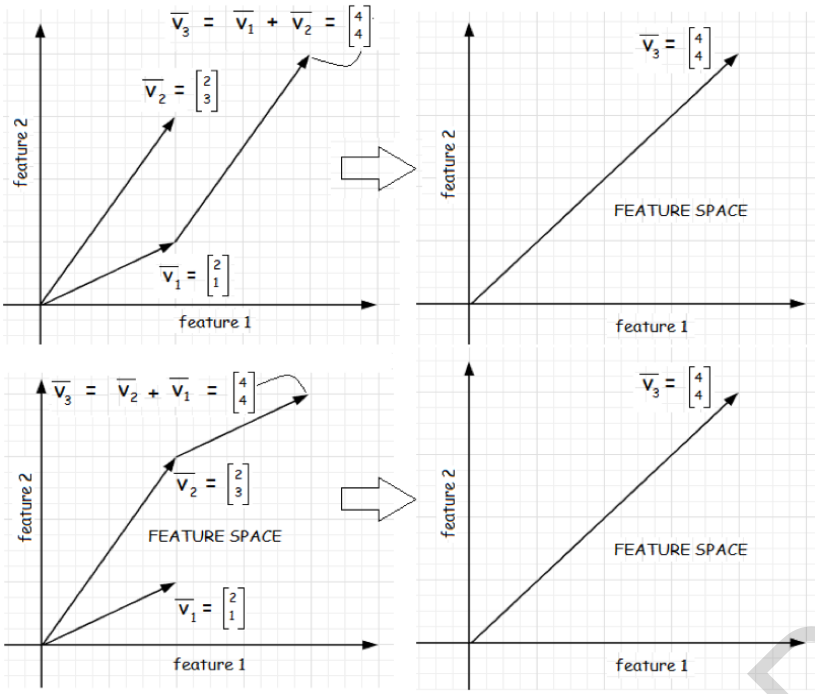
1. Vector addition or combination.
2. Vector scaling
3. Vector dot product

Vectors are graphically represented as arrows such that, the “arrow head” represents location in the feature space with respect to the start point of arrow. This helps us visualize the vector operations mentioned above. The explanations & images that follow will illustrate this further.

VECTOR COMBINATION OR ADDITION:

Vector addition is the operation that gives us the resultant location in the feature space, if we take sequential steps based on the positional information provided by a pair of vectors.

Consider the image shown below, the location represented by vector v3 is the resultant location we would reach if we first moved to the location represented by vector v1 and then relative to v1 we move to the location represented by vector v2. In other words, vector 3 represents the combination/addition of vectors 1 & 2. Note that the sequence of steps does not matter.



The above mentioned Vector addition is mathematically expressed as:

$$\overline{v_3} = \overline{v_1} + \overline{v_2} = \begin{bmatrix}2\\1\end{bmatrix} + \begin{bmatrix}2\\3\end{bmatrix} = \begin{bmatrix}4\\4\end{bmatrix}$$

In general:

$$\begin{bmatrix}f_{1,1}\\f_{2,1}\end{bmatrix} + \begin{bmatrix}f_{1,2}\\f_{2,2}\end{bmatrix} = \begin{bmatrix}f_{1,1} + f_{1,2}\\f_{2,1} + f_{2,2}\end{bmatrix}$$

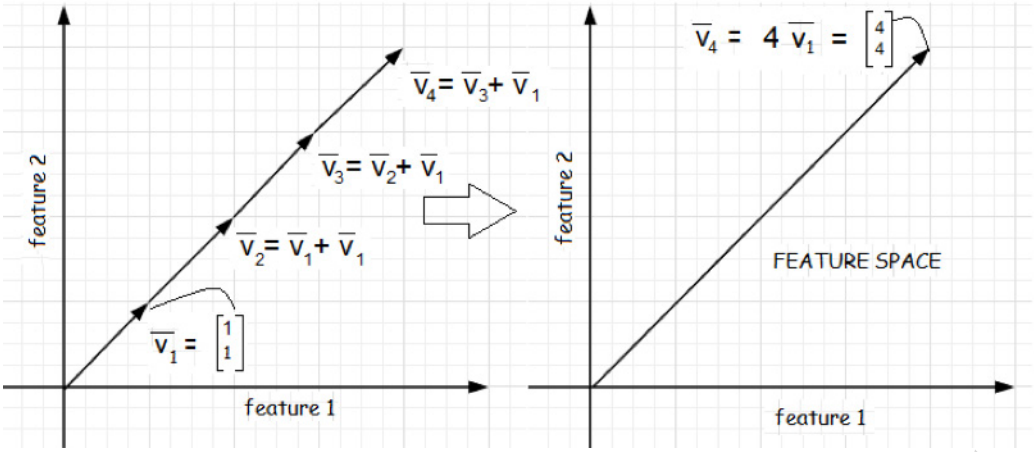
Vector Scaling Or Multiplication:

Vector scaling is the operation that gives us the resultant location in the feature space, if we take repeated sequential steps based on the positional information provided by a single vector.

Consider the image shown below, the location represented by vector v4 is the resultant location we would reach if we sequentially moved 4 times to the "relative location" represented by vector v1. In other words, vector 4 represents the vector addition of vector 1 with itself performed in the manner shown below:

$$\overline{v_4} = 4\overline{v_1} = (((\overline{v_1} + \overline{v_1}) + \overline{v_1}) + \overline{v_1})$$

This operation has the effect of "scaling" the original vector by the scalar value it is multiplied with.



The above mentioned Vector scaling is mathematically expressed as:

$$\overline{V}_4 = 4\overline{V}_1 = 4\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 4 \\ 4 \end{bmatrix}$$

In General:

$$a\begin{bmatrix} f_1 \\ f_2 \end{bmatrix} = \begin{bmatrix} a\,f_1 \\ a\,f_2 \end{bmatrix}$$

COLINEARITY AND LINEAR INDEPENDENCE:

Two vectors v1 & v2 are said to be collinear if one of them can be expressed as a "scaled" version of the other. Conversely, if vectors v1 & v2 are non collinear, they are said to be Linearly Independent of each other.

Vectors $\overline{V}_1$ & $\overline{V}_2$ are said to be collinear if:

$$\overline{V}_1 = a\,\overline{V}_2$$

In the image above, that explains the vector scaling operation, vectors v1, v2, v3 & v4 are considered to be collinear to each other or in other words, they are not linearly independent.

DISTANCE BETWEEN TWO VECTORS:

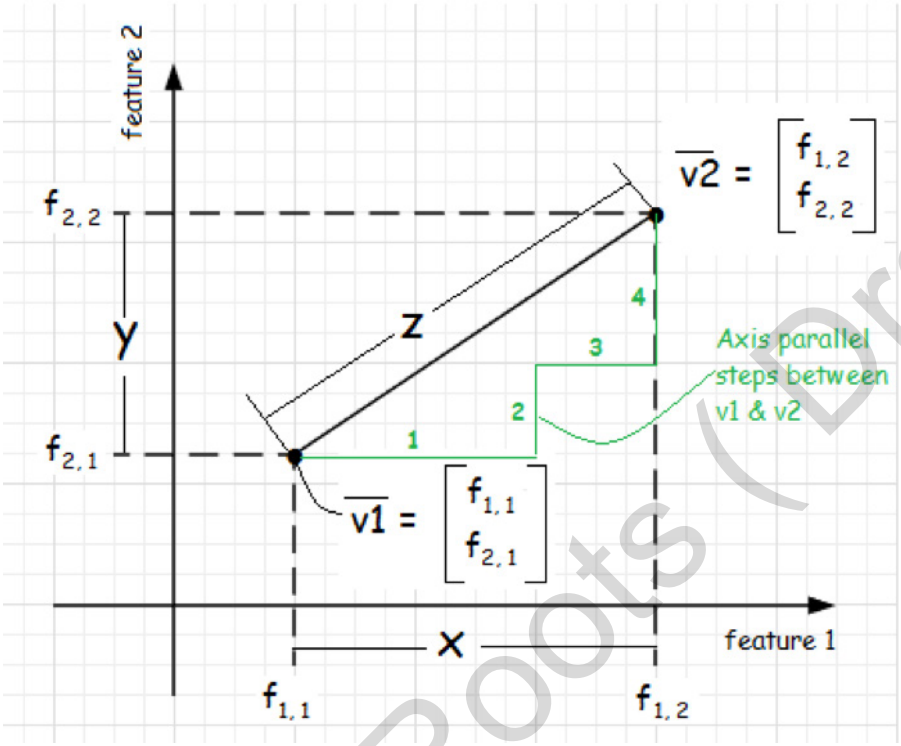
A vector and the location it represents are generally interpreted as one and the same for the sake of convenience. So we can say "distance between two vectors" and this would mean the distance between the locations represented by the two vectors.

Since the mathematics of Linear Algebra generalizes over any feature space, irrespective of its dimensionality, even simple intuitive concepts like "distance" can sometimes seem too abstract to practically conceptualize. Imagine what distance would mean in a 5 dimensional

dimensional (feature) space. Though this is not possible to imagine, we can still mathematically extrapolate our 3 dimensional definitions of "distance" into higher dimensional feature spaces using the power of Linear Algebra.

TYPES OF DISTANCES:

The concept of distance is best understood with reference to 2 dimensional feature spaces:



EUCLIDEAN DISTANCE:

This is a measure of distance based on the Pythagorean theorem. It is what we conventionally or intuitively consider "distance" to be. Its definition is based on the question: "What is the shortest distance between two locations". With reference to the image above, the Euclidean distance is given by the value of "z", which is equal to the square root of the sum of the squares of "x" & "y". Euclidean distance is expressed mathematically as shown below:

$$\|\overline{v1} - \overline{v2}\|_2 = \left(|f_{1,2} - f_{1,1}|^2 + |f_{2,2} - f_{2,1}|^2 \right)^{1/2}$$ — For 2 dimensional feature spaces

$$\|\overline{v1} - \overline{v2}\|_2 = \left(\sum_{i=1}^{n} |f_{i,2} - f_{i,1}|^2 \right)^{1/2}$$ — Generalized for "n" dimensional feature space

*Norm 2

Special brackets denote absolute value

MINKOWSKI'S DISTANCE AND NORMS:

The Minkowski's distance is a generalization of the Pythagorean theorem with respect to the exponentiation used within its formula:

$$z^2 = x^2 + y^2$$

The Minkowsky's distance generalizes the above formula as:

$$z^P = x^P + y^P$$

In the equation above, the value of z (ie: the pth root of the right hand side of the equation) for any specific value of p would be considered as a particular interpretation of distance. For example, if p were equal to 5, the distance (z) would be called a "Norm 5" distance. In the same vein, the Euclidean distance can also be called the "Norm 2" distance. Fractional Norms and those other than Norms 1 & 2, are abstract measures of distance that have no geometric interpretations.

The Minkowski's distance is expressed mathematically as shown below:

$$\|\overline{v2} - \overline{v1}\|_p = \left(|f_{1,2} - f_{1,1}|^p + |f_{2,2} - f_{2,1}|^p \right)^{1/p}$$ — For 2 dimensional feature spaces

$$\|\overline{v2} - \overline{v1}\|_p = \left(\sum_{i=1}^{n} |f_{i,2} - f_{i,1}|^p \right)^{1/p}$$ — Generalized for "n" dimensional feature space

MAGNITUDE OF A VECTOR:

The Magnitude of a vector is the general terminology used to describe the "Norm 2" (or Euclidean) distance of that vector, from the origin of its feature space.

Consider the mathematical representation of the Euclidean Distance just shown previously. The magnitude of vector v2 can be obtained by substituting a zero vector in place of vector v1. A zero vector is a vector that contains only zero values and it is representative of the origin of the feature space.

Thus the magnitude of a vector is mathematically expressed as:

$$\|\overline{v}\|_2 = \left(|f_1|^2 + |f_2|^2 \right)^{1/2}$$ — For 2 dimensional feature spaces

$$\|\overline{v}\|_2 = \left(\sum_{i=1}^{n} |f_i|^2 \right)^{1/2}$$ — Generalized for "n" dimensional feature space

MANHATTAN DISTANCE:

This is a grid based distance measure. Manhattan distance is the distance between vectors v1 & v2 if we only take "axis-parallel" steps (ie: Any single step has to be parallel to at least one axis of the feature space). Therefore there are many combinations of steps we can take to move from v1 to v2, the simplest being: Taking a step "x" and then taking a step "y" (or vice-versa).

In other words, the Manhattan distance is the "Norm 1" interpretation of distance and it is mathematically expressed as shown below:

$$\|\overline{v2} - \overline{v1}\|_1 = |f_{1,2} - f_{1,1}| + |f_{2,2} - f_{2,1}| \quad \text{For 2 dimensional feature spaces}$$

$$\|\overline{v2} - \overline{v1}\|_1 = \sum_{i=1}^{n} |f_{i,2} - f_{i,1}| \quad \text{Generalized for "n" dimensional feature space}$$

Norm 1

THE DOT PRODUCT & COSINE DISTANCE:

The dot product of 2 vectors v1 & v2, belonging to the same feature space is mathematically defined as:

$$\overline{v}_1 \cdot \overline{v}_2 = \begin{bmatrix} f_{1,1} \\ f_{2,1} \end{bmatrix} \cdot \begin{bmatrix} f_{1,2} \\ f_{2,2} \end{bmatrix} = f_{1,1}\, f_{1,2} + f_{2,1}\, f_{2,2} \quad \text{For 2 dimensional feature spaces}$$

$$\overline{v}_1 \cdot \overline{v}_2 = \sum_{i=1}^{n} f_{i,1}\, f_{i,2} \quad \text{Generalized for "n" dimensional feature space}$$

Represents Dot Product

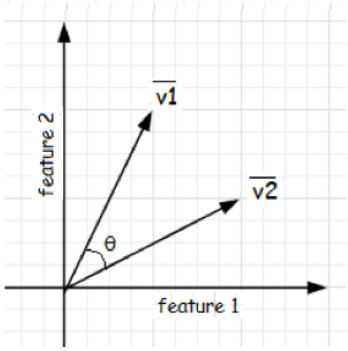
Note that the result of the dot product is always a scalar quantity irrespective of the dimensionality of the vectors being considered. This resulting scalar value has tremendous utility, because some very basic Linear Algebraic concepts can be expressed in terms of the dot product.

Consider the following:

$$\|\overline{v}\| = (\overline{v} \cdot \overline{v})^{1/2}$$

$$\text{Cos } \theta = \frac{\overline{v}_1 \cdot \overline{v}_2}{\|\overline{v}_1\| \, \|\overline{v}_2\|} = \frac{\overline{v}_1 \cdot \overline{v}_2}{(\overline{v}_1 \cdot \overline{v}_1)^{1/2} (\overline{v}_2 \cdot \overline{v}_2)^{1/2}}$$

$$\text{Cosine Distance} = 1 - \text{Cos } \theta$$



The magnitude of a vector and the angle between two vectors (ie: The angle between the lines joining the vectors to the origin) can easily be expressed in the form of dot products. Many important concepts in Linear Algebra, like feature space transformations and the ability to perform matrix factorization operations (both of which we will be exploring further on), can be conveniently expressed and computed using the dot product.

The cosine of the angle between a pair of vectors is used as a measure of "similarity" between the 2 vectors. The lesser this angle the more the value tends towards 1 and at zero angle, the value becomes equal to one, indicating maximum similarity. This similarity measure between a pair of vectors is called the "Cosine Similarity".

Since the values of Cosine similarity lie within the range of zero & one, [1 - cosine similarity] can be used as a measure of "dissimilarity" (or distance between a pair of vectors). This distance measure is called the "Cosine Distance".

Cosine Distance is one of the most extensively used distance measures in Machine Learning and it has proven to have very good utility while working with high dimensional data.

MATHEMATICAL PROPERTIES OF DOT PRODUCTS:

The dot product has the following mathematical properties:

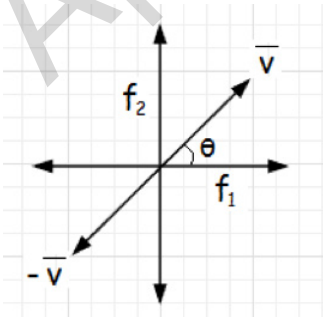
$$\bar{v}_1 \cdot \bar{v}_2 = \bar{v}_2 \cdot \bar{v}_1 \quad \text{COMMUTATIVITY}$$

$$(\bar{v}_1 + \bar{v}_2) \cdot \bar{v}_3 = \bar{v}_1 \cdot \bar{v}_3 + \bar{v}_2 \cdot \bar{v}_3 \quad \text{DISTRIBUTIVITY}$$

$$a\,\bar{v}_1 \cdot \bar{v}_2 = a\,(\bar{v}_1 \cdot \bar{v}_2) \quad \text{ASSOCIATIVITY}$$

THE "DIRECTION" OF A VECTOR:

The direction of a vector is defined as the angle that the vector makes with any one of the axes of the feature space (Generally this axis represents "feature 1"). Vectors having the same or opposite directions are collinear to each other, since each is a scaled version of the other. Any vector scaled by a value of negative one, produces a vector of the same magnitude in the opposite direction.



UNIT VECTORS:

Vectors having unit magnitude are considered as unit vectors. These vectors could be perceived as the original "unscaled" vectors, from which all other vectors are scaled from. Every direction in a feature space has its own corresponding unit vector.

Given a particular vector, the unit vector corresponding to its direction is got by scaling that vector by the reciprocal of its magnitude.

$$\text{Unit vector along vector } \bar{v} = \frac{\bar{v}}{||\bar{v}||_2}$$

LINEAR COMBINATION OF VECTORS:

Given two vectors v1 and v2 belonging to the same feature space, a linear combination of v1 and v2 is any vector of the form:

$$a_1\bar{v}_1 + a_2\bar{v}_2 \quad \text{where: } a_1, a_2 \in R$$

The Linear Combination of a set of vectors belonging to the same feature space is mathematically expressed as:

The Linear Combination of a group of vectors:

$\bar{v}_1, \bar{v}_2, \bar{v}_3, \bar{v}_4 \ldots\ldots \bar{v}_n \in R^d$ is defined as:

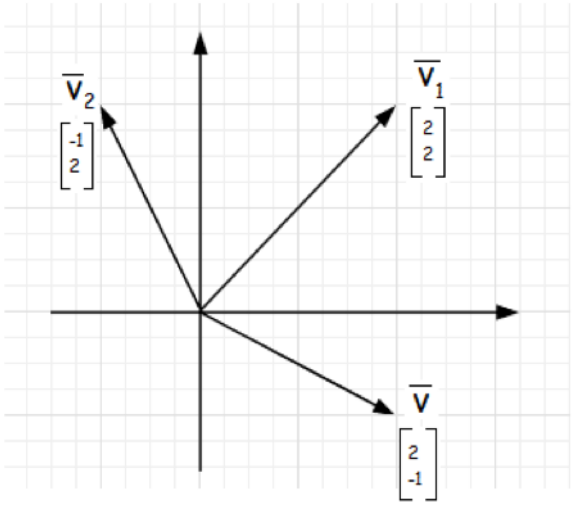
Any vector of the form:

$$a_1\bar{v}_1 + a_2\bar{v}_2 + a_3\bar{v}_3 + a_4\bar{v}_4 \ldots\ldots + a_n\bar{v}_n$$

where:

$$a_1, a_2, a_3, a_4 \ldots a_n \in R$$

The utility of linear combinations of vectors becomes evident when the vectors under consideration are linearly independent. Consider two 2 dimensional (2D) vectors v1 & v2 that are linearly independent of each other. Now any other vector in 2D space can be represented as a linear combination of v1 & v2, as shown below:



Vectors v1 & v2 are Linearly independent.
Hence any vector v in same feature space can represented as a linear combination of v1 & v2.

ie: $a\bar{v}_1 + b\bar{v}_2 = \bar{v}$

ie: $a\begin{bmatrix}2\\2\end{bmatrix} + b\begin{bmatrix}-1\\2\end{bmatrix} = \begin{bmatrix}2\\-1\end{bmatrix}$

Using what we know about vector scaling we have:

$$\begin{bmatrix}2a\\2a\end{bmatrix} + \begin{bmatrix}-1b\\2b\end{bmatrix} = \begin{bmatrix}2\\-1\end{bmatrix}$$

Using what we know about vector addition we have:

$$\begin{bmatrix}2a - 1b\\2a + 2b\end{bmatrix} = \begin{bmatrix}2\\-1\end{bmatrix} \quad \text{ie:} \quad \begin{matrix}2a - 1b = 2\\2a + 2b = -1\end{matrix} \Rightarrow \begin{matrix}a = 5/6\\b = -1/3\end{matrix}$$

Similarly, all vectors in a 3D space can be represented as a linear combination of any 3 linearly independent vectors of the same feature space. Generalizing this concept, we can say that: All vectors in an n dimensional space can be represented as a linear combination of any n linearly independent vectors of the same feature space.

SPAN OF A SET OF VECTORS:

The span of a set of vectors is defined as the set of all linear combinations possible between those set of vectors. It could be perceived as the "space" that can be covered by a set of vectors. A set of n, linearly independent n-Dimensional vectors has the capacity to represent the entirety of n dimensional space and hence the span of these vectors is defined as such:

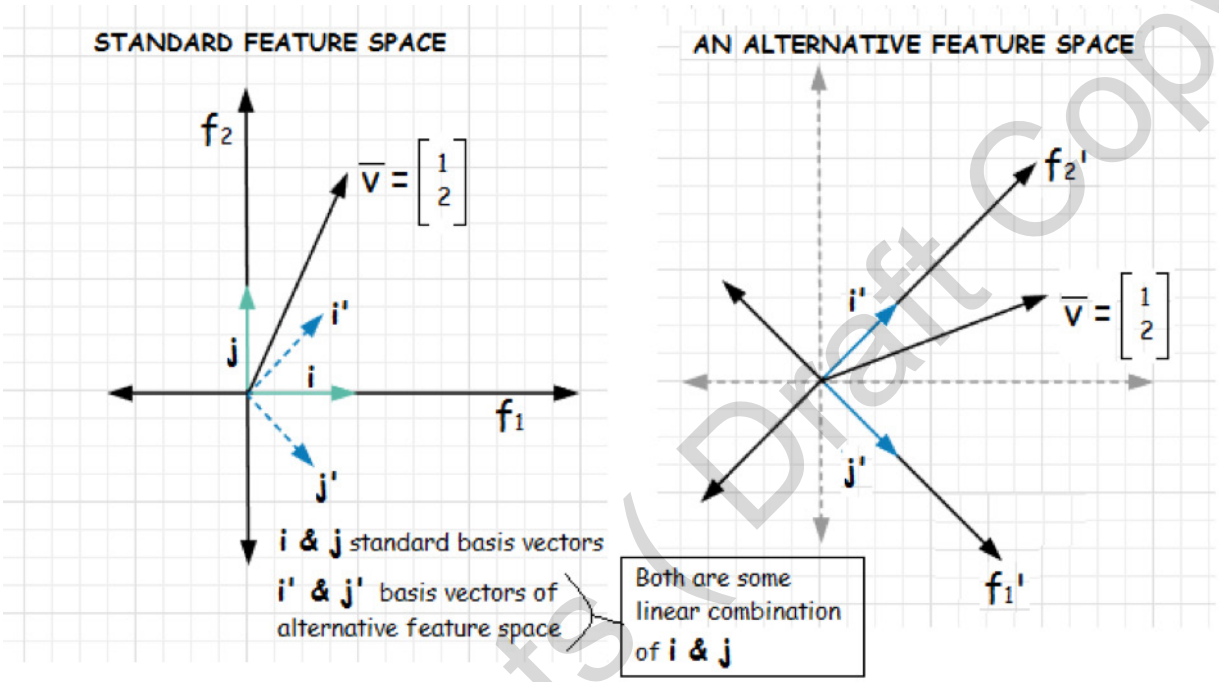
$$\text{span}(\, v_1, v_2, v_3, \cdots v_n \,) = \mathbb{R}^n \quad \text{if all vectors} \in \mathbb{R}^n \ \& \ \text{all of them are linealy independent}$$

BASIS OF A FEATURE SPACE, STANDARD BASIS & ALTERNATIVE FEATURE SPACES:

A "basis" for a feature (or vector) space is a set of linearly independent vectors that span that space.

In other words, any set of linearly independent vectors that can represent all other vectors of the feature space they belong to, as a linear combination of themselves, can be considered as the basis vectors of that feature space. A feature space can have infinite number of basis vectors since all that is required is that the number of linearly independent vectors should be equal to the number of axes that the feature space has (ie: Number of linearly independent vectors should be equal to the dimensionality of the feature space).

The unit vectors along the axes of a "standard" or generic feature space are called the standard basis vectors of the feature space. Standard basis vectors play an important role as the "basic" vectors, with respect to which alternate features spaces can be defined. Consider the image shown below:



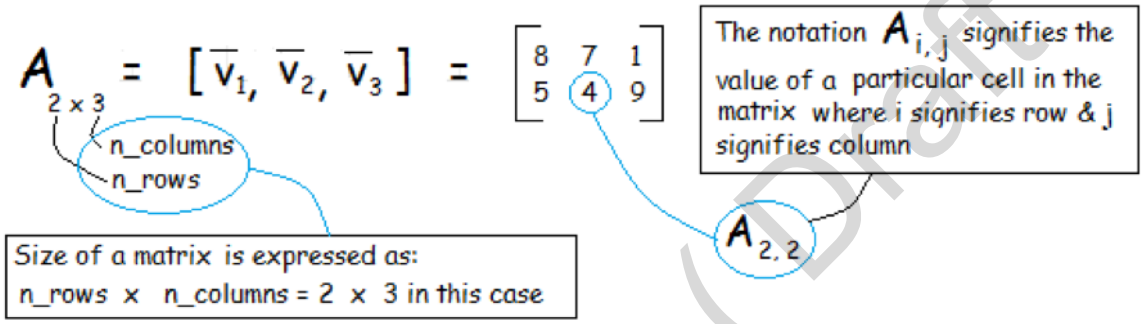
The alternative feature space shown above is defined with respect to the unit basis vectors i' & j' which are linear combinations of the standard basis vectors i & j. The same vector v will have a different representation in the alternative feature space, as shown above. The ability to define alternative feature spaces and represent the data in these alternative feature spaces is key to understanding Deep Learning which is one of the most important machine learning methods we will come across.

8. MATRICES AND TRANSFORMATIONS

Matrices And Transformations

MATRIX REPRESENTATION OF DATA:

A matrix is a collection of vectors belonging to the same feature space. The vectors are enclosed within a square bracket such that they form the columns of the matrix. Hence each column within the matrix represents a data-object (ie: vector) and each row represents the individual features of the data-objects. Shown below is the mathematical representation of a matrix containing three 2D vectors:



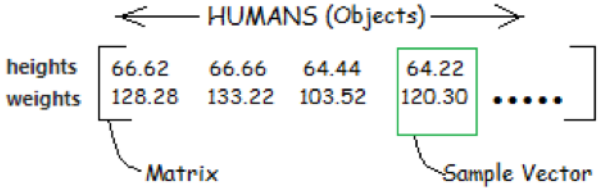
In the above example, the order or sequence of how the vectors are enclosed within the matrix is not important, since it only represents a collection of objects (ie: The order of the vectors do not have any meaning). Note that the alphabet representing the matrix is in bold capitals and has a subscript expressing the matrix dimensions (ie:size).

Consider the Heights & Weights dataset we explored in the previous chapter:

```
1  df_data = pd.read_csv('heights_weights.csv', index_col=0)
2
3  df_data.sample(5)
```

Index	heights	weights
2923	66.62434	128.3887
13887	66.66077	133.2206
21104	64.44608	103.5210
10228	64.22788	120.3019
13150	66.66816	133.0192

TABULAR REPRESENTATION
(Table of observations)

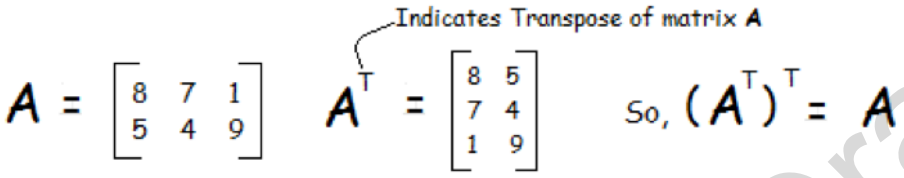


MATRIX REPRESENTATION
(Collection of vectors belonging to the same feature space)

The order of the vectors within matrix does not matter since they are just representative of locations within a feature space. Notice the difference in how data is represented in the " generic" Tabular format and the "formal" matrix format.The tabular format is more readable, but the matrix representation is more suited for understanding Linear Algebraic concepts like matrix transformations.

THE MATRIX TRANSPOSE:

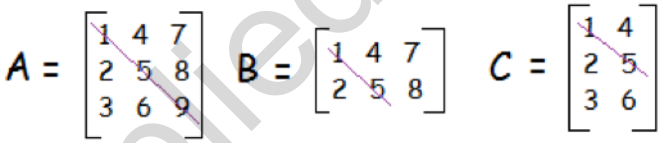
The transpose of a matrix is a new matrix whose rows are the columns of the original, with the same sequence maintained. The transpose of a matrix is expressed as shown below:



Looking at the Tabular representation of a dataset (vectors stacked as rows) from this perspective, we can see that it is the transpose of the Linear algebraic matrix representation of the same dataset. In other words, to obtain the Linear algebraic data matrix, we need to transpose the matrix containing the tabular data. All Linear Algebraic operations are performed only after the tabular data is transposed to obtain the actual matrix representation of the data.

DIAGONAL OF A MATRIX:

The diagonal of a matrix consists of those elements, the cells of which have the same row number & column number. Shown below are 3 example matrices with their diagonals marked.



For all $A_{i,j}$ $B_{i,j}$ & $C_{i,j}$ in above matrices, the elements with i = j form the diagonal

SQUARE & SYMMETRIC MATRICES:

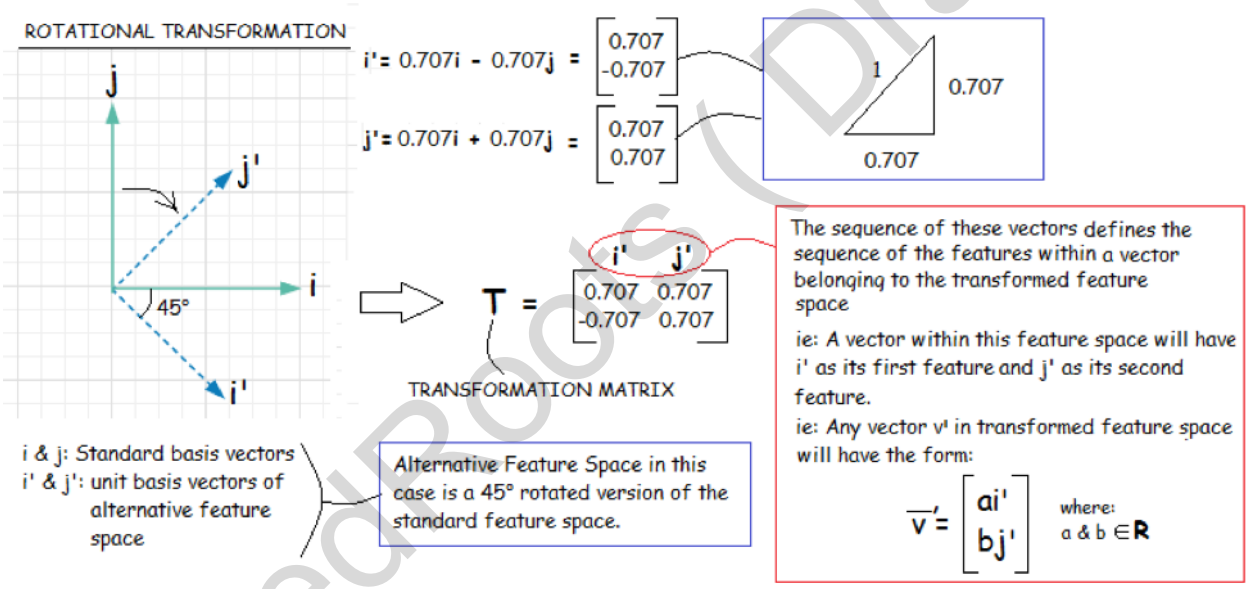
Square matrices are those matrices which have the same number of rows and columns. Symmetric matrices are square matrices which have the same elements, in the same (reflective) order, on either sides of their diagonal.

In other words, if the transpose of a matrix results in the same matrix, then that matrix is a symmetric matrix. The example below demonstrates this:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 5 & 6 \\ 3 & 6 & 9 \end{bmatrix} \quad A^T = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 5 & 6 \\ 3 & 6 & 9 \end{bmatrix}$$

TRANSFORMATION MATRICES:

Apart from the matrix type just discussed, there is another kind of matrix where the sequence or order in which the individual vectors are enclosed within the matrix, has meaning. These kinds of matrices are called Transformation matrices. A transformation matrix basically contains the unit basis vectors of an alternative feature space in a very specific sequence.

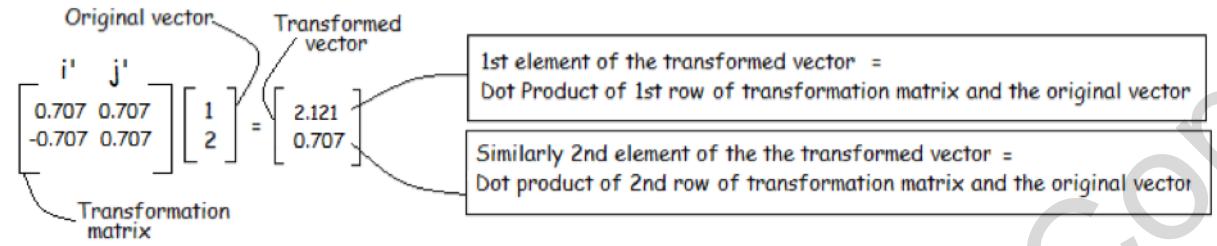


Consider the Transformation Matrix described in the image shown above, it defines an alternate feature space (which is a 45 degree rotated version of the standard feature space), by describing the unit vectors that lie along the axes of the alternate feature space. The new axes now represent "transformed" features. Any vector v in the standard feature space will now have a corresponding representation v' in the alternate feature space.

MATRIX TRANSFORMATION OF A VECTOR:

Given any vector v belonging to the standard feature space, we can compute its corresponding representation in an alternative feature space by "transforming" that vector by the transformation matrix representing the alternative feature space.

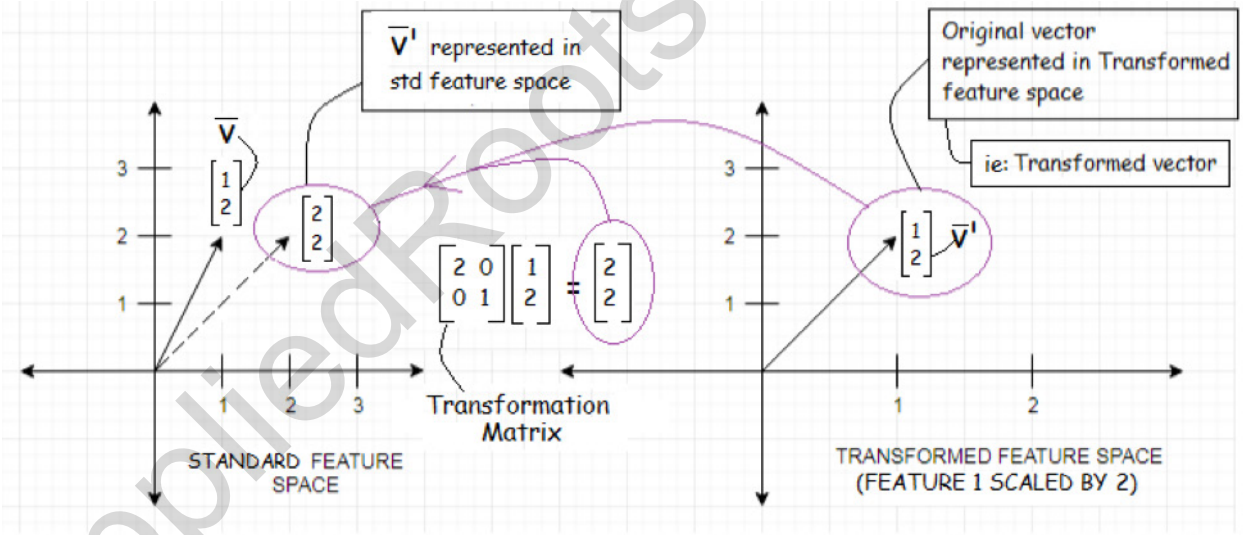
Consider the following example, which is with respect to the transformation matrix mentioned previously:



NOTE :

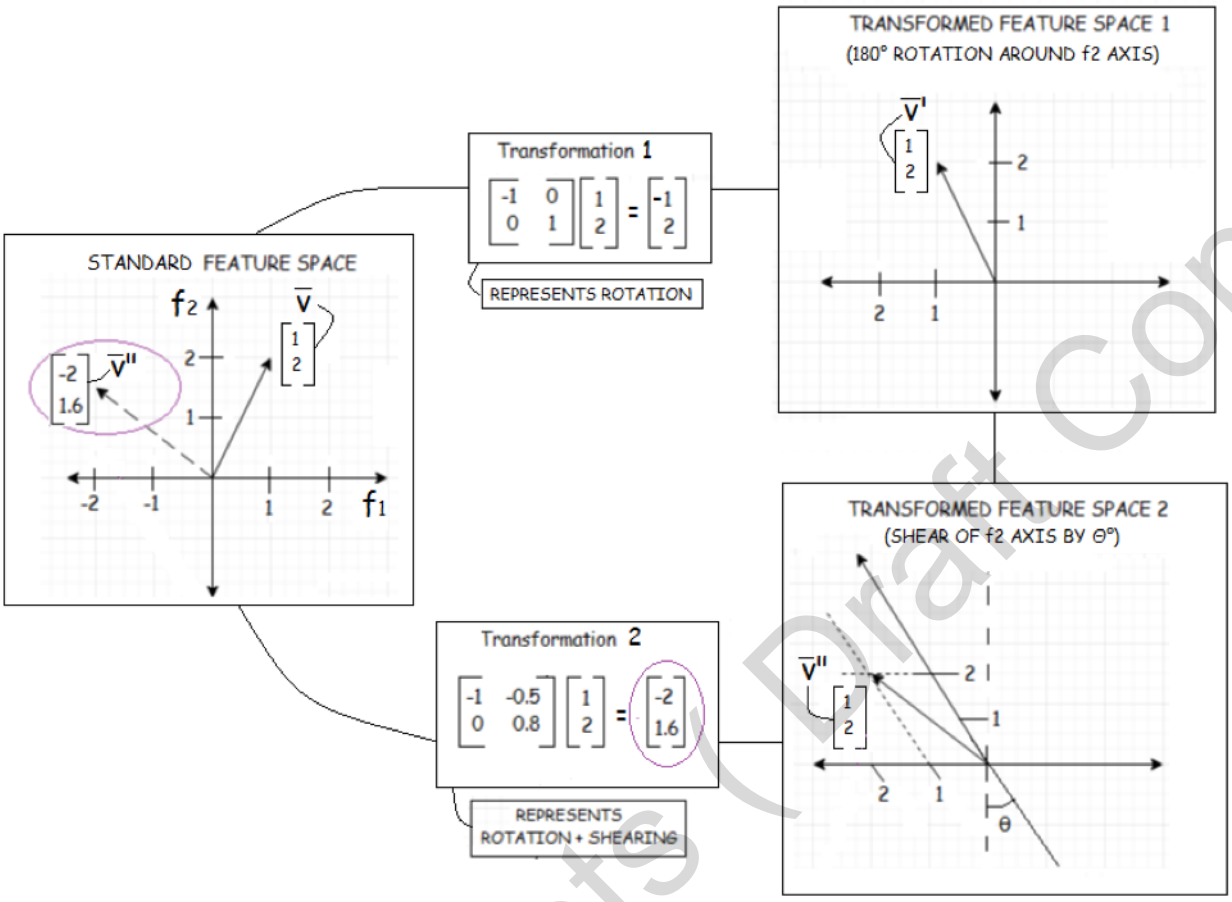
1. The general convention for matrix transformation operation, is to place the transformation matrix in front of the vector that needs to be transformed.
2. For transformation to be possible, the number of elements of the vector that needs to be transformed should be = the number of columns in transformation matrix. (ie: The vector being transformed should belong to the span of i' & j')
3. The rows of a transformation matrix are also vectors. (Since sequence i' - j' is fixed)
4. These same set of rules are applicable to all matrix transformations irrespective of the dimensionalities involved.

Note that the transformed vector derived after a transformation, represents the location that the original vector would have in the transforment feature space. The image below describes this:



The transformation operations allows us to understand the location that any vector would have in an alternative feature space, by representing its transformed counterpart in the standard feature space.

Consider a more complex transformation as shown in the image below:



The image above shows:

1. The "rotational" transformation of vector v to vector v' by transformation matrix 1.
2. The "rotational + shear" transformation of vector v to vector v" by transformation matrix

Notice that transformation matrix 2 represents both "rotation" and "shear" within itself as a single compound transformation.

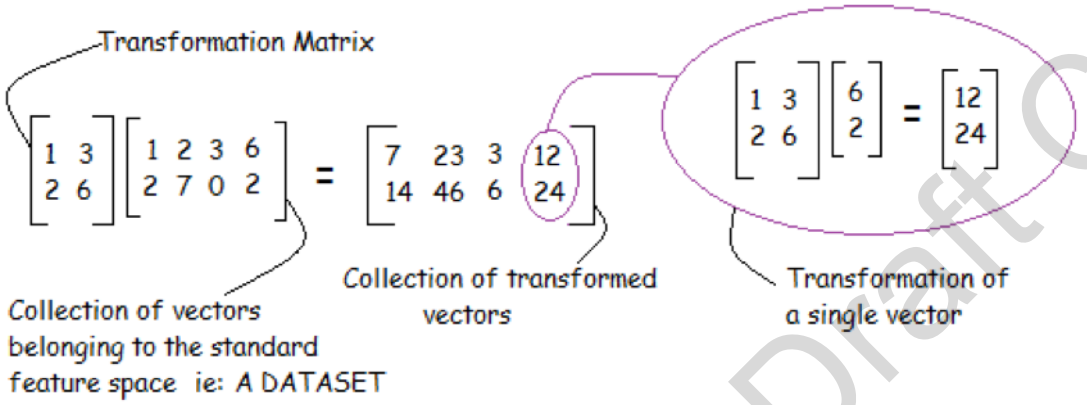
DIAGONAL & IDENTITY MATRICES:

Diagonal Matrices are matrices in which all elements except for the diagonal have zero values. An identity matrix is a diagonal matrix with only unit values. Any vector transformed by an identity matrix results in itself (ie: An identity matrix does not produce any "transformation"), as shown below:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

MATRIX TRANSFORMATION OF A GROUP OF VECTORS (MATRIX MULTIPLICATION):

The image below shows the transformation of an entire dataset to an alternative feature space. In other words, multiple vectors(represented as a matrix) are simultaneously transformed by a single transformation matrix. The Transformation Matrix could thus be viewed as mapping function between two different feature spaces.



THE MATRIX INVERSE OR THE INVERSE TRANSFORMATION MATRIX:

The matrix inverse is a matrix that corresponds to the inverse mapping of a given Transformation Matrix. In other words, given a transformation matrix that transforms vectors from standard feature space to an alternative feature space, the matrix inverse of the transformation matrix is a transformation matrix that transforms vectors from the alternative feature space to the standard feature space.

The inverse of a matrix A is represented as:

Inverse of matrix $A = A^{-1}$

If $AV = V'$, then $A^{-1}V' = V$

The code below shows the dot product, matrix transformation & matrix inverse computed in python:

```
1  import numpy as np
```

```
1  v1 = np.array([-1, 0])
2  v2 = np.array([-0.5, 0.8])
3
4  v1
```

Numpy representation of vectors

```
array([-1,  0])
```

```
1  v1 @ v2
```

Numpy dot product operator

```
0.5
```

```
1  A = np.array([v1, v2])
2  A
```

Numpy array of a list of vectors

```
array([[-1. ,  0. ],
       [-0.5,  0.8]])
```

Note that Numpy stacks vectors as rows instead of as columns

```
1  Transf_matrix = A.T
2  Transf_matrix
```

". T" after after any matrix produces its transpose

```
array([[-1. , -0.5],
       [ 0. ,  0.8]])
```

After transpose, every column represents a vector. This matrix represents the same matrix shown in the previous "rotation+shear" Transformation example

```
1  v = np.array([1, 2])
2
3  v_transformed = Transf_matrix @ v
4  v_transformed
```

Transformation Operator

Rotation + Shear Transformation of vector v

```
array([-2. ,  1.6])
```

Vector v after being Transformed

Note that the same operator is used for matrix transformation and for vector dot product

```
1  Transf_matrix_inverse = np.linalg.inv(Transf_matrix)
2  Transf_matrix_inverse
```

Numpy function for getting inverse of a matrix

```
array([[-1.   , -0.625],
       [ 0.   ,  1.25 ]])
```

```
1  Transf_matrix_inverse @ v_transformed
```

INVERSE TRANSFORMATION

```
array([1., 2.])
```

Resultant vector is same as original vector v

```
1  Transf_matrix_inverse @ v_transformed == v
```

```
array([ True,  True])
```

ANALOGS BETWEEN GEOMETRY AND LINEAR ALGEBRA:

It is essential that one understands the Linear Algebraic perspective of basic geometric concepts such as lines, planes and spaces when trying to understand vector and matrix operations over hyper dimensional spaces.

A line can be interpreted as the Span of a single vector. A plane can be interpreted as the span of 2 linearly independent vectors, a 3D space as the span of 3 linearly independent vectors, a 4D space as the span of 4 linearly independent vectors and so on . These definitions hold true, irrespective of the dimensionality of the feature space they exist in. In other words, the span of two linearly independent vectors will be a 2D plane even if those vectors exist in 100-D feature space (ie: The two linearly independent vectors being considered will each be 100 dimensional, but the space between them will be 2 dimensional). Similarly a line is always 1D irrespective of

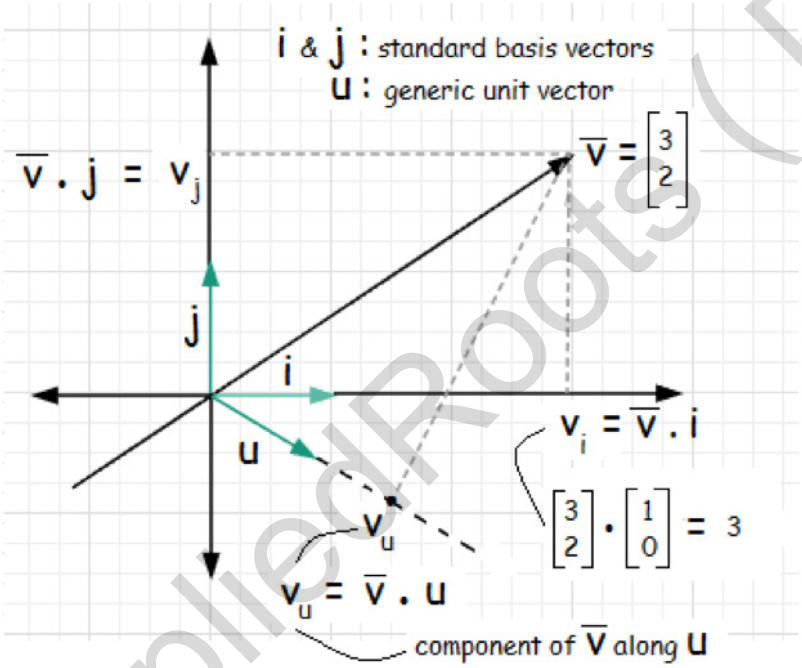
the feature space it exists in.

An “n” dimensional object can exist only in feature spaces that have a dimensionality greater than or equal to **“n+1”**.

Therefore we can say:

1. A line, which is the span of a single vector, can exist in feature spaces that are 2D and above.
2. A plane, which is the span of two linearly independent vectors, can exist in feature spaces that are 3D and above
3. A three dimensional space, which is the span of three linearly independent vectors, can exist in features spaces that are 4D and above (and so on for all higher dimensions).

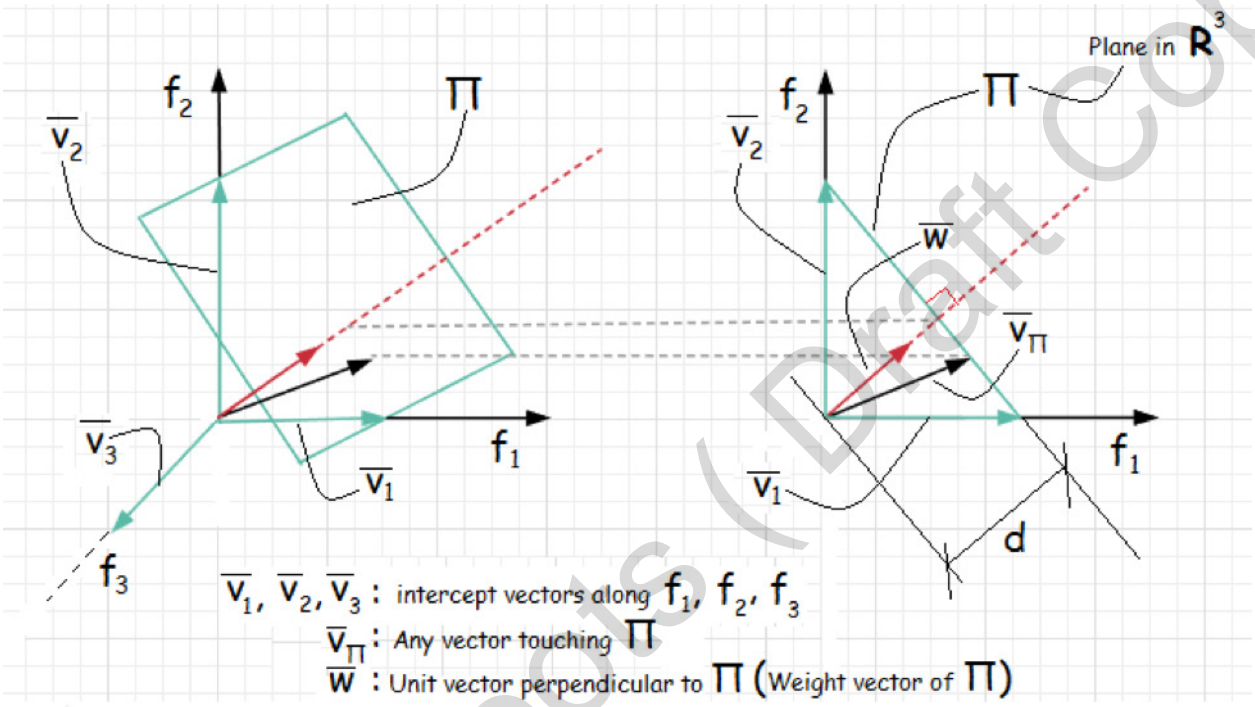
COMPONENT (PROJECTION) OF A VECTOR ALONG ANY DIRECTION IN THE FEATURE SPACE:



Consider the image shown above, the component of vector **v** along unit basis vector **i** is given by the dot product of **v & i**, which is equal to 3. Similarly the component of vector **v** along unit basis vector **j** is given by the dot product of **v & j,** which is equal to 2. Generalizing this concept, we can say that, the component of a vector along any direction is given by the dot product of that vector with the unit vector representing the desired direction.

Equation of A Plane (& The Weight Vector):

Every plane has one unique unit vector **w** associated with it, that is perpendicular to it (and passes through the origin). This unique unit vector is called the "weight vector" of the plane and forms the reference with which the equation of the plane is defined. Consider the image shown below:



we have:

$$\overline{v}_\Pi \cdot \overline{w} = \overline{v}_1 \cdot \overline{w} = \overline{v}_2 \cdot \overline{w} = \overline{v}_3 \cdot \overline{w} = d$$

ie: $\overline{v}_\Pi \cdot \overline{w} = \overline{v}_{intercept} \cdot \overline{w}$

ie: $\overline{w} \cdot (\overline{v}_\Pi - \overline{v}_{intercept}) = 0$ — General equation of a Plane

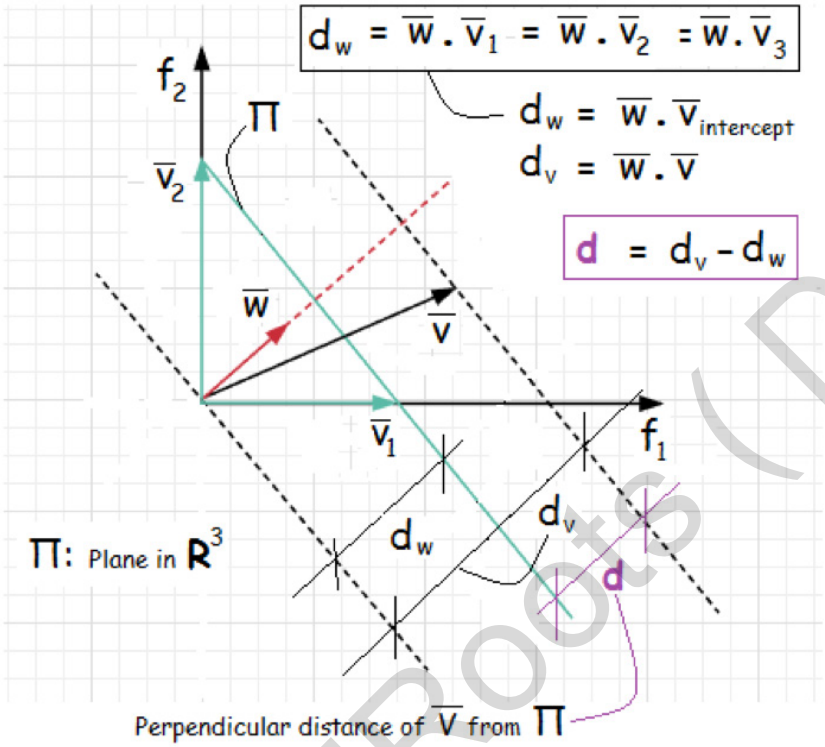
If the plane passes through the origin of the feature space then the intercept vector becomes a zero vector and the equation of the plane is then given by :

$$\overline{w} \cdot \overline{v}_\Pi = 0$$

Perpendicular Distance Between A Location (Or Vector) In The Feature Space & A Plane:

1. The dot product of the "weight vector" of the plane in consideration and vector v representing the location
2. The dot product of the weight vector and any intercept vector .

This is illustrated in the image below:



$$d = \overline{w} \cdot (\overline{v} - \overline{v}_{intercept})$$

General formula for distance of a location from a given Plane.

If the plane passes through the origin of the feature space then the intercept vector becomes a zero vector and hence the distance between any location in the feature space and the given plane becomes:

$$d = \overline{w} \cdot \overline{v}$$

9. FUNDAMENTALS OF MACHINE LEARNING – 1

FUNDAMENTALS OF MACHINE LEARNING - 1

ARTIFICIAL INTELLIGENCE & MACHINE LEARNING:

The field of Artificial intelligence is a composite of many sciences that deal with mankind's efforts at trying to achieve the ideal of creating machines that can emulate human capabilities and attributes. It spans a wide variety of perspectives - from purely computational requirements of designing & manufacturing the hardware framework that would support such a goal, to the algorithmic deconstruction of human tasks that involve "intelligence"- like the ability to extract information from data and make decisions or predictions based on it. Machine Learning is primarily concerned with the latter.

To clearly understand what Machine Learning is, it is important to understand what distinguishes it from the previous orientation one had, while trying to create autonomous decision making systems. Prior to Machine Learning, the general paradigm was :

1. Study/Analyse the data with respect to the type of decisions/predictions that need to be made.
2. Craft mathematical/Logical rules based on the insights gained from step one, using which, one could automate the decision making process, for further use on new data of the same kind.

Machine Learning in contrast is organized around the task of creating, optimizing and using algorithms that can automatically discover the mathematical/logical rules mentioned in step 2. In other words, Machine Learning primarily deals with algorithms, which when exposed to data, have the capability to "learn" the rules that go behind the decision making process.

Consider the simplistic idea of an "Apple-Pineapple classifier". Imagine that we want to create an app, which when given the weight and length of a fruit (either an apple or a pineapple), can correctly predict which fruit it is. In the "Rule Based Paradigm", one would first analyse a reasonably sized dataset of the type shown below and perhaps craft rules such as: "If weight greater than 0.3 kgs and length greater than 7.5 cms, then the fruit is a pineapple, else it is an apple". Once such a rule(or set of rules) is defined, it can be used to further classify any apple-pineapple data.

observations	weight_kg	length_cms	labels
1	0.15	5	apple
2	0.92	10	pineapple
3	1.20	15	pineapple
4	0.20	3	apple
5	1.00	11	pineapple
6	0.17	5	apple
7	0.92	12	pineapple
.	.	.	.
.	.	.	.
.	.	.	.

In the Machine Learning paradigm, we would use an algorithm which when exposed to the same dataset, would detect patterns/associations/rules that define:

1. Correlations between the fruit and its features (ie: measurements).
2. Differences in the measurements/features of one type of fruit from the other.

Once the algorithm "learns" these correlations and differences, it becomes capable of classifying all other apples and pineapples based on their weight & length. The advantages of Machine Learning become more apparent, when the size or dimensionality of the dataset being considered is too large and beyond the scope for human analysis.

SUPERVISED & UNSUPERVISED MACHINE LEARNING:

Machine Learning can be broadly classified into 2 major categories - supervised learning & unsupervised learning. The difference is based upon the fact that in the case of the former, the data contains "labels". The previous example involving the Apple-Pineapple classifier is an example of supervised learning. Here the data which the algorithm learns from, is already "labeled" with correct answers. The Machine Learning algorithm uses to its advantage, the underlying rules/correlations that could exist between the features and their corresponding labels, to arrive at its predictive mechanism. In other words, in supervised machine learning the learning algorithm is "supervised" by the labels that the data contains.

Supervised learning is used to perform 2 major tasks, which differ based on the nature of their labels:

1. Classification: The labels used in this case are categorical or discrete in nature. This involves tasks similar to the one just described, ie: detecting the “classes” of a collection of data-objects.
2. Regression: The labels used in this case are continuous in nature. Consider a dataset containing apple measurements, like the one shown below. The supervised algorithm is designed to learn from this kind of data and is expected to predict the weight of any apple given its length and width measurements.

observations	width_cms	length_cms	LABELS(weights_kg)
1	3.922590	4.138513	0.15
2	4.181110	7.932882	0.10
3	5.622229	6.533748	0.16
4	6.348631	5.132360	0.20
5	5.221969	5.685304	0.13
.	.	.	.
.	.	.	.
.	.	.	.

Unsupervised Learning is Machine Learning applied to unlabelled data. These algorithms are designed to optimize for criteria that are based upon the structure of the data itself (like detecting directions of maximum variance or recognizing density patterns). They are generally used to perform tasks like:

1. Dimensionality Reduction - Finding more interpretable lower dimensional representations of high dimensional data.
2. Clustering - Structuring the data in terms of the groups it might comprise of.

Supervised Learning algorithms are easier to optimize, since their labels provide a solid benchmark with reference to which the algorithm’s performance can be evaluated. Due to this fact, they are usually employed at the final stages of a machine learning pipeline, where one expects the machine to make a prediction of some sort.

Optimizing the performance of unsupervised models is not as straightforward, since there are no benchmarks with reference to which the algorithm can be evaluated. Often the only way to evaluate an unsupervised algorithm is to inspect its results manually. Due to these facts, unsupervised machine learning algorithms are usually employed in the initial data exploratory stages of machine learning pipelines.

PARAMETRIC AND NON PARAMETRIC METHODS:

Machine learning algorithms can be broadly classified into 2 major groups, based on the core axioms used during their implementation:

1. Parametric Machine learning algorithms.
2. Non Parametric Machine learning algorithms.

Parametric Algorithms have their basis in Linear Algebra. They use the concept of vectors and feature spaces to learn from the data provided. Non parametric algorithms base their learning on probabilistic or decision tree methods.

This chapter focuses on parametric methods and aims at equipping the reader with an understanding of the following algorithms:

1. Principal Component Analysis, which is an unsupervised machine learning algorithm that is used to perform dimensionality reduction.
2. K-means Clustering, which is an unsupervised machine algorithm used to analyse the structure of the data in terms of the groups/clusters they might comprise of.
3. Logistic Regression, which is a supervised machine learning algorithm used on data with categorical labels.
4. Linear Regression, which is a supervised machine learning algorithm used on data with continuous labels.

The comprehension, application and evaluation of these 4 algorithms will provide the reader with a general overview of the machine learning landscape.

DIMENSIONALITY REDUCTION & THE CURSE OF DIMENSIONALITY:

The curse of dimensionality refers to the problems one faces when working with high dimensional data. These issues mostly arise due to:

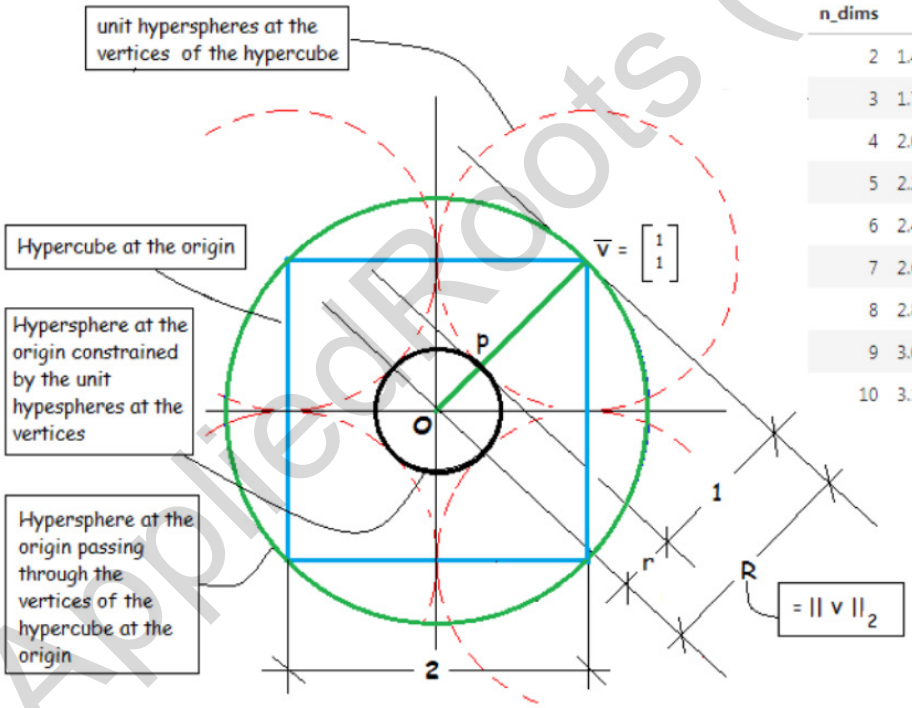
1. The exponential increase in computational overheads as the dimensionality of the data increases.
2. The counter intuitive, non linear structure of hyperdimensional spaces, which causes the efficacy of the distance measures used for analysis to become more and more irrelevant as the dimensionality increases.

With reference to the first point made above, as the dimensionality of the feature space increases, the amount of information required to describe a set of data points increases exponentially. Consider ten data points in a one dimensional space (ie: a line). Since the data is one dimensional, it can be easily represented using only 10 values. But if we add one more feature, the same number of data points will need 10 x 10 = 100 values to represent it. If we add a 3rd feature, we will need 10 x 10 x 10 = 1000 values to represent a dataset of the same size..

In general, an 'n' - dimensional feature space will require 100 ^ n values to represent 100 data points. This exponential increase in the amount of values required to describe the data, incurs huge computational overheads, making the processing of high dimensional data, time consuming and sometimes intractable.

With reference to the second point, consider the code & image shown below:

```
max_n_dims = 10
listO_dists = []
for n_dims in range(2, max_n_dims+1):

    v = np.ones(n_dims) #------- Coordinates of a particular vertex of the hypercube
    R = (np.array([i**2 for i in v]).sum())**(1/2) #----Norm 2 or euctedian distance
    listO_dists.append(R)

arryO_dists = np.array(listO_dists)
df = pd.DataFrame().assign(n_dims = range(2, max_n_dims+1),
                           R = arryO_dists,
                           r = arryO_dists-1)
df
```



n_dims	R	r
2	1.414214	0.414214
3	1.732051	0.732051
4	2.000000	1.000000
5	2.236068	1.236068
6	2.449490	1.449490
7	2.645751	1.645751
8	2.828427	1.828427
9	3.000000	2.000000
10	3.162278	2.162278

With reference to the image/code above, we have the following observations:

1. The black hypersphere at the origin is constrained to be tangential to the unit hyperspheres at the vertices of the blue hypercube also at the origin. Studying the output of the code shown above, we can see that the distance R of the vertices of the blue hypercube from the origin, will keep increasing as the dimensionality of the feature space increases, even though the length of each side of this hypercube remains the same = 2 units.

2. Distance ‘r’ of point P which is collinear to the vertex under consideration, will also keep increasing as the dimensionality of the feature space increases. This implies that the black hypersphere will become “relatively” closer and closer to the green hypersphere even though the distance between them will remain constant.

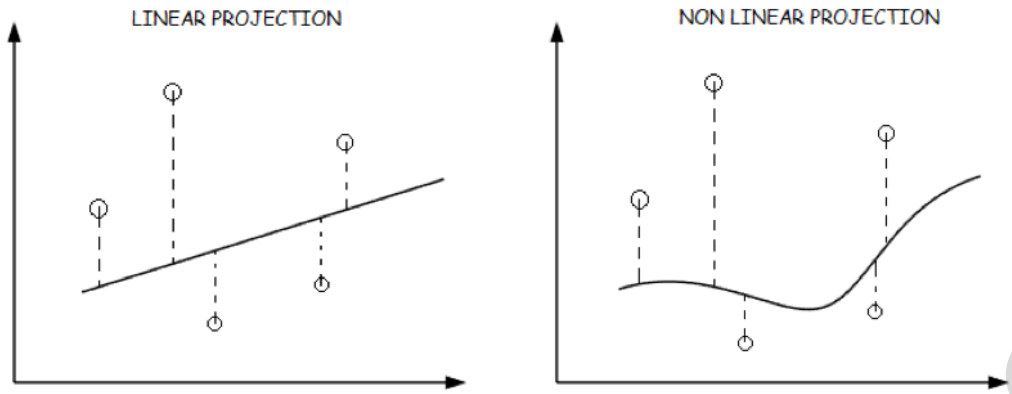
The observations made above are counter intuitive and express the nonlinear nature of hyperdimensional spaces. The black hypersphere keeps expanding as the dimensionality of the feature space increases, despite being constrained by the hyperspheres at the vertices of the blue hypercube. This also implies that, as the dimensionality of the feature space increases, the density of the data points that inhabit the feature space will tend to be concentrated at the circumference of the hypersphere that encompasses the data (ie: Hyperspaces tend to be more hollow at the center). Due to this property of hyperdimensional spaces, distance measures like Euclidean distance “break down” as dimensionality increases. The cosine distance measure exhibits better resilience to the curse of dimensionality but it too has its limits. One could experiment with other distance norms like Manhattan distance or norms greater than norm 2 or even fractional norms, to find the most suitable distance measure for a particular hyper dimensional feature space.

Dimensionality reduction techniques are algorithms that project high dimensional data to lower dimensional spaces, while retaining as much of the salient information as possible, thus making them more computationally tractable and more conducive to the distance measures being used. This is based upon the fact that:

1. Usually many features in high dimensional data are redundant or contain very little information.
2. Features in high dimensional data are often correlated and hence possess an intrinsic lower dimensional structure.

LINEAR & NON LINEAR PROJECTIONS:

The projection of higher dimensional data into lower dimensional spaces can be broadly classified into 2 categories - Linear & Non linear projections. Non Linear projections use specifically designed lower dimensional curved, non linear spaces to embed the high dimensional data in, whereas Linear projections are limited to only linear spaces. The images below expresses the essential difference between these 2 types of projections.



EIGENVECTORS & EIGENVALUES:

Given a Matrix, that transforms vectors from standard feature space to an alternate feature space, an Eigenvector is any unit vector in the standard feature space, which after being transformed maintains collinearity to itself. In other words, eigenvectors are those vectors that do not change their span after transformation.

The only change that an eigenvector could undergo during the Transformation is the scaling of its magnitude (or its norm 2 value). The amount by which an eigenvector is scaled after transformation is called its Eigenvalue. Therefore if **T** is a transformation matrix and if v is an eigenvector of **T**, then:

$$\mathbf{T}\,\bar{\mathbf{v}} = \lambda\,\bar{\mathbf{v}}$$

Where λ represents the eigenvalue of v.
Shown below is an example depicting the transformation of an eigenvector:

$$\begin{bmatrix} 1 & 0.772 \\ 0.772 & 1 \end{bmatrix} \begin{bmatrix} 0.707 \\ 0.707 \end{bmatrix} = \begin{bmatrix} 1.25 \\ 1.25 \end{bmatrix} = 1.768 \begin{bmatrix} 0.707 \\ 0.707 \end{bmatrix}$$

Note that:

1. All matrices do not have eigenvectors.
2. A matrix could have more than one eigenvector.

EIGENVECTORS OF SYMMETRIC MATRICES:

Symmetric transformation matrices have the unique property that an n x n symmetric matrix will have n eigenvectors vectors, all of which are mutually perpendicular to each other. This property makes them very useful while performing the dimensionality reduction transformation called Principal Component Analysis, mentioned earlier.

FEATURE STANDARDIZATION:

Feature standardization is a feature preprocessing technique which makes each feature in the dataset have zero-mean and unit standard deviation. This is achieved by subtracting all values of the random variable representing a feature, by its mean and then dividing it with its standard deviation. That is, given a random variable f representing a particular feature, the standardized version of f would have each of its values modified as shown below:

$$x' = (x - \mu)/\sigma$$

Where μ & σ are the mean & standard deviation of feature **f**.

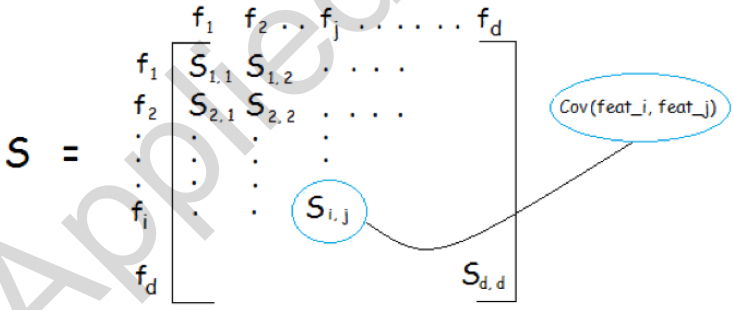
Standardization puts different features/variables on the same scale without losing any information. This process allows one to compare statistical scores between features having varied value ranges.

THE COVARIANCE MATRIX OF A DATASET:

Given a dataset containing **n** observations and **d** features, its corresponding matrix representation **D** will have **d** rows and **n** columns. The covariance matrix **S** of this data-matrix **D** is a **symmetric** matrix which is defined as:

$$S = M M^T$$

Where **M** is the row (feature) standardized form of **D**.
The Covariance matrix **S** has the form as shown below. The elements of the covariance matrix represent the covariance between all pairs of features of the dataset.



EIGENVECTORS OF A COVARIANCE MATRIX (PRINCIPAL COMPONENTS):

Covariance matrices have the following properties with reference to their eigenvectors:

1. They will have the same number of eigenvectors as the dimensionality of the data and each of those vectors will be perpendicular to each other (Because covariance matrices are symmetric by nature).
2. Their eigenvectors will represent directions of maximum variance within the data (ie: Directions in the standard feature space along which the data spreads out the most) and the magnitude of their eigenvalues will be proportional to the amount of variance along these directions. These eigenvectors are called the "Principal Components" of the data.

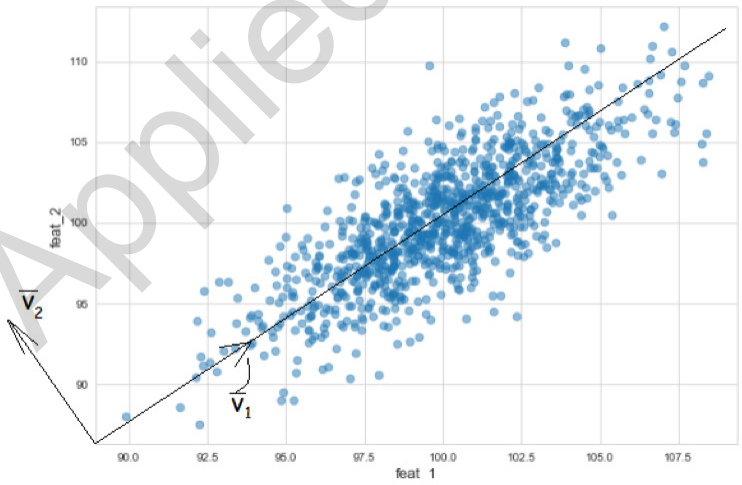
Consider the dataset shown below.

```
1  df = pd.read_csv('correlated_feats.csv')
2  display(df.shape, df.head())
```

```
(1000, 2)
```

	feat_1	feat_2
0	101.219056	101.735045
1	102.544758	100.934452
2	96.997143	98.337371
3	100.667230	101.477049
4	104.049367	102.948822

```
1  plt.figure(figsize = (10, 7))
2  fn_plot_scatter(df.feat_1, df.feat_2)
```



We can infer by studying the plot above that the data varies most along the directions v1 & v2. Hence by definition the eigenvectors of the covariance matrix of the above data, will have the same directions as v1 & v2.

PRINCIPAL COMPONENT ANALYSIS (PCA):

Principal Component Analysis is a dimensionality reduction algorithm which uses the eigenvectors of a covariance matrix to create linear projections of high dimensional data into lower dimensional spaces. It works as follows:

I. Derive the covariance matrix of the dataset after standardizing its features:

```
feat_1 = fn_standardize_feat(df.feat_1.values)
feat_2 = fn_standardize_feat(df.feat_2.values)

df_standardized = pd.DataFrame().assign(feat_1 = feat_1, feat_2 = feat_2)

cov_mat = df_standardized.cov()
cov_mat
```

	feat_1	feat_2
feat_1	1.001000	0.772725
feat_2	0.772725	1.001000

II. Determine the eigenvectors of the covariance matrix and arrange them in a matrix in descending order of their eigenvalues. This matrix now represents a transformation from the standard feature space into an alternate feature space, the basis vectors of which represent the directions of maximum variance (ie: Principal Components) of the Data.

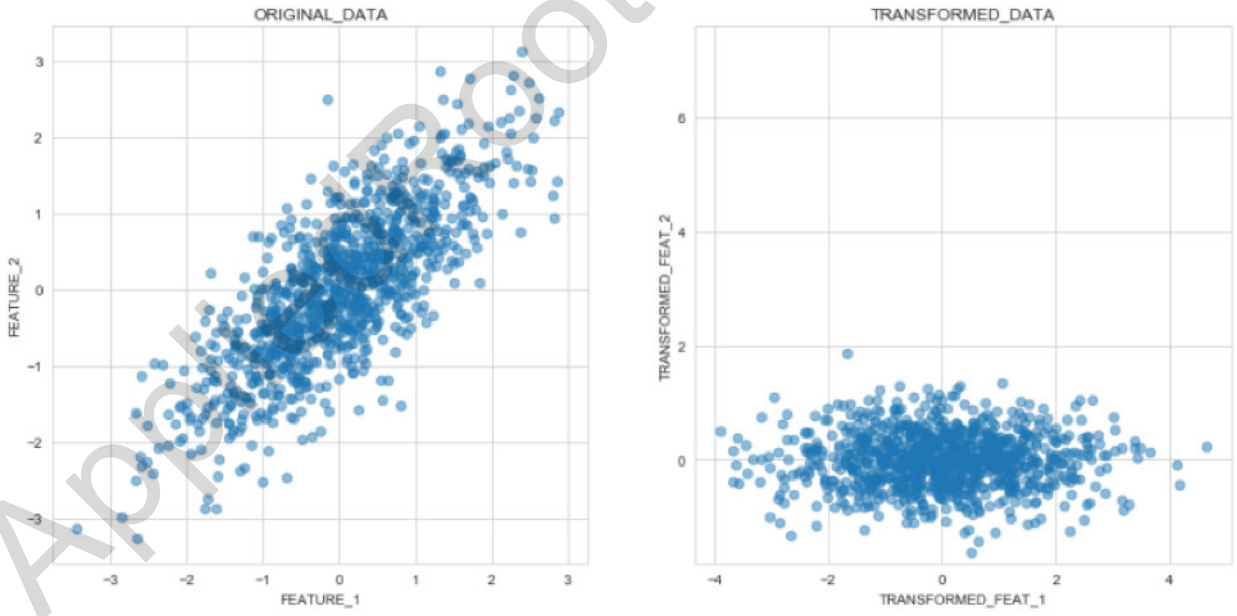
```
def fn_mat0_sorted_eigvecs(mat):

    eigvals, eigvecs  = np.linalg.eig(mat) #------Eigenvectors & Eigenvalues

    sorted_idxs = eigvals[::-1].argsort() #----- Indexs of sorted eigenvalues
    mat0_sorted_eigvecs = eigvecs[:, sorted_idxs]
    mat0_sorted_eigvecs = np.around(mat0_sorted_eigvecs, decimals = 4)

    return mat0_sorted_eigvecs
```

```
mat0_sorted_eigvecs = fn_mat0_sorted_eigvecs(cov_mat)

mat0_sorted_eigvecs
```

```
array([[-0.7071, -0.7071],
       [-0.7071,  0.7071]])
```

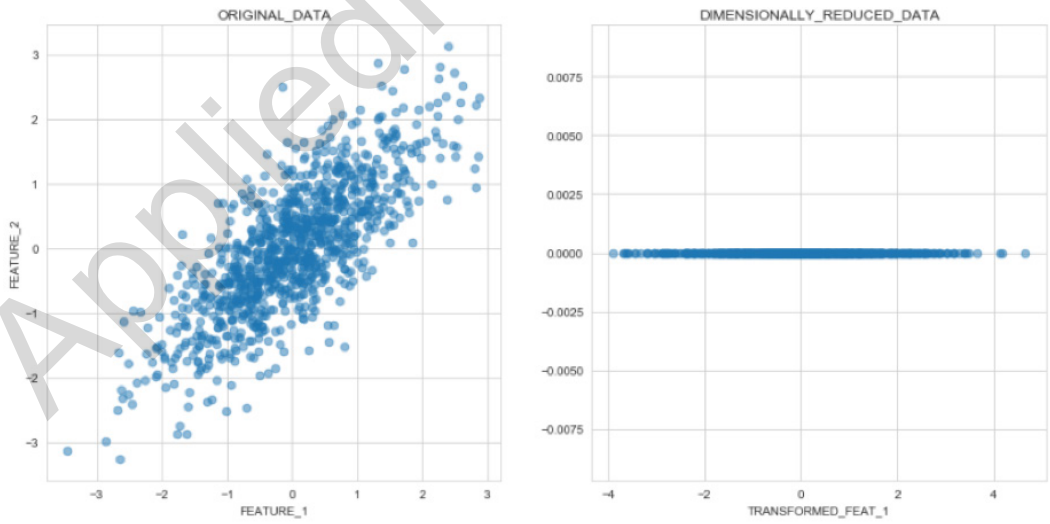
III. Transform the standardized dataset using this matrix of sorted eigenvectors. This has the effect of transforming the feature space such that it is now in alignment with the principal axes of the dataset (ie: in alignment with the eigenvectors contained within the transformation matrix).

```
# TRANSFORMATION TO ALTERNATE FEATURE SPACE:
matrix0_transformed_data = mat0_sorted_eigvecs @ df_standardized.values.T

# TRANSFORMED FEATURES:
transformed_feat_1 = matrix0_transformed_data[0, :]
transformed_feat_2 = matrix0_transformed_data[1, :]

# PLOTTING:
original_feat_pair = feat_1, feat_2
transformed_feat_pair = transformed_feat_1, transformed_feat_2

list0_pairs0_feats = [original_feat_pair, transformed_feat_pair]
list0_titles = ['ORIGINAL_DATA', 'TRANSFORMED_DATA']

axis_label_pair_1 = ['FEATURE_1', 'FEATURE_2']
axis_label_pair_2 = ['TRANSFORMED_FEAT_1', 'TRANSFORMED_FEAT_2']
list0_pairs0_axis_labels = [axis_label_pair_1, axis_label_pair_2]

fn_subplots_scatter(list0_pairs0_feats, list0_titles, list0_pairs0_axis_labels)
```



IV. Linearly project the transformed data into a lower dimensional space. This is achieved by ignoring those features within the transformed data that represent very little of the total variance. The eigenvalue of any principal axis divided by the sum of all eigenvalues, gives the percentage of variance explained by that particular principal axis. Since in case of the PCA algorithm the transformation matrix contains eigenvectors sorted in terms of the percentage of variance they explain, the act of simply ignoring the higher dimensional features of the transformed feature space is the same as linearly projecting those higher dimensions into lower dimensional space. This is expressed in the plot shown below:

```
# DROPPING FEATURE(S) WITH LESSER VARIANCE:
transformed_feat_2 = np.zeros(len(transformed_feat_1))

# ORIGINAL FEATURES:
original_feat_pair = feat_1,  feat_2

# TRANSFORMED FEATURES:
transformed_feat_pair = transformed_feat_1, transformed_feat_2

# PLOTTING:
list0_pairs0_feats = [original_feat_pair, transformed_feat_pair]
list0_titles = ['ORIGINAL_DATA', 'DIMENSIONALLY_REDUCED_DATA']

axis_label_pair_1 = ['FEATURE_1', 'FEATURE_2']
axis_label_pair_2 = ['TRANSFORMED_FEAT_1', '']
list0_pairs0_axis_labels = [axis_label_pair_1, axis_label_pair_2]

fn_subplots_scatter(list0_pairs0_feats, list0_titles,
                    list0_pairs0_axis_labels, same_scale = False)
```



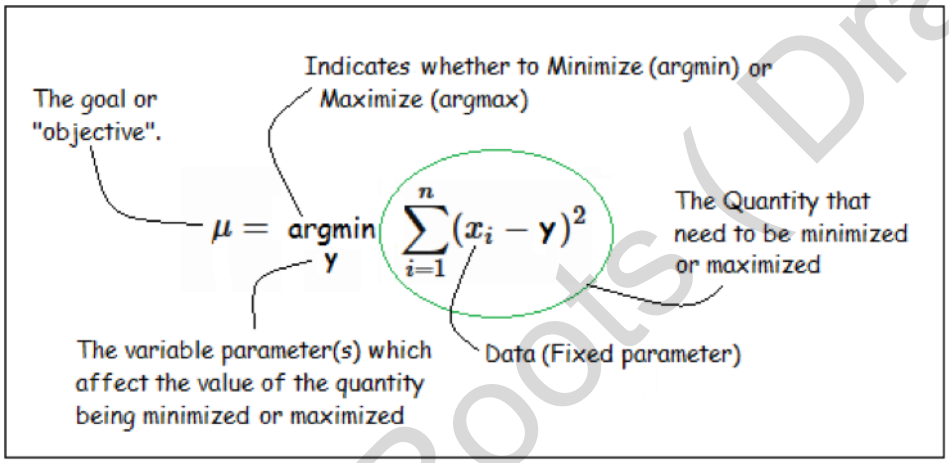
The 4 steps outlined above form the basis of the PCA algorithm. In the example above, we have used a 2D dataset solely for the purposes of demonstration. The same method could be used to dimensionally reduce a dataset of any dimensionality.

OBJECTIVE FUNCTIONS:

Objective functions are the mathematical expression of the task/problem that an algorithm aims to solve. They express the problem in terms of:

1. The fixed & variable parameters that constitute the particular task
2. The relationship between these parameters and a certain value that is being sought (ie: The Objective).
3. The variable parameter(s) that need to be optimized so as to achieve the objective.
4. The type of optimization required: Minimization or Maximization.
5. The constraints (conditions) within which the objective should be achieved.

Consider the simple task of finding the mean of a set of data points. This could be expressed in terms of an objective functions as:



The above objective functions basically indicates that we need to find the optimal **y** which minimizes the composite value to the right of the "argmin" term.

THE PCA OBJECTIVE FUNCTION:

Since Principal Component Analysis is about finding the direction of maximum variance within the feature space of the dataset, its objective function can be defined as:

$$\bar{u} = \underset{\bar{u}}{\text{argmax}} \frac{1}{n} \sum_{i=1}^{n} \bar{x_i} \cdot \bar{u} \qquad \text{such that: } ||\bar{u}|| = 1$$

In simple words, the objective function tells us that we need to find that unit vector, along which the average projection of the dataset is maximum (direction of maximum variance). We can achieve this objective using the concept of eigenvectors and the covariance matrix as shown earlier.

K-MEANS CLUSTERING:

K-means algorithm is an Unsupervised Machine Learning algorithm that is used to partition the dataset into **'K'** pre-defined subgroups (clusters). The algorithm tries to find the best possible **K** centroids (cluster centers) which minimizes the total "Inertia" or " Within Cluster Sum of Squares " criterion for the dataset. Note that the algorithm expects one to specify the number of clusters the dataset comprises of, before it can initiate the clustering process.

INERTIA OR WITHIN CLUSTER SUM OF SQUARES (WCSS):

Given a particular cluster, the WCSS is defined as the sum of squared distances of all the points within that cluster to its centroid.

$$\underset{\text{(For cluster S)}}{WCSS} = \sum_{\mathbf{x} \in S} \|\mathbf{x} - \boldsymbol{\mu}\|^2$$

Where **μ** is the centroid of the points in cluster **S**.

Essentially, WCSS measures the variability of the data points within each cluster. In general, a cluster that has a small sum of squares is more compact than a cluster that has a large sum of squares.

WCSS is influenced by the number of data points, as the number of data points increases, the sum of squares increases. This means that WCSS is often not directly comparable across clusters with different sizes. To compare within-cluster variabilities of different sized clusters, the average within cluster squared distance of the data points from the centroid (ie: the average squared distance of the data points from the nearest centroid) is a better measure.

K-MEANS CLUSTERING OBJECTIVE FUNCTION:

The objective function of the K Means Clustering algorithm can be expressed as:

$$S = \underset{\boldsymbol{\mu}}{\arg\min} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2$$

Where $\boldsymbol{\mu}_i$ is the mean (or centroid) of points in cluster $\mathbf{S}_i$

A purely mathematical solution to the above objective function (like in the case of PCA) is computationally intractable and hence heuristic algorithms that have the same effect as minimizing the total WCSS, while at the same time guaranteeing quick convergence, are employed to perform the clustering operation.

LLOYD'S ALGORITHM:

Lloyd's algorithm is one of the most commonly used heuristic algorithms for achieving convergence while trying to minimize the WCSS of a dataset. It consists of the following steps:

1. Initialize K centroids by randomly selecting K unique points from the feature space.
2. Assign a centroid for every data point in the dataset, based on least distance. Thus K initial clusters are created with reference to these randomly initiated centroids.
3. Recompute centroids for every cluster thus obtained.
4. Repeat steps 2 & 3 until convergence (ie: Distance between the newly computed centroids and the previous ones fall below a certain predefined tolerance level).

Consider the dataset shown below:

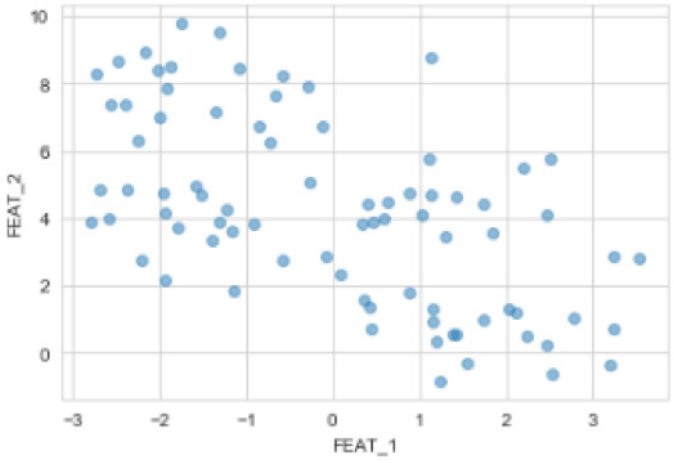
```
df_clusters = pd.read_csv('2_clusters.csv')

display(df_clusters.shape, df_clusters.sample(5), df_clusters.describe())
```

```
(80, 2)
```

	FEAT_1	FEAT_2
44	-1.884102	8.511893
24	-1.516404	4.703753
66	2.232694	0.495883
12	2.784358	1.026647
60	1.840706	3.561622
39	1.134889	8.779940
48	-1.399992	3.319872
30	-1.922588	7.867291

	FEAT_1	FEAT_2
count	80.000000	80.000000
mean	-0.013888	4.179812
std	1.778348	2.772568
min	-2.797389	-0.828619
25%	-1.619112	1.813484
50%	0.008101	4.050057
75%	1.312705	6.263594
max	3.543520	9.764992

```
feat_1, feat_2 = df_clusters.FEAT_1, df_clusters.FEAT_2
xlabel, ylabel = 'FEAT_1', 'FEAT_2'

fn_plot_scatter(feat_1, feat_2, xlabel, ylabel)
```



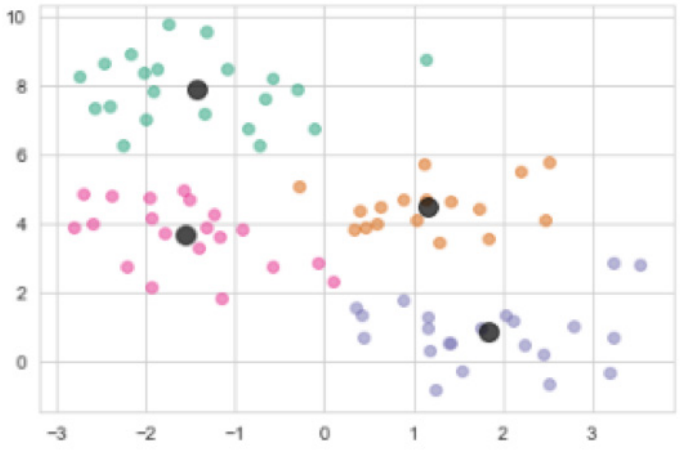
Shown below is an implementation of Lloyd's algorithm applicable to 2D data, for demonstration purposes. The function fn_K_means_clustering takes as its arguments:

1. The 2D dataset arranged in matrix format.
2. The number of centroids, K.
3. The "tolerance level" mentioned in step 4 earlier.

```
def fn_K_means_clustering(data_matrix, n_centroids, list0_centroids = [],
                          max_iteration = 1000, tolerance = 0.01):
    '''
    RETURNS:
    1. dict0_clusters = {cluster_n : data points belonging to cluster}
    2. list0_centroids
    '''
    # STEP 1:
    if len(list0_centroids) == 0:
        list0_centroids = fn_random_list0_centroids(data_matrix, n_centroids)

    iter_n = 1

    # STEP 4:
    while True:
        # STEP 2:
        dict0_clusters = fn_assign_datapts_to_centroids(data_matrix, list0_centroids)
        # STEP 3:
        list0_new_centroids = fn_compute_new_centroids(dict0_clusters, n_centroids)

        arry0_new_centroids = np.array([*list0_new_centroids]).T
        arry0_centroids = np.array([*list0_centroids]).T
        difference = np.around(np.abs(arry0_new_centroids - arry0_centroids).mean(), decimals = 2)

        if (iter_n >= max_iteration) or (difference <= tolerance):
            break
        list0_centroids = list0_new_centroids
        iter_n += 1

    return dict0_clusters, list0_centroids
```

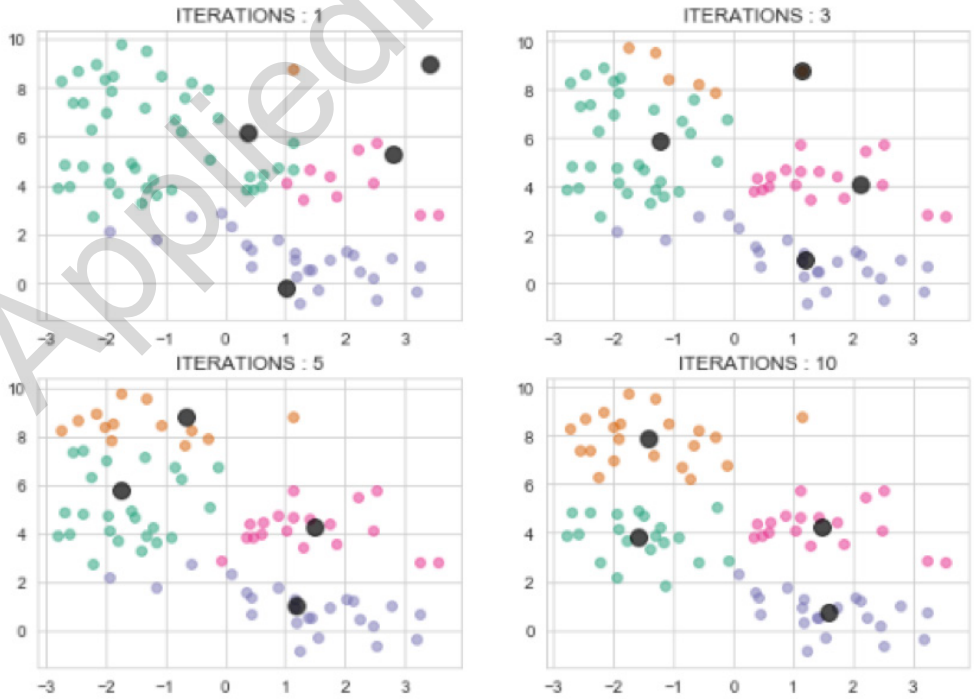
Choosing K = 4 for the dataset above, we have:

```
data_matrix = df_clusters.values.T
n_centroids = 4

dict0_clusters, list0_centroids = fn_K_means_clustering(data_matrix , n_centroids)

fn_plot_clusters(dict0_clusters, list0_centroids)
```



Shown below is the clustering status at various iterations of the algorithm:

```
data_matrix = df_clusters.values.T
n_centroids = 4

fn_show_clustering_process(data_matrix, n_centroids)
```



Determining The Best 'K':

The K Means clustering algorithm is incapable of determining the number of clusters that exist within the data. It requires that this information be provided by the user. One of the commonly used methods devised to aid the user arrive at the correct number of clusters is the "Elbow Method". It consists of the following steps:

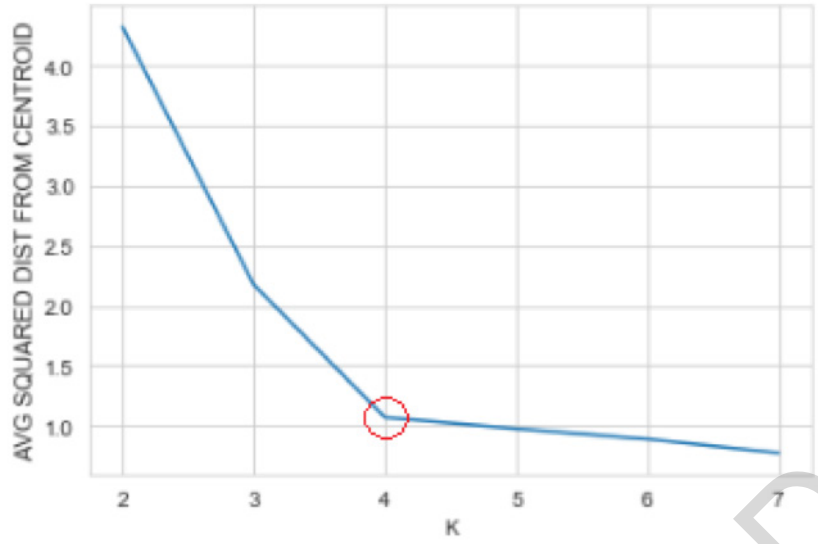
1. Perform K-Means Clustering over a range of probable K values.
2. Compute the "average squared distance from nearest centroid" for the entire dataset, at each of these K values.
3. Plot the values computed in step 2 against their corresponding K values.
4. Choose the K that corresponds to the "elbow" in the plot (The K value at which the slope of the curve changes the most).

Shown below is the code for computing the "average squared distance from nearest centroid" given clustered data along with their corresponding centroids:

```
def fn_avg_squared_dist_from_centroid(list0_centroids, dict0_clusters):
    '''
    dict0_clusters = {cluster_n : [data_pts belonging to that cluster_n]}
    '''

    list0_all_squared_dists = []
    for idx, centroid in enumerate(list0_centroids):
        cluster_pts = dict0_clusters[idx]

        # Within_Cluster_Squared_Dist (WCSD):
        list0_WCSDs = [(np.linalg.norm(centroid - datapt))**2 for datapt in cluster_pts]
        list0_all_squared_dists = list0_all_squared_dists + list0_WCSDs

    avg_squared_dist_from_centroid = np.array(list0_all_squared_dists).mean()
    return avg_squared_dist_from_centroid
```

In case of the dataset above we have:

```
listO_Ks = list(range(2, 8))

fn_elbow_method(listO_Ks, data_matrix)
```



DUNN INDEX:

The Dunn Index value is another measure for determining effectiveness of a clustering technique. It is defined as follows:

$$\text{DUNN INDEX} = \frac{\text{min(intercluster distance amongst all clusters)}}{\text{max(intracluster distance amongst all clusters)}}$$

Where

- Intercluster distance is the distance between centroids of two clusters and
- Intracluster distance is the distance between two two points within the same cluster

The greater the value of the Dunn index, the better is the clustering achieved by the clustering technique.

LOCAL OUTLIER FACTOR (OUTLIER DETECTION) :

Local Outlier Factor is a measure to determine how much of an outlier a data point is with respect to the rest of the data which it is a part of. It is founded on two other measures, defined below:

1. REACHABILITY DISTANCE:

Reachability distance is a measure of whether some given point **x_i** could be considered as a "neighbour" of another point **x_j.** It is defined as:

$$R_d(x_i, x_j) = \max(\text{dist}(x_i, x_j),\ k\_\text{dist}(x_j))$$

where: $k\_\text{dist}(x_j)$ is the distance of k^{th} Nearest Neighbour of x_j

2. LOCAL REACHABILITY DISTANCE (LRD):

Local Reachability Distance is a measure that gives us information about the "degree of isolation" of a data point with respect to the dataset. Lower the value, the more isolated the data point under consideration is, with respect to the rest of the data. It is defined as:

$$LRD(x_i) = \frac{1}{\text{average}\left(\sum_j R_d(x_i, x_j)\right)}$$

where: j represents datapoints in the neighbourhood of point x_i

Local outlier factor (LOF) is then defined as follows:

$$LOF(x_i) = \frac{\text{average LRD of neighbours of } x_i}{\text{LRD of } x_i}$$

10. DIMENSIONALITY REDUCTION (SVD/PCA)

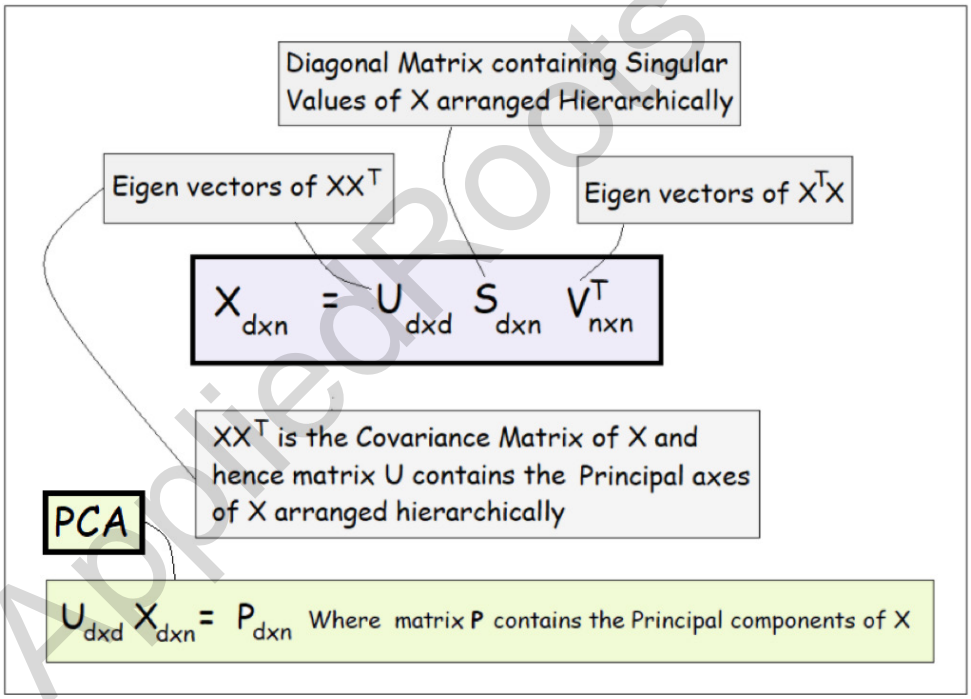
DIMENSIONALITY REDUCTION (SVD/PCA)

SINGULAR VALUE DECOMPOSITION (SVD):

Matrix decomposition refers to the factorization of a matrix into a product of two or more matrices. Singular Value Decomposition is an extremely handy matrix decomposition method, which is used in the field of machine learning for various purposes such as:

1. Feature Extraction
2. Dimensionality Reduction
3. Image Compression
4. Document Corpus Analysis (Latent Semantic Analysis)
5. Recommender Systems

Singular Value Decomposition basically decomposes any matrix **X** into the product of three matrices **U, S** and **V** as shown in the image below. Note that the matrix **X** contains **n** vectors of **d** dimensions each, arranged as per the linear algebraic convention (columns represent vectors).



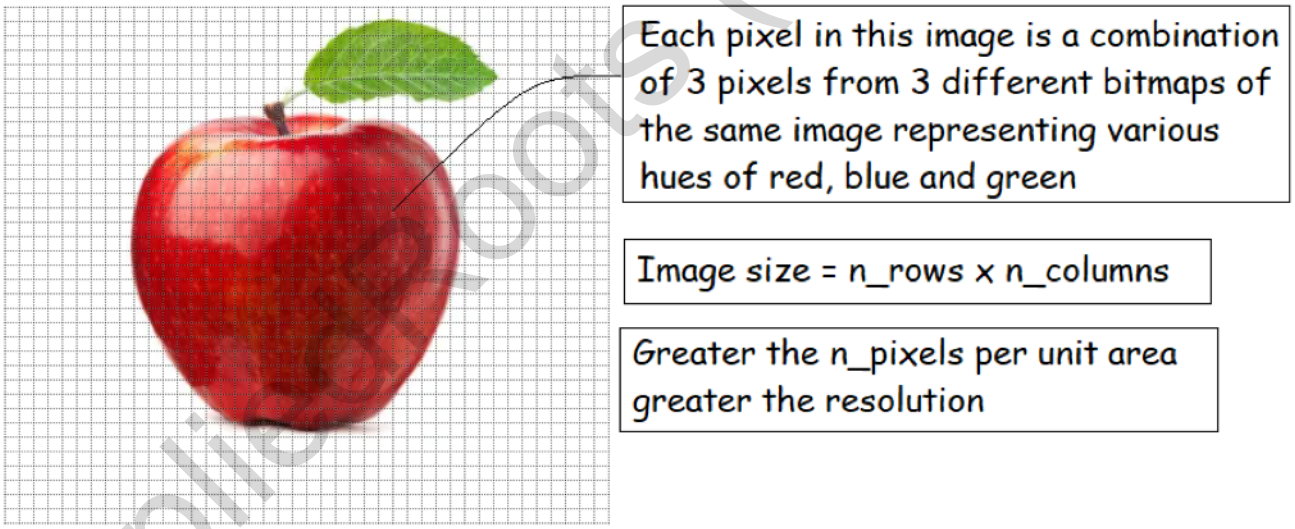
From our previous discussions about **Principal Component Analysis**, we know that the eigenvectors of the covariance matrix of **X**, represents the **principal axes** of **X**. Therefore if the matrix **X** is transformed using matrix **U**, we will get the **principal components** of **X**. This is the basis of how we perform dimensionality reduction using SVD.

The Singular Values of a matrix are simply the square root of its eigenvalues and hence they are proportional to the degree of variance explained by the principal components they correspond to (each column in **S** contains the singular value associated with the corresponding column in **U**) . If the number of vectors in X (ie: **n**) is greater than the dimensionality (ie: **d**) of the vectors, then the singular values beyond the **d**th column in **S** will be zero and vice-versa.

The details of how the matrix decomposition is computed will not be discussed as it is beyond the scope of the book, instead we will be using the numpy and scikit learn libraries to perform the decomposition for us.

SVD AND IMAGES:

A grayscale image is represented by a matrix of pixel values, where each pixel represents some shade of grey. A color image is represented by a set of 3 bitmaps, where each bitmap contains pixels that encode various hues of red, blue and green respectively. In other words, a color image is represented by a blue bitmap, a green bitmap and a green bitmap. These 3 bitmaps when combined together create a color image.



FEATURE EXTRACTION AND DIMENSIONALITY REDUCTION:

Feature extraction means finding a representation of your data that is better suitable for analysis than using it in its raw form. This holds more true for image data, which are usually made up of thousands of pixels, which are meaningful only if viewed as a whole. For the sake of demonstration, we will be using images from "**Labelled faces in the wild**" dataset, which contains faces of celebrities from the early 2000's downloaded from the internet. This dataset is available with the scikit learn library as one of their sample datasets.

The dataset can be accessed using the following code:

```
faces = fetch_lfw_people(min_faces_per_person = 20, resize = 0.7)
faces.data.shape
```

```
(3023, 5655)
```

The "min_faces_per_person" kwarg specifies that only images of celebrities that have a minimum of 20 images will be used/downloaded, the "resize" kwarg reduces the image size by 70% for the sake of faster processing. This results in images of size **87 x 65** (ie: 5655) pixels. The images in the dataset are represented as row vectors, by flattening the pixel matrices that represent the image. Each row thus represents an image and each pixel represents a feature.

For the sake of ease of explainability, some helper functions have been created that simplify the process of exploring the image data and for other purposes such as performing SVD. These functions are quite simple to understand and follow the same conventions followed previously.

DATA EXPLORATION:

As discussed in the previous chapters, the scikit library uses the "tabular" convention to represent the data, but for the sake of our discussion we will be using the "linear algebraic" convention, and hence all datasets within scikit learn need to be transposed before usage in any linear algebraic calculations.

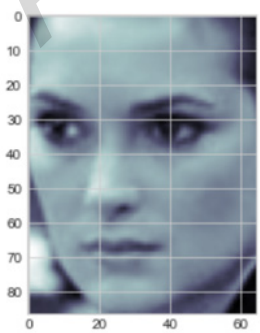
```
X = faces.data.T
X.shape
```

```
(5655, 3023)
```

To be able to view any image one would have to resize the vector representing the image back into a image matrix as shown below:

```
plt.imshow(X[:,0].reshape(87, 65), cmap=plt.cm.bone)
```

```
<matplotlib.image.AxesImage at 0x2097eebf730>
```



Shown below is a sample set of images randomly chosen from the dataset:

```
matrix0_imgs = X

fn_show_imgs(matrix0_imgs, n_imgs = 30, random_ = True)
```

EIGENFACES (FEATURE EXTRACTION):

As mentioned earlier, each image in the dataset **X** is represented as a vector, by flattening the pixel matrix of the image and hence each image pixel could be considered as a feature. In other words, each image is represented as a vector in a **87 x 65** dimensional feature space, where each axis represents a particular pixel and could have any value between from 0 to 255. Such features are not particularly useful as they do not contain any interpretable information about the images in the dataset as a whole.

Eigenfaces is the name given to the vectors in the matrix **U**, which were obtained by applying SVD on the data matrix X which contains the image vectors (ie: Eigenfaces are the image representation of the principal axes of X). The images shown below are the image representation of the first 30 principal axes (eigenfaces) of X.

```
matrix0_imgs = X
U,S,V = fn_SVD(matrix0_imgs)

U.shape, S.shape, V.shape
```

```
((5655, 5655), (3023, 3023), (3023, 3023))
```

```
matrix0_imgs = U

fn_show_imgs(matrix0_imgs, n_imgs = 30, random_ = False)
```

These eigenfaces or principal axes can be considered as features of the images contained in X. Thus SVD helps us extract more relevant features of the images as compared to just using the image pixels as features. Any image in X can now be expressed as a linear combination of these eigenfaces.

DIMENSIONALITY REDUCTION (PCA):

Since we have matrix **U** which contains the **principal axes of X,** we can use it to transform **X** (ie: represent the vectors contained in **X** in the feature space represented by **U**) and thus get the **principal components** of X as shown below:

```
P = U @ X
P.shape
```

```
(5655, 3023)
```

Since the magnitudes of the singular values indicate the degree of variance in data X explained by the principal axes they correspond to, we can reduce the dimensionality of matrix P using the following method:

1. Decide how much information (ie: variance) you want capture while reducing the dimensionality of the data (say: 90%)
2. Check how many singular values will be required to do this (ie: Sum of all singular values expresses 100% variance in data, therefore how many singular values will be required to express 90% variance)
3. Transform data X using matrix U to get principal components.
4. Keep only those many dimensions of the transformed data as the number of singular values determined in step 2

For the image data X, the first 1000 singular values explain around 90% of the data variance. This expressed in the code shown below:

```
S[:1000].sum()/S.sum()
```

```
0.8993847
```

Therefore the dimensionally reduced form of X would be:

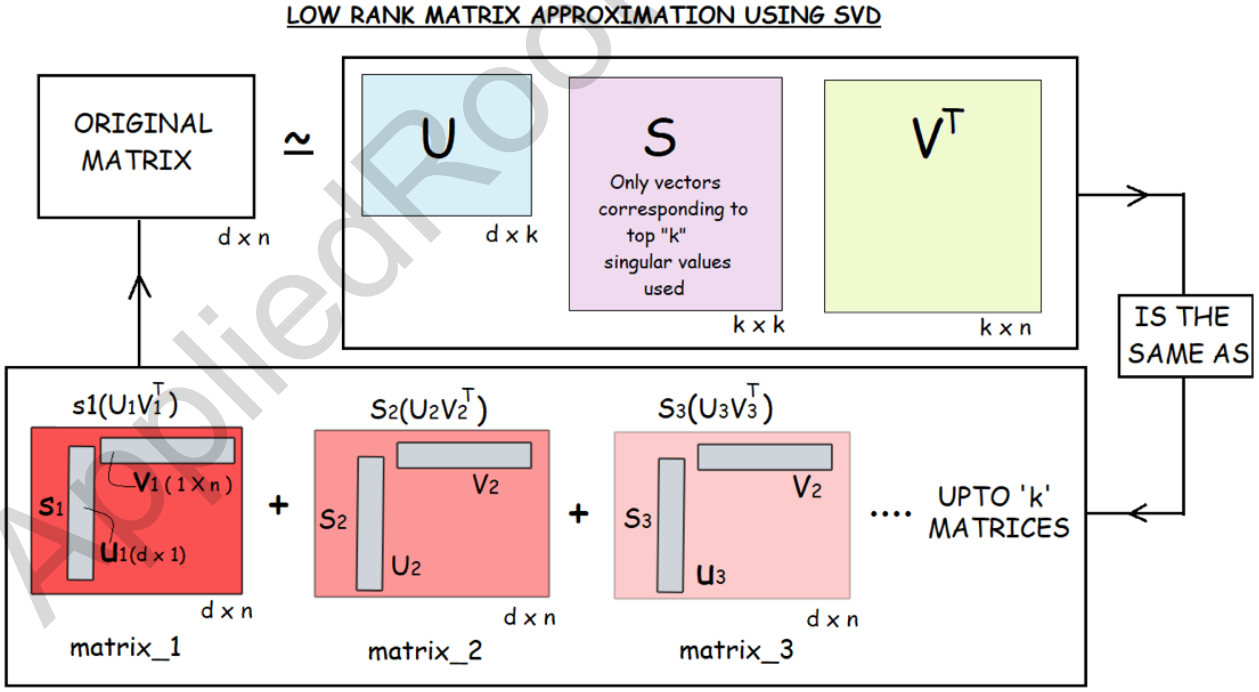
```
Dim_reduced_data = P[:1000]
Dim_reduced_data.shape
```

```
(1000, 3023)
```

As shown above, we have dimensionally reduced a 5655 dimensional dataset to just 1000 dimensions while conserving 90% of information.

LOW RANK MATRIX APPROXIMATION USING SVD (TRUNCATED SVD):

SVD can also be expressed as the sum of outer dot products (matrix multiplication of a column matrix and a row matrix) of the vectors contained in matrices **U & V**. By choosing to consider only the top "k" singular values, we can reconstruct an approximation of the original matrix as shown below:

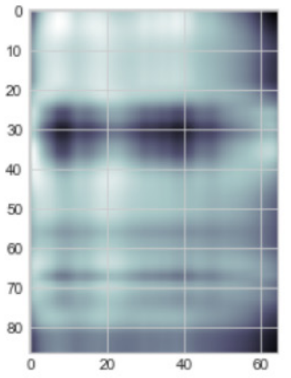


Note that each of these outer dot products ($\mathbf{UV^T}$) produces a **dxn** matrix and is scaled by their corresponding singular values **s**. The closer the value of 'k' is, to the original number of singular vectors, the better is the reconstruction (approximation) of the original matrix.

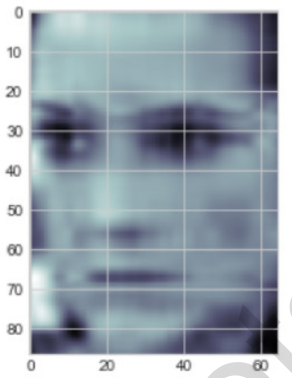
SVD can be applied to an individual image as well, instead of applying it to a dataset of multiple images, since each image itself can be expressed as a matrix. Thus given an **m x n** image matrix **I**, we can derive another **m x n** matrix **I'** which is an approximation of **I** using the technique shown above. This is demonstrated in the example shown below:

```
img = X[:, 0].reshape(87, 65)
```

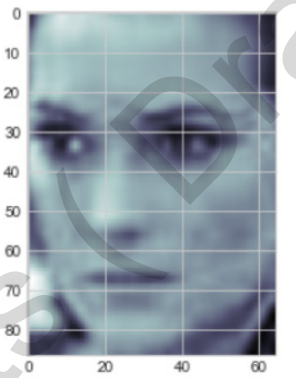
```
i = 1
fn_img_compress(img, i)
```



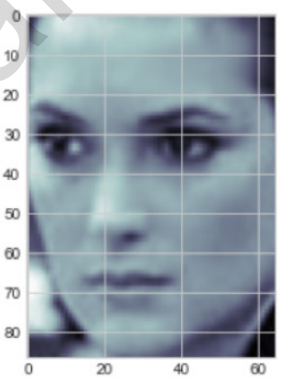
```
i = 5
fn_img_compress(img, i)
```



```
i = 10
fn_img_compress(img, i)
```



```
i = 20
fn_img_compress(img, i)
```



As can be seen from the four images shown above, using the summation of the first twenty $\mathbf{sUV^T}$ values, we can arrive at an image that is as good as the actual image. This means that we are using only (87 + 65) x 20 = 3040 values to describe the image instead of 87 x 65 = 5655 values. Hence we can achieve quite an advantageous amount of **data compression** using SVD.

The optimal number of "k" can be more formally deduced by choosing the percentage of variance one wants to conserve (say 85%) and then finding out how many singular values will be needed to do this. The method for doing this was previously explained under the previous sub topic "DIMENSIONALITY REDUCTION (PCA)" in this chapter.

The technique just described is called "Low Rank Matrix Approximation" because the number of linearly independent vectors in the reconstructed matrix will be lesser than that of the original matrix. Each singular value obtained from SVD decomposition corresponds to a linearly independent vector in the original matrix. Choosing to use only the top "k" singular values, results in a matrix reconstruction using only top "k" singular values and their corresponding vectors in matrices U and V. Thus the reconstructed matrix will have a rank of "k" which is lesser than the original rank of the matrix.

11. T-DISTRIBUTED STOCHASTIC NEIGHBOR EMBEDDING (T-SNE)

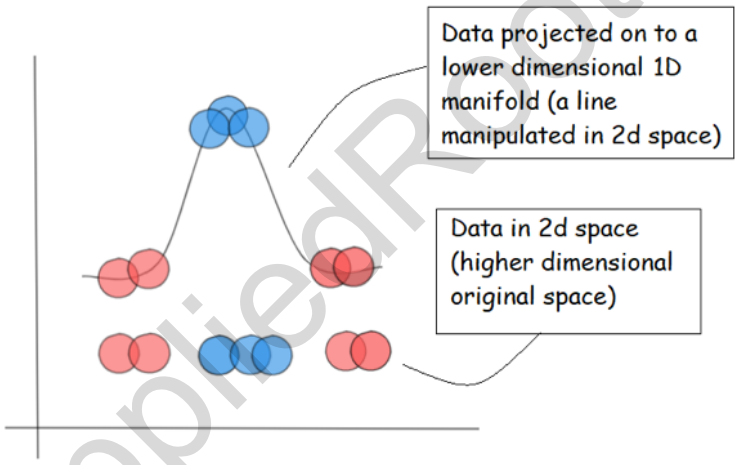
T-DISTRIBUTED STOCHASTIC NEIGHBOR EMBEDDING (T-SNE)

tSNE is a Dimensionality Reduction technique which belongs to a subclass of unsupervised learning algorithms called manifold learning algorithms. It is a modification/improvement of the stochastic neighbor embedding technique **(SNE).**

To understand what a manifold is, imagine a plane within a 3D space. The plane is 2D but it could be folded and twisted in a 3 dimensional fashion. Now consider that the 3D space has data points that contain clusters. Manifold learning attempts to do one of the following:

1. Discover the right manipulation of the 2D plane, such that when the 3D points are projected onto it, the distances/similarities of the data points in the original high dimensional feature space are replicated as closely as possible on the 2D surface. In other words, the cluster structure of the data points in the high dimensional original space should be preserved as much as possible on the 2D plane. The manipulated lower dimensional plane/space in the description above is called a **manifold**.

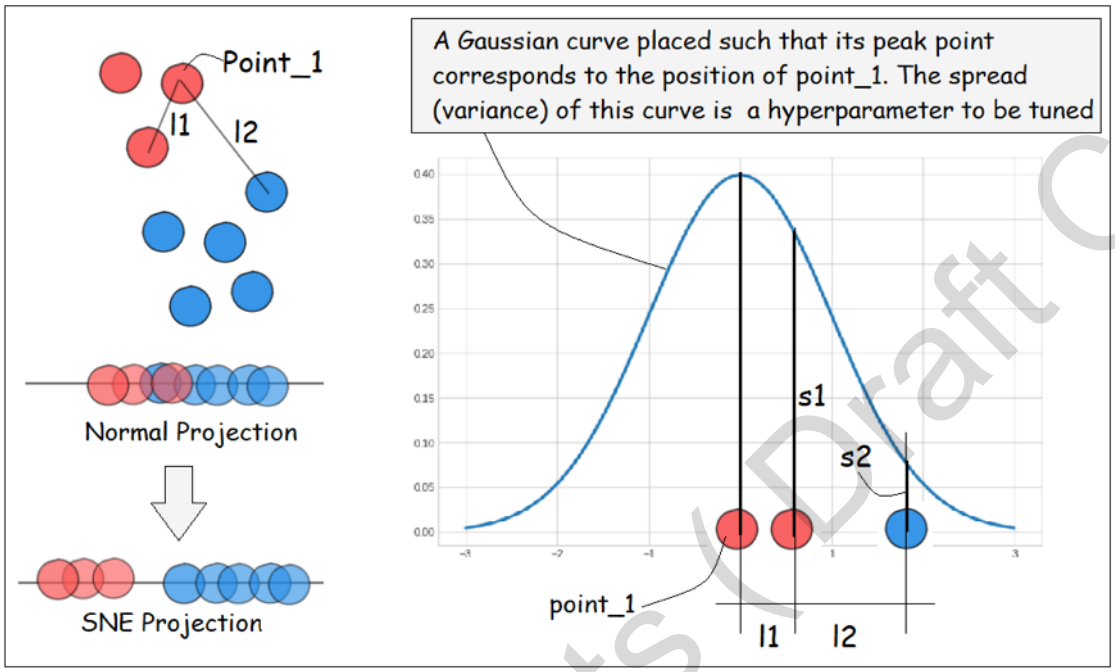
The image expresses the above concept using a 2D original space and a 1D manifold.



2. Randomly initiate the same number of data points on a 2D plane and then move the points along this flat manifold till the positions of the data points best replicate the similarities/differences of the data points in the original high dimensional space. Here the manifold is not manipulated but instead the positions of the data points on the manifold are. tSNE uses this method.

Before exploring **tSNE**, let us understand what Stochastic neighbor embedding **(SNE)** is. Consider the image shown below. It could be understood in the following way:

- In the upper left corner we see a dataset with two clusters - blue and red. If these data points are projected onto a 1D line, the clustering information is lost.



- Consider **point_1** in the image. Its distances (**l1**) from points within its own cluster will be generally smaller than its distances (**l2**) from points in the other cluster.
- Stochastic neighbor embedding **(SNE)** tries to proportionately replicate the **l1 & l2** distances shown above, on the 1D line (ie: similarity information between data points) thus trying to preserve the clustering. It does this for all pairs of data points present in the higher dimensional space

In essence, the **SNE** algorithm consists of the following steps:

1. Consider a single data point. Now the distance of this datapoint from all other data points can be projected onto a gaussian as shown in the right side of the above image. This involves creating a gaussian distribution centered around the data point under consideration. Then plot the distances **l1, l2** etc of another data point laterally, as shown above. Project this lateral location onto the gaussian curve.
2. The distances **s1, s2**, etc that results from step above can be considered as a similarity measure of any other data point with respect to this current point. The larger this distance s is, the more similar the data point corresponding to it is, to the data point being considered.

3. Steps 1 & 2 are repeated for all data points. Thus every data point has its own set of distance values **'s'** with respect to all other points. These values are then normalized thus making all **s** values to be on the same scale.
4. Randomly place all data points on a low dimensional surface and repeat steps 1 & 2 to the data points on this lower dimensional surface. Thus on this surface too, each data point will have its own set of **s** values.
5. Using some form of gradient descent, iteratively move the data points in very small steps on the lower dimensional surface such that the s values of the data points in the lower dimensional space are proportional to those in the higher dimensional space. This is done using a suitable loss function (KL divergence - see chapter on multiclass classification).

The distance $\mathbf{s}_j$ corresponding point **j** on a gaussian curve plotted for **point i** could be considered as the measure of the probability that point **j** is a neighbor of **i** (ie: **Pj|i**). This has its corresponding version in the lower dimensional manifold represented by **qj|i** (ie: the probability that a point j on the lower dimensional manifold is a neighbor of **i**). The KL divergence loss is defined using the probabilities just described, as shown below:

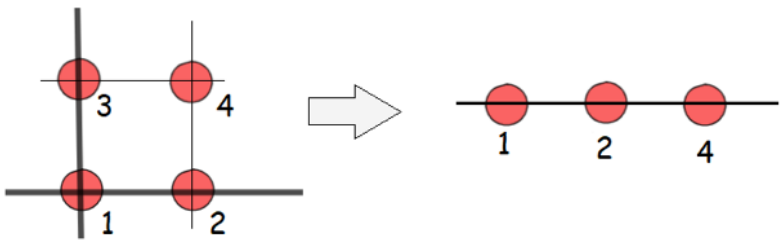
$$L = \sum_i KL(P_i||Q_i) = \sum_i \sum_j p_{j|i} log \frac{p_{j|i}}{q_{j|i}}$$

The KL DIvergence loss is such that, the loss is high if close points in the high dimensional points are mapped as far points on the lower dimensional manifold. But at the same time the penalty is low if far points are mapped as close points. In other words, SNE tries to preserve local structure more than the global structure.

TSNE (REDUCING THE CROWDING PROBLEM):

Consider the data points on the left of the image below. All points are equidistant and hence points 1 and 4 can be considered as neighboring points of 2 & 3.

While embedding these points on a lower dimensional line on the right, only point one of the points 2 or 3 can be represented. In other words all neighborhood distances cannot be preserved. This situation is more likely to occur when the points in the high dimensional space are densely clustered. This is called the crowding problem.



tSNE reduces the crowding problem by replacing the gaussian curve used in the lower dimensional manifold with a **t-distribution**. t-distributions are similar to gaussian distributions except that they have a lower peak and wider tails. This attribute of t-distributions helps even small differences in the distances between the points in the higher dimensional space to be more spread out in the lower dimensional manifold, thus reducing the crowding problem.

tSNE HYPERPARAMETERS:

tSNE basically has two tuning parameters:

1. **Number of iterations:** As discussed earlier, tSNE is optimized using gradient descent techniques and so it follows the following steps:
   a. Stochastically initiate at first the positions of the data points on the lower dimensional manifold.
   b. The loss resulting with respect to these positions of the data points is computed.
   c. The points are then moved by very small amounts based on the loss gradient (KL divergence).
   d. Steps b & c are repeated. At every iteration, the data points are moved by the gradient descent algorithm such that they lessen the KL Divergence. This means that points belonging to the same cluster/neighborhood will be highly "attracted" to one another, while points belonging to different clusters/neighborhood are mildly "repelled" from one another.
   e. After a certain amount of iterations, the embedding on the lower dimensional manifold reaches a stable state, and so the iterations are stopped. Choosing a high number of iterations generally gives better results.

2. **Perplexity:** This parameter may be loosely associated with the spread (variance) of the gaussians used in the lower dimensional embedding process. The wider it is, the more number of neighbors there would be corresponding to each point.

   For fixed number of iterations, the quality of the lower dimensional representation (ie: embedding) improves as the the perplexity increases till it reaches a peak and then it reduces until the perplexity value is the same as the number of data points (All points are considered as neighbors of each other).

cell(i, j) contains the similarity value '$\mathbf{s_{i,j}}$' between data points **i** and **j**) , tSNE basically finds locations on the lower dimensional manifold such that the similarity matrix of these lower dimensional data points are as close as possible to that of the original space.

TSNE on MNIST DATASET:

The **MNIST database** (Modified National Institute of Standards and Technology database) is a database of 70,000 images of handwritten digits (1 to 9) that is commonly used for training various image processing systems. Each image is 28 x 28 pixels and is stored as a flattened 784 dimensional vector.

SAMPLE DATA:

```
df_Xy = pd.read_csv('df_MNIST.csv')

df_Xy.sample(5)
```

	0	1	2	3	4	5	6	7	8	9	...	775	776	777	778	779	780	781	782	783	labels
28475	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	8
551	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2
66195	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4
1475	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	8
35403	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2

LABELS DISTRIBUTION:

```
df_Xy.labels.value_counts()
```

```
1    7877
7    7293
3    7141
2    6990
9    6958
0    6903
6    6876
8    6825
4    6824
5    6313
```

DIMENSIONALITY REDUCTION OF EACH IMAGE TO 15D DENSE VECTORS:

tSNE can be quite time consuming when working with large high dimensional datasets. It is generally a good practice to reduce the dimensionality of the datasets using some parametric dimensionality technique like PCA/SVD before trying to visualize it in 2D using tSNE.

Image vectors have high sparsity and as seen in the previous chapter on SVD, we can easily convert high dimensional sparse matrices to lower dimensional dense matrices (15D) using the code shown below.

```
from sklearn.decomposition import TruncatedSVD

svd_model = TruncatedSVD(n_components = 15).fit(X)
X_svd = svd_model.transform(X)

df_Xy_svd = pd.DataFrame(X_svd).assign(labels = y)

df_Xy_svd.sample(5)
```

	0	1	2	3	.....	9	10	11	12	13	14	labels
37879	1718.427149	-399.658730	252.008904	380.616218		505.864616	-143.515171	-176.512378	447.143992	-235.433046	218.966914	9
43548	1471.954225	-285.119918	-629.953022	-352.793797		-57.739078	-281.304333	-55.817539	457.922913	149.085054	-306.267823	8
29901	966.364026	-699.649920	-583.753002	193.170579		25.411478	-388.566814	-45.875669	178.188380	-279.077206	-213.607960	1
31476	2434.272015	1726.069427	-387.389247	589.087749		-369.678442	-11.599647	383.320002	155.265134	-388.243851	136.385557	0
9877	1960.562777	1311.254213	-144.930065	844.627404		-326.200755	-140.524930	29.372582	242.314226	-315.563577	111.057154	0

APPLYING tSNE:

We apply tSNE to the 15 dimensional reduced data, and get visualizable 2D embedding as shown in the code below:

```
idxs = df_Xy_svd.sample(5000).index

sample_X = df_Xy_svd.iloc[idxs, :-1]
sample_y = df_Xy_svd.iloc[idxs, -1]

kwargs = dict(n_components=2, n_jobs = -2, learning_rate = 300, verbose = 1)
%time X_embedded = TSNE(**kwargs).fit_transform(sample_X)
```

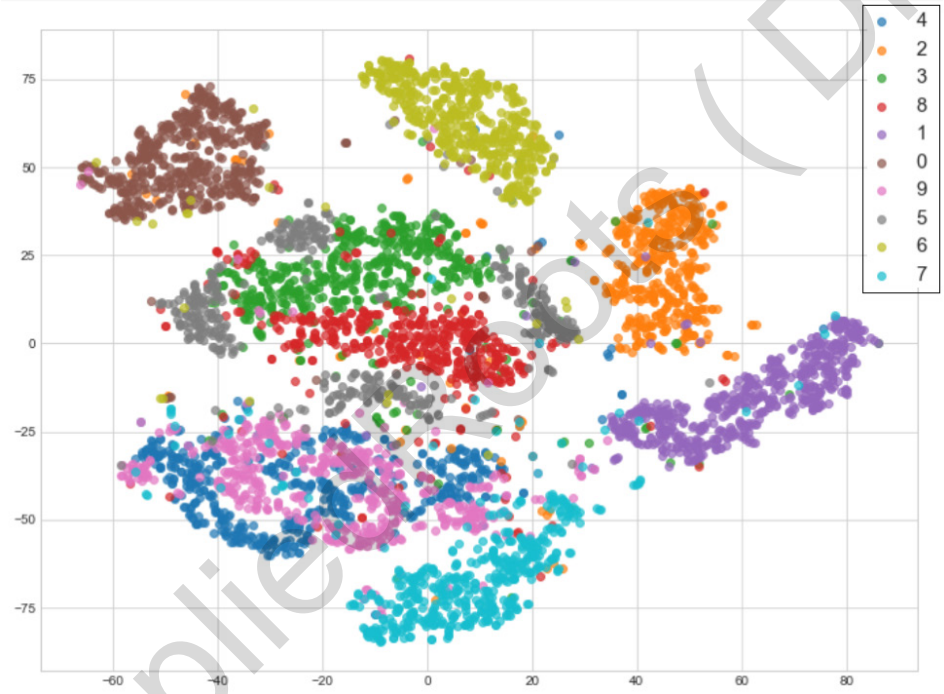
```
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 5000 samples in 0.017s...
[t-SNE] Computed neighbors for 5000 samples in 0.339s...
[t-SNE] Computed conditional probabilities for sample 1000 / 5000
[t-SNE] Computed conditional probabilities for sample 2000 / 5000
[t-SNE] Computed conditional probabilities for sample 3000 / 5000
[t-SNE] Computed conditional probabilities for sample 4000 / 5000
[t-SNE] Computed conditional probabilities for sample 5000 / 5000
[t-SNE] Mean sigma: 267.178867
[t-SNE] KL divergence after 250 iterations with early exaggeration: 81.267014
```

```
df_Xy_tsne = pd.DataFrame(X_embedded).assign(labels = sample_y.values)

df_Xy_tsne.head(5)
```

	0	1	labels
0	7.021192	-33.939091	4
1	-49.283974	-37.952053	4
2	35.899254	17.111765	2
3	-9.511842	6.375481	3
4	-25.324253	-36.320717	4

DATA VISUALIZATION:

```
df_Xy = df_Xy_tsne

fn_plot_clusters(df_Xy)
```



As can be seen from the image above. The tSNE algorithm clearly shows 7 main clusters:

1. Brown: digit = 0
2. Light green: digit = 6
3. Orange: digit = 2
4. Purple: digit = 1
5. Light blue: digit = 7
6. Pink + dark blue: digits = 4 & 9
7. Green + red + grey: digits = 3, 8 & 5

The clusters 6 & 7 form two of the biggest clusters in the middle of the plot above. The sixth cluster contains the digits 4 and 9 grouped together. This is understandable as these digits can be mistaken for each other when handwritten. The same is true for the digits 3, 8 and 5 grouped together in one cluster.

POINTS TO CONSIDER WHILE USING tSNE:

1. tSNE is stochastic in nature and hence its results will have slight variations everytime it is used to embed the same data. Therefore it is prudent to run tSNE multiple times once the parameters are fixed, just to observe the variations.
2. The spread and density of a cluster in the dimensionally reduced visualization does not have any actual "structural" significance. These are just points whose positions are optimized so as to create the best possible representation of the cluster differentiations in 2D. They are not linked to the higher dimensional original space through a mapping function.

   This is unlike PCA/SVD where the dimensionally reduced embedding is a linear algebraic transformation of the original high dimensional data and hence still structurally connected to the original data (a linear algebraic mapping exists between the original data and the transformed data).
3. Expanding point 2 further, even the distances between clusters have no structural significance.
4. To get the best out of tSNE repeat the implementation with various values of perplexity and choose the one that shows best clustering information during visualization.

From points 2 and 3 above, it is clear that tSNE is a dimensionality reduction technique that has only visualization as its goal. On the other hand the dimensionality reductions done using PCA/SVD are done to reduce the computational overheads of using very high dimensional data.

The dimensionally reduced embeddings of PCA/SVD can be considered as lower dimensional structural representations of the original data and hence other statistical/machine learning operations can be performed on it, almost as if it is being performed on the original data. This is what was done in the example above, when we dimensionally reduced the MNIST data to 15 dimensions using SVD. tSNE was applied to this dimensionally reduced data to obtain the final 2D visualization.