

AI · Data Science Diploma

Machine Learning

2019
EDITION

QualityThought®

Reach Us:

Quality Thought Infosystems Pvt. Ltd.
#208B, 2nd Floor, Nilgiri Block, Aditya Enclave
Ameerpet, Hyderabad, Telangana - 38

+ 91 - 9515151992

About Institute?

Quality Thought is a professional Training Organization for software developers, IT administrators, and other professionals. It's Located in Hyderabad, India. The training is offered in Four major modes: Classroom Room Trainings, Online instructor Led Trainings, Self-paced e-learning trainings, and Corporate Trainings.

About Course?

This is an Artificial Intelligence Engineer Master Course that is a comprehensive learning approach for mastering the domains of Artificial Intelligence, Data Science, Business Analytics, Business Intelligence, Python coding, and Deep Learning with TensorFlow. Upon completion of the training, you will be able to take on challenging roles in the artificial intelligence domain.

why we take course?

Artificial intelligence is one of the hottest domains being heralded as the one with the ability to disrupt companies cutting across industry sectors. This Quality Thought Artificial Intelligence Engineer Master Course will equip you with all the necessary skills needed to take on challenging and exciting roles in the artificial intelligence, data science, business analytics, Python, R statistical computing domains and grab the best jobs in the industry at top-notch salaries.

Courses Covered in AI Diploma:

Course 1: As a Future ready IT employee I am interested to learn foundations of Data Analytics with statistics and Tableau, R

Course 2: I want to redefine my Analytical knowledge into python programming for Machine Learning Engineer

Course 3: I want to apply statistics and Python on Machine Learning models for Predictions& classifications of Data in various industry segments for intelligent Business

Course 4: So now I am ready to apply machine Learning models to implement Recommendation systems and Creating Machine Learning service in the cloud(IBM WATSON,AWS)

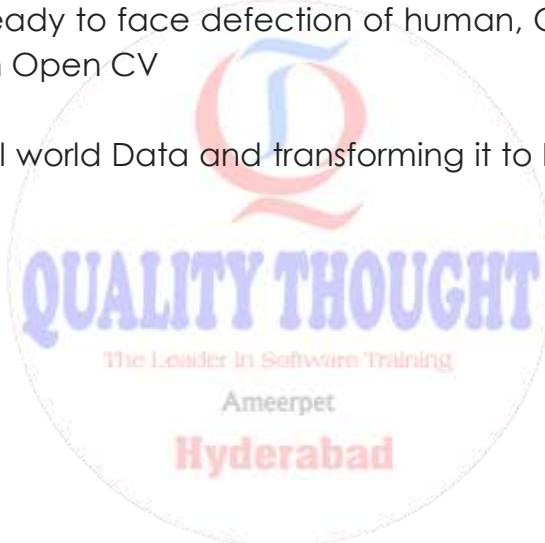
Course 5: Understanding tools of Bigdata and implementation using Machine Learning

Course 6: Applying Machine learning to speech Analytics to process Text using NLP

Course 7: Start Learning Neural Networks using Tensor flows and Keras for image classification and Data Extraction from Image(OCR)

Course 8: Now I am ready to face detection of human, Object using computer vision Technology with Open CV

Course 9: Sensing Real world Data and transforming it to Intelligent actions using IOT



Artificial intelligence vs Machine Learning vs Deep Learning

Nowadays many misconceptions are there related to the words **machine learning**, **deep learning** and **artificial intelligence(AI)**, most of the people think all these things are same whenever they hear the word AI, they directly relate that word to machine learning or vice versa, well yes, these things are related to each other but not the same. Let's see how.

Machine Learning:

Before talking about machine learning lets talk about another concept that is called data mining. Data mining is a technique of examining a large pre-existing database and extracting new information from that database, it's easy to understand, right, machine learning does the same, in fact, machine learning is a type of data mining technique.

Here's is a basic definition of machine learning –

"Machine Learning is a technique of parsing data, learn from that data and then apply what they have learned to make an informed decision"

Now a day's many of big companies use machine learning to give there users a better experience, some of the examples are, Amazon using machine learning to give better product choice recommendations to there costumers based on their preferences, Netflix uses machine learning to give better suggestions to their users of the Tv series or movie or shows that they would like to watch.

Deep Learning:

Deep learning is actually a subset of machine learning. It technically is machine learning and functions in the same way but it has different capabilities.

The main difference between deep and machine learning is, machine learning models become better progressively but the model still needs some guidance. If a machine learning model returns an inaccurate prediction then the programmer needs to fix that problem explicitly but in the case of deep learning, the model does it by himself. Automatic car driving system is a good example of deep learning.

Let's take an example to understand both machine learning and deep learning –

Suppose we have a flashlight and we teach a machine learning model that whenever someone says "dark" the flashlight should be on, now the machine learning model will analyse different phrases said by people and it will search for the word "dark" and as the word comes the flashlight will be on but what if someone said "I am not able to see anything the light is very dim", here the user wants the flashlight to be on but the sentence does not consist the word "dark" so the flashlight will not be on. That's where deep learning is different from machine learning. If it were a deep learning model it would on the flashlight, a deep learning model is able to learn from its own method of computing.

Artificial intelligence:

Now if we talk about AI, it is completely a different thing from Machine learning and deep learning, actually deep learning and machine learning both are the subsets of AI. There is no fixed definition for AI, you will find a different definition everywhere, but here is a definition that will give you idea of what exactly AI is.

"AI is a ability of computer program to function like a human brain "

AI means to actually replicate a human brain, the way a human brain thinks, works and functions. The truth is we are not able to establish a proper AI till now but we are very close to establish it, one of the examples of AI is Sophia, the most advanced AI model present today. The reason we are not able to establish proper AI till now is, we don't know the many aspects of the human brain till now like why do we dream ? etc.

Why people relate machine learning and deep learning with artificial intelligence?

Machine learning and deep learning is a way of achieving AI, which means by the use of machine learning and deep learning we may able to achieve AI in future but it is not AI.

Machine Learning Introduction

Machine Learning is a field which is raised out of Artificial Intelligence(AI). Applying AI, we wanted to build better and intelligent machines. But except for few mere tasks such as finding the shortest path between point A and B, we were unable to program more complex and constantly evolving challenges. There was a realization that the only way to be able to achieve this task was to let machine learn from itself. This sounds similar to a child learning from its self. So machine learning was developed as a new capability for computers. And now machine learning is present in so many segments of technology, that we don't even realize it while using it.

Finding patterns in data on planet earth is possible only for human brains. The data being very massive, the time taken to compute is increased, and this is where Machine Learning comes into action, to help people with large data in minimum time.

If big data and cloud computing are gaining importance for their contributions, machine learning as technology helps analyze those big chunks of data, easing the task of data scientists in an automated process and gaining equal importance and recognition.

The techniques we use for data mining have been around for many years, but they were not effective as they did not have the competitive power to run the algorithms. If you run deep learning with access to better data, the output we get will lead to dramatic breakthroughs which is machine learning.

Kinds of Machine Learning

There are three kinds of Machine Learning Algorithms.

- ✓ Supervised Learning
- ✓ Unsupervised Learning
- ✓ Reinforcement Learning

Supervised Learning

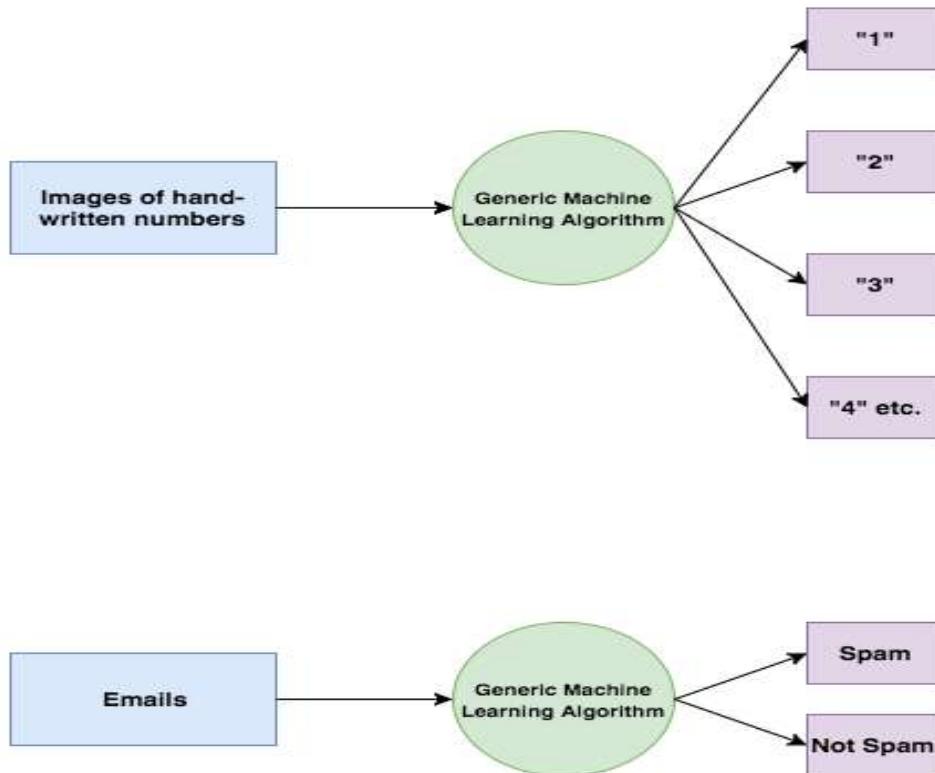
A majority of practical machine learning uses supervised learning.

In supervised learning, the system tries to learn from the previous examples that are given. (On the other hand, in unsupervised learning, the system attempts to find the patterns directly from the example given.)

Speaking mathematically, supervised learning is where you have both input variables (x) and output variables(Y) and can use an algorithm to derive the mapping function from the input to the output.

The mapping function is expressed as $Y = f(X)$.

Example :



Supervised learning problems can be further divided into two parts, namely classification, and regression.

Classification: A classification problem is when the output variable is a category or a group, such as "black" or "white" or "spam" and "no spam".

Regression: A regression problem is when the output variable is a real value, such as "Rupees" or "height."

Unsupervised Learning

In unsupervised learning, the algorithms are left to themselves to discover interesting structures in the data.

Mathematically, unsupervised learning is when you only have input data (X) and no corresponding output variables.

This is called unsupervised learning because unlike supervised learning above, there are no given correct answers and the machine itself finds the answers.

Unsupervised learning problems can be further divided into association and clustering problems.

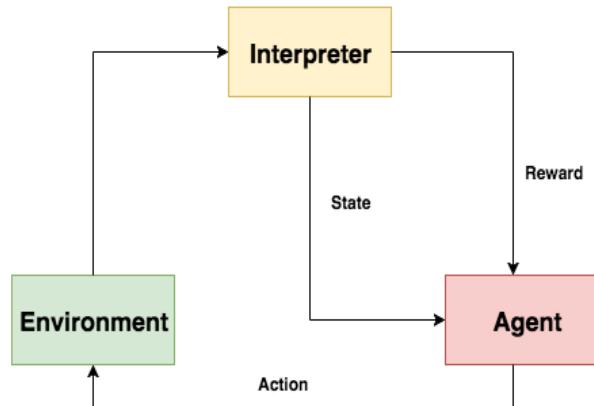
Association: An association rule learning problem is where you want to discover rules that describe large portions of your data, such as “people that buy X also tend to buy Y”.

Clustering: A clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behavior.

Reinforcement Learning

A computer program will interact with a dynamic environment in which it must perform a particular goal (such as playing a game with an opponent or driving a car). The program is provided feedback in terms of rewards and punishments as it navigates its problem space.

Using this algorithm, the machine is trained to make specific decisions. It works this way: the machine is exposed to an environment where it continuously trains itself using trial and error method.



The Math of Intelligence

Machine Learning theory is a field that meets statistical, probabilistic, computer science and algorithmic aspects arising from learning iteratively from data which can be used to build intelligent applications.

9963799240 / 7730997544

Ameerpet / Kondapur

Hyderabad

Why Worry About The Maths?

There are various reasons why the mathematics of Machine Learning is necessary, and I will highlight some of them below:

Selecting the appropriate algorithm for the problem includes considerations of accuracy, training time, model complexity, the number of parameters and number of characteristics.

Identifying underfitting and overfitting by following the Bias-Variance tradeoff.

Choosing parameter settings and validation strategies. Estimating the right determination period and uncertainty.

What Level of Maths Do We Need?

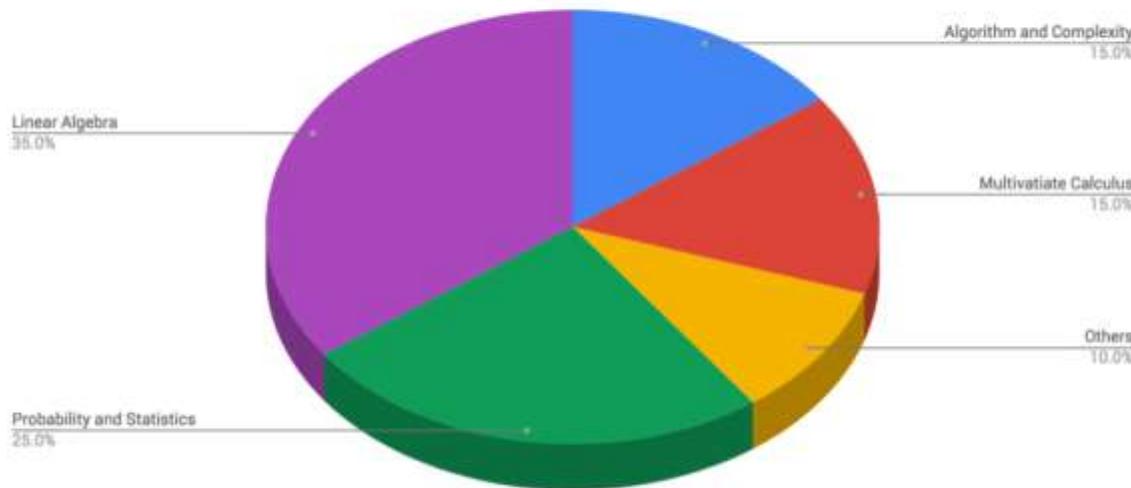
The foremost question when trying to understand a field such as Machine Learning is the amount of maths necessary and the complexity of maths required to understand these systems.

The answer to this question is multidimensional and depends on the level and interest of the individual.

Here is the minimum level of mathematics that is needed for Machine Learning Engineers / Data Scientists.

1. **Linear Algebra** (Matrix Operations, Projections, Factorisation, Symmetric Matrices, Orthogonalisation)
2. **Probability Theory and Statistics** (Probability Rules & Axioms, Bayes' Theorem, Random Variables, Variance and Expectation, Conditional and Joint Distributions, Standard Distributions.)
3. **Calculus** (Differential and Integral Calculus, Partial Derivatives)
4. **Algorithms and Complex Optimisations** (Binary Trees, Hashing, Heap, Stack)

Importance of Mathematic topics needed for Machine Learning



Regression Models

- ✓ In this Machine Learning tutorial, we are going to get the basic understanding of what exactly the Regression Models in Machine Learning.
- ✓ Regression Models are the most popular among all statistical models which are generally used to estimate the relationship between variables.
- ✓ There are different types of Regression Models. Out of which below are the most commonly used models for Regression Analysis.

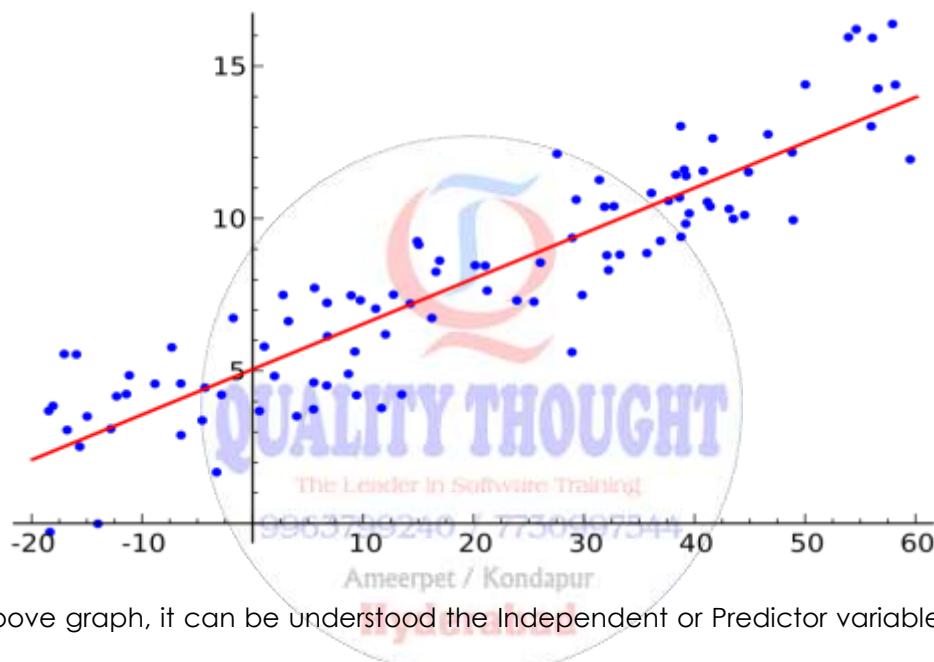
1. Simple Linear Regression
2. Multiple Linear Regression
3. Polynomial Regression
4. Logistic Regression

5. Ridge Regression
6. Lasso Regression
7. ElasticNet Regression
8. Support Vector Regression

These above models will be discussed elaborately in the next upcoming topics in this Machine learning tutorial.

Understanding Regression Analysis

Regression Analysis involves in creating the machine learning models which predict a numeric value. Prediction always happens with a solid machine learning model which estimates the relationship between a dependent variable and Independent variable.



From the above graph, it can be understood the Independent or Predictor variable is on the X-axis,

Whereas the dependent or response variable is on the Y-axis. Coming to the inclined line is none other than the Regression Line. And the Plotted data points can be seen as blue colored dots.

What are Dependent and Independent Variables?

Dependent Variable: By the name itself we can clearly understand that this variable will vary depending on other variables or other factors.

Dependent variable is also called as Response variable (Outcome).

Example:

Consider the students score in the examination which could vary based on several factors.

Independent Variable: This is the variable which is not influenced by other variable rather we can say this variable standalone which can have a quality of influencing others.
Independent variable is also called as Predictor variable (Input).

Example:

Consider the same example as student score in the examination. Generally, student score will depend on various factors like hours of study, attendance etc. So, the time spent by the student to prepare for examination can be considered as independent variable.

Let us continue to learn the Simple Linear Regression....

Simple Linear Regression**Independent and Dependent variables:**

In the context of Statistical learning, there are two types of data:

- ✓ **Independent variables:** Data that can be controlled directly.
- ✓ **Dependent variables:** Data that cannot be controlled directly.

The data that can't be controlled i.e. dependent variables need to predicted or estimated.

Model:

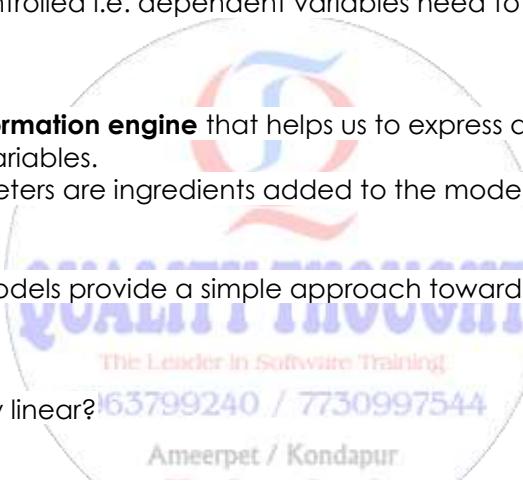
A model is a transformation engine that helps us to express dependent variables as a function of independent variables.

Parameters: Parameters are ingredients added to the model for estimating the output.

Concept

Linear regression models provide a simple approach towards supervised learning. They are simple yet effective.

Wait, what do we mean by linear?



Linear implies the following: arranged in or extending along a straight or nearly straight line. Linear suggests that the relationship between dependent and independent variable can be **expressed in a straight line**.

Recall the geometry lesson from high school. What is the equation of a line?

$$y = mx + c$$

Linear regression is nothing but a manifestation of this simple equation.

- ✓ **y** is the dependent variable i.e. the variable that needs to be estimated and predicted.
- ✓ **x** is the independent variable i.e. the variable that is controllable. It is the input.
- ✓ **m** is the slope. It determines what will be the angle of the line. It is the parameter denoted as β .
- ✓ **c** is the intercept. A constant that determines the value of y when x is 0.

George Box, a famous British statistician, once quoted:
 "All models are wrong; some are useful."

Linear regression models are not perfect. It tries to approximate the relationship between dependent and independent variables in a straight line. Approximation leads to errors. Some errors can be reduced. Some errors are inherent in the nature of the problem. These errors cannot be eliminated. They are called as an **irreducible error**, the noise term in the true relationship that cannot fundamentally be reduced by any model.

The same equation of a line can be re-written as:

$$Y = \beta_0 + \beta_1 X + \epsilon$$

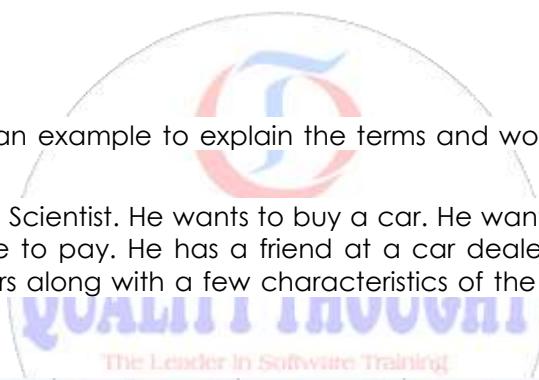
β_0 and β_1 are two unknown constants that represent the intercept and slope. They are the parameters.

ϵ is the error term.

Formulation

Let us go through an example to explain the terms and workings of a Linear regression model.

Fernando is a Data Scientist. He wants to buy a car. He wants to estimate or predict the car price that he will have to pay. He has a friend at a car dealership company. He asks for prices for various other cars along with a few characteristics of the car. His friend provides him with some information.



The Lender in Software Training

make	fuelType	nDoors	engineSize	price
alfa-romero	gas	two	130	13495
alfa-romero	gas	two	130	16500
alfa-romero	gas	two	152	16500
audi	gas	four	109	13950
audi	gas	four	136	17450
audi	gas	two	136	15250
audi	gas	four	136	17710
audi	gas	four	136	18920
audi	gas	four	131	23875

The following are the data provided to him:

make: make of the car.

fuelType: type of fuel used by the car.

nDoor: number of doors.

engineSize: size of the engine of the car.

price: the price of the car.

First, Fernando wants to evaluate if indeed he can predict car price based on engine size. The first set of analysis seeks the answers to the following questions:

- ✓ Is price of car price related with engine size?
- ✓ How strong is the relationship?
- ✓ Is the relationship linear?
- ✓ Can we predict/estimate car price based on engine size?

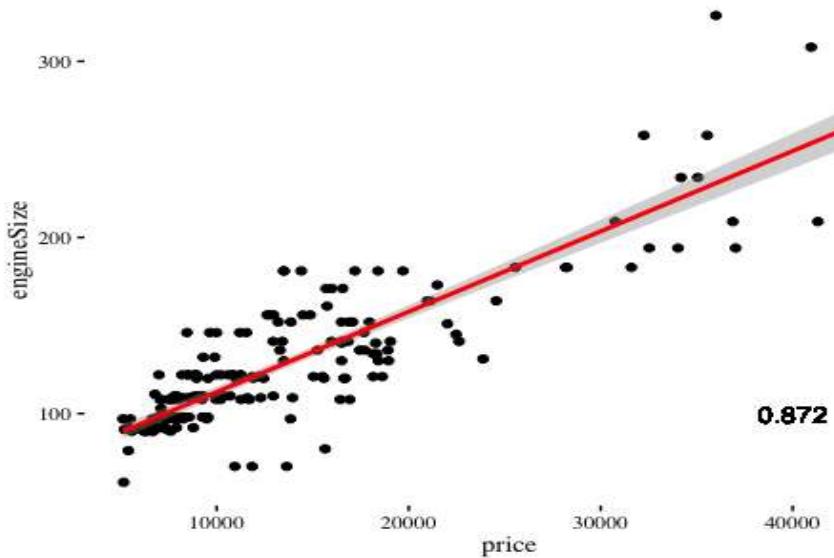
Fernando does a correlation analysis. Correlation is a measure of how much the two variables are related. It is measured by a metric called as the **correlation coefficient**. Its value is between 0 and 1.

If the correlation coefficient is a large(> 0.7) +ve number, it implies that as one variable increases, the other variable increases as well. A large -ve number indicates that as one variable increases, the other variable decreases.

He does a correlation analysis. He plots the relationship between price and engine size.

He splits the data into training and test set. 75% of data is used for training. Remaining is used for the test.

He builds a linear regression model. He uses a statistical package to create the model. The model creates a linear equation that expresses **price of the car** as a function of **engine size**.



Following are the answers to the questions:

- ✓ Is price of car price related with engine size?

- ✓ Yes, there is a relationship.
- ✓ How strong is the relationship?
- ✓ The correlation coefficient is 0.872 => There is a strong relationship.
- ✓ Is the relationship linear?
- ✓ A straight line can fit => A decent prediction of price can be made using engine size.
- ✓ Can we predict/estimate the car price based on engine size?
- ✓ Yes, car price can be estimated based on engine size.

Fernando now wants to build a linear regression model that will estimate the price of the car price based on engine size. Superimposing the equation to the car price problem, Fernando formulates the following equation for price prediction.

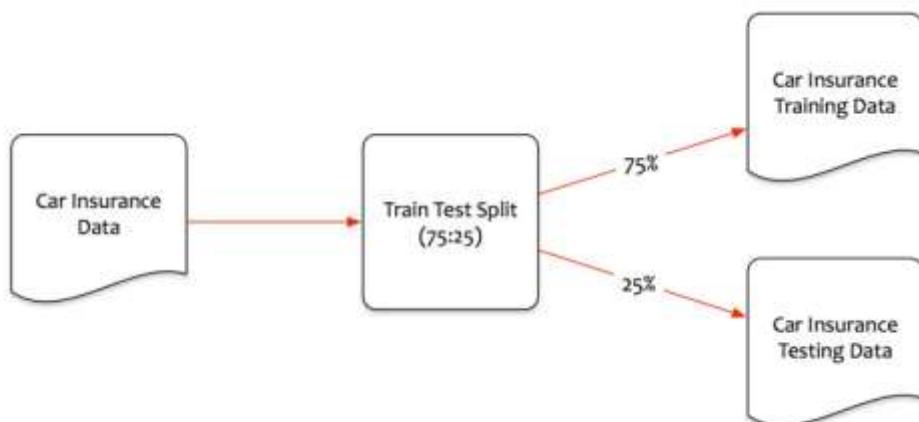
$$\text{price} = \beta_0 + \beta_1 \times \text{engine size}$$

Model Building and Interpretation

Model

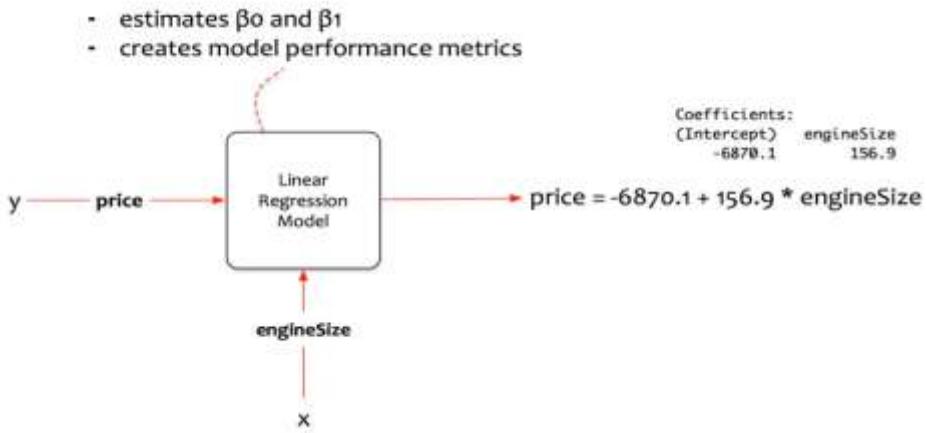
Recall the earlier discussion, on how the data needs to be split into training and testing set. The training data is used to learn about the data. The training data is used to create the model. The testing data is used to evaluate the model performance.

Fernando splits the data into training and test set. 75% of data is used for training. Remaining is used for the test. He builds a linear regression model. He uses a statistical package to create the model. The model produces a linear equation that expresses **price of the car** as a function of **engine size**.



He splits the data into training and test set. 75% of data is used for training. Remaining is used for the test.

He builds a linear regression model. He uses a statistical package to create the model. The model creates a linear equation that expresses **price of the car** as a function of **engine size**.



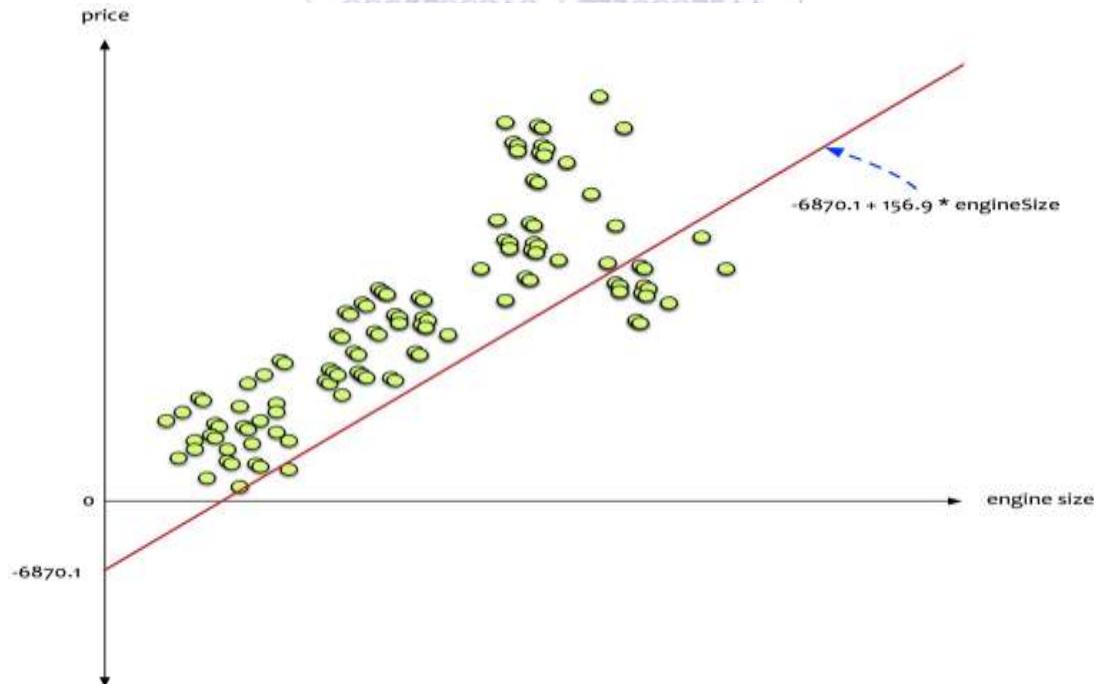
The model estimates the parameters:

- β_0 is estimated as -6870.1
- β_1 is estimated as 156.9

The linear equation is estimated as:

$$\text{price} = -6870.1 + 156.9 \times \text{engine size}$$

Interpretation



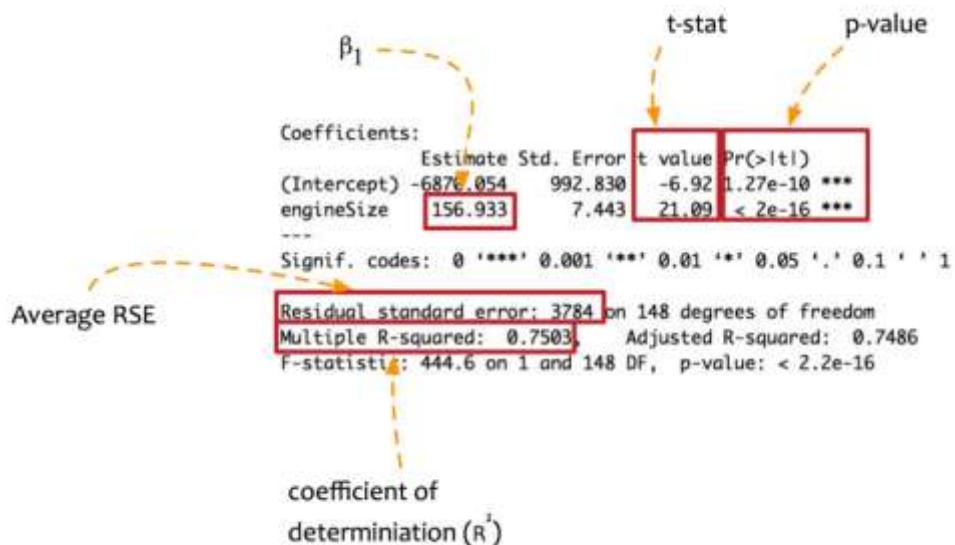
The model provides the equation for the predicting the **average car price** given a specific engine size. This equation means the following:

One unit increase in engine size will increase the average price of the car by 156.9 units.

Evaluation

The model is built. The robustness of the model needs to be evaluated. How can we be sure that the model will be able to predict the price satisfactorily? This evaluation is done in two parts. First, test to establish the robustness of the model. Second, test to evaluate the accuracy of the model.

Fernando first evaluates the model on the training data. He gets the following statistics.



There are a lot of statistics in there. Let us focus on key ones (marked in red). Recall the discussion on hypothesis testing. The robustness of the model is evaluated using hypothesis testing.

H_0 and H_a need to be defined. They are defined as follows:

- H_0 (NULL hypothesis): There is no relationship between x and y i.e. there is no relationship between price and engine size.
- H_a (Alternate hypothesis): There is some relationship between x and y i.e. there is a relationship between price and engine size.

β_1 : The value of β_1 determines the relationship between price and engine size. If $\beta_1 = 0$ then there is no relationship. In this case, β_1 is positive. It implies that there is some relationship between price and engine size.

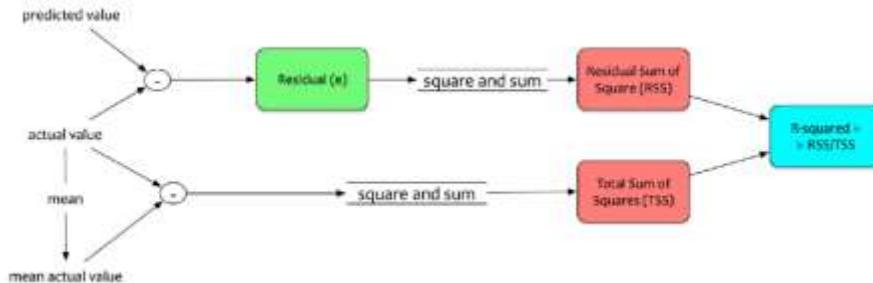
t-stat: The t-stat value is how many standard deviations the coefficient estimate (β_1) is far away from zero. Further, it is away from zero stronger the relationship between price and engine size. The coefficient is significant. In this case, t-stat is 21.09. It is far enough from zero.

p-value: p-value is a probability value. It indicates the chance of seeing the given t-statistics, under the assumption that NULL hypothesis is true. If the p-value is small e.g. < 0.0001 , it implies that the probability that this is by chance and there is no relation is very low. In this case, the p-value is small. It means that relationship between price and engine is not by chance.

With these metrics, we can safely **reject the NULL hypothesis and accept the alternate hypothesis. There is a robust relationship between price and engine size**

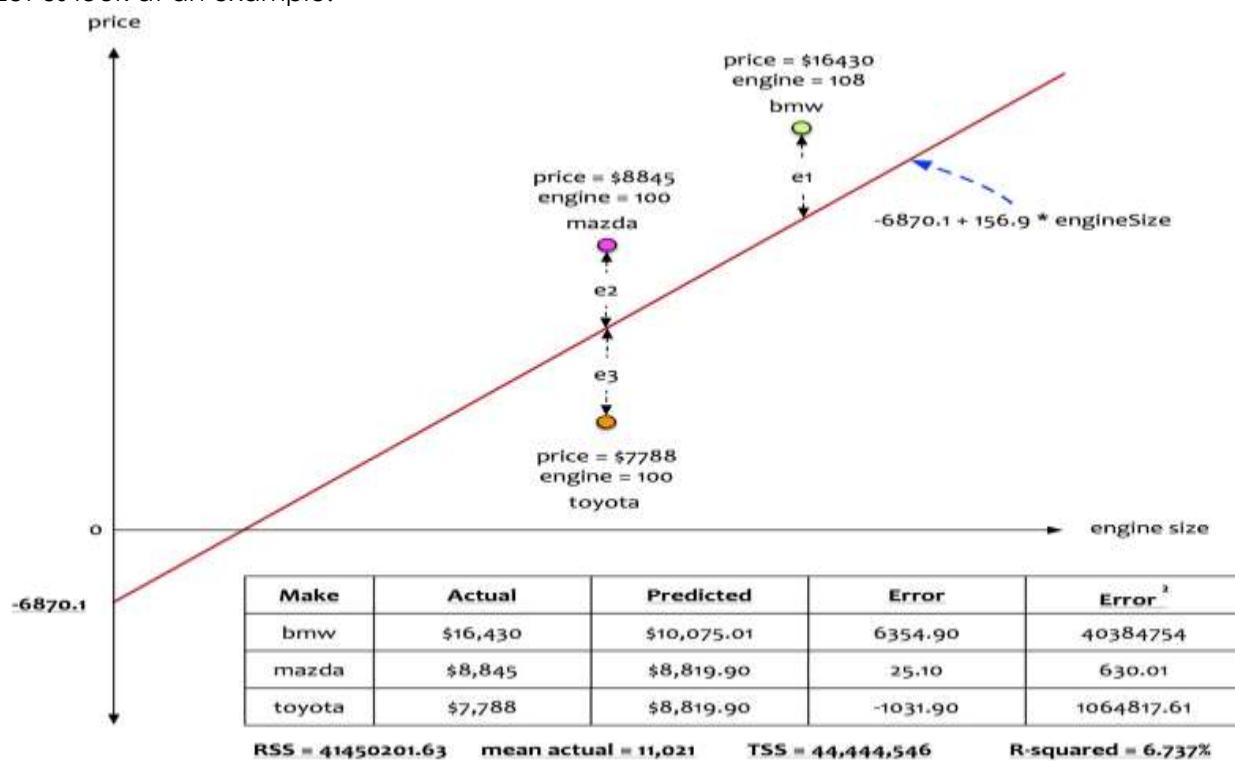
The relationship is established. How about accuracy? How accurate is the model? To get a feel for the **accuracy of the model, a metric named R-squared or coefficient of determination is important.**

R-squared or Coefficient of determination: To understand these metrics, let us break it down into its component.



- **Error (e)** is the difference between the actual y and the predicted y. The predicted y is denoted as \hat{y} . This error is evaluated for each observation. These errors are also called as **residuals**.
- Then all the residual values are squared and added. This term is called as **Residual Sum of Squares (RSS)**. Lower the RSS, the better it is.
- There is another part of the equation of R-squared. To get the other part, first, the mean value of the actual target is computed i.e. average value of the price of the car is estimated. Then the differences between the mean value and actual values are calculated. These differences are then squared and added. It is the **total sum of squares (TSS)**.
- **R-squared** a.k.a coefficient of determination is computed as $1 - \text{RSS}/\text{TSS}$. This metric explains the fraction of the variance between the values predicted by the model and the value as opposed to the mean of the actual. This value is between 0 and 1. The higher it is, the better the model can explain the variance.

Let us look at an example.



In the example above, RSS is computed based on the predicted price for three cars. RSS value is 41450201.63. The mean value of the actual price is 11,021. TSS is calculated as 44,444,546. R-squared is computed as 6.737%. For these three specific data points, the model is only able to explain 6.73% of the variation. Not good enough!! *730997544*

Ameerpet / Kondapur

However, for Fernando's model, it is a different story. The R-squared for the training set is **0.7503** i.e. **75.03%**. It means that the model can explain more 75% of the variation.

Conclusion

Voila!! Fernando has a good model now. It performs satisfactorily on the training data. However, there is 25% of data unexplained. There is room for improvement. How about adding more independent variable for predicting the price? When more than one independent variables are added for predicting a dependent variable, a **multivariate regression model** is created i.e. more than one variable.

Multiple Linear Regression

Multiple linear regression (MLR) is a statistical technique that uses several explanatory variables to predict the outcome of a response variable. The goal of multiple linear regression (MLR) is to model the relationship between the explanatory and response variables.

The model for MLR, given n observations, is:

$$y_i = B_0 + B_1 x_{i1} + B_2 x_{i2} + \dots + B_p x_{ip} + E \text{ where } i = 1, 2, \dots, n$$

BREAKING DOWN Multiple Linear Regression - MLR

A simple linear regression is a function that allows an analyst or statistician to make predictions about one variable based on the information that is known about another variable.

Linear regression can only be used when one has two continuous variables – an independent variable and a dependent variable.

The independent variable is the parameter that is used to calculate the dependent variable or outcome.

For example, an analyst may want to know how the movement of the market affects the price of Exxon Mobil (XOM). In this case, linear equation will have the value of S&P 500 index as the independent variable or predictor, and the price of XOM as the dependent variable.

In reality, there are multiple factors that predict the outcome of an event. The price movement of Exxon Mobil, for example, depends on more than just the performance of the overall market. Other predictors such as the price of oil, interest rates, and the price movement of oil futures can affect the price of XOM and stock prices of other oil companies.

To understand a relationship in which more than two variables are present, a multiple linear regression is used.

Multiple linear regression (MLR) is used to determine a mathematical relationship among a number of random variables. In other terms, MLR examines how multiple independent variables are related to one dependent variable. Once each of the independent factors have been determined to predict the dependent variable, the information on the multiple variables can be used to create an accurate prediction on the level of effect they have on the outcome variable. The model creates a relationship in the form of a straight line (linear) that best approximates all the individual data points.

The model for multiple linear regression is:

$$y_i = B_0 + B_1x_{i1} + B_2x_{i2} + \dots + B_p x_{ip} + E$$

Where y_i = dependent variable - price of XOM

x_{i1} = independent variable – interest rates

x_{i2} = independent variable – oil price

x_{i3} = independent variable – value of S&P 500 index

x_{i4} = independent variable – price of oil futures

E = random error in prediction, that is variance that cannot be accurately predicted by the model. Also known as residuals.

B_0 = y-intercept at time zero.

B_1 = regression coefficient that measures a unit change in the dependent variable when x_{i1} changes – change in XOM price when interest rates change

B_2 = coefficient value that measures a unit change in the dependent variable when x_{i2} changes – change in XOM price when oil prices change

Etc.

The least squares estimates, $B_0, B_1, B_2, \dots, B_p$ are usually computed by statistical software. As many variables can be included in the regression model in which each independent variable is differentiated with a number - 1, 2, 3, 4, ..., p.

The multiple regression model allows an analyst to predict an outcome based on information provided on multiple explanatory variables. Still, the model is not always perfectly accurate as each data point can differ slightly from the outcome predicted by the model.

The residual value, E , which is the difference between the actual outcome and the predicted outcome, is included in the model to account for such slight variations.

The multiple regression model is based on the following assumptions:

- There is a linear relationship between the dependent variables and the independent variables
- The independent variables are not too highly correlated with each other
- y_i observations are selected independently and randomly from the population
- Residuals should be normally distributed with a mean of 0 and variance σ

The co-efficient of determination, R-squared or R^2 , is a statistical metric that is used to measure how much of the variation in outcome can be explained by the variation in the independent variables.

R^2 always increases as more predictors are added to the MLR model even though the predictors may not related to the outcome variable. Therefore, R^2 by itself, cannot be used to identify which predictors should be included in a model and which should be excluded.

R^2 can only be between 0 and 1, where 0 indicates that the outcome cannot be predicted by any of the independent variables and 1 indicates that the outcome can be predicted without error from the independent variables.

Assuming we run our XOM price regression model through a statistics computation software that returns this output:

9963799240 / 7730997544

An analyst would interpret this output to mean if other variables are held constant, the price of XOM will increase by 7.8% if the price of oil in the markets increases by 1%. The model also shows that the price of XOM will decrease by 1.5% following a 1% rise in interest rates. R^2 indicates that 86.5% of the variations in the stock price of Exxon Mobil can be explained by changes in the interest rate, oil price, oil futures, and S&P 500 index.

Polynomial Regression

This function fits a polynomial regression model to powers of a single predictor by the method of linear least squares. Interpolation and calculation of areas under the curve are also given.

If a polynomial model is appropriate for your study then you may use this function to fit a k order/degree polynomial to your data:

- where Y caret is the predicted outcome value for the polynomial model with regression coefficients b_1 to k for each degree and Y intercept b_0 .

The model is simply a general linear regression model with k predictors raised to the power of i where $i=1$ to k .

A second order (k=2) polynomial forms a quadratic expression (parabolic curve).

A third order (k=3) polynomial forms a cubic expression and a fourth order (k=4) polynomial forms a quartic expression.

Some general principles:

- the fitted model is more reliable when it is built on large numbers of observations.
- do not extrapolate beyond the limits of observed values.
- choose values for the predictor (x) that are not too large as they will cause overflow with higher degree polynomials; scale x down if necessary.
- do not draw false confidence from low P values, use these to support your model only if the plot looks reasonable.

More complex expressions involving polynomials of more than one predictor can be achieved by using the general linear regression function.

For more detail from the regression, such as analysis of residuals, use the general linear regression function. To achieve a polynomial fit using general linear regression you must first create new workbook columns that contain the predictor (x) variable raised to powers up to the order of polynomial that you want. For example, a second order fit requires input data of Y, x and x^2 .

Model fit and intervals

Subjective goodness of fit may be assessed by plotting the data and the fitted curve. An analysis of variance is given via the analysis option; this reflects the overall fit of the model. Try to use as few degrees as possible for a model that achieves significance at each degree.

The plot function supplies a basic plot of the fitted curve and a plot with confidence bands and prediction bands. You can save the fitted Y values with their standard errors, confidence intervals and prediction intervals to a workbook.

Area under curve

The option to calculate the area under the fitted curve employs two different methods. The first method integrates the fitted polynomial function from the lowest to the highest observed predictor (x) value using Romberg's integration. The second method uses the trapezoidal rule directly on the data to provide a crude estimate.

Example

Here we use an example from the physical sciences to emphasize the point that polynomial regression is mostly applicable to studies where environments are highly controlled and observations are made to a specified level of tolerance. The data below are the electricity consumptions in kilowatt-hours per month from ten houses and the areas in square feet of those houses:

Home Size	KW Hrs/Mnth
1290	1182
1350	1172
1470	1264
1600	1493
1710	1571

1840	1711
1980	1804
2230	1840
2400	1956
2930	1954

For this example:

Polynomial regression

Intercept	$b_0 = -1216.143887$	$t = -5.008698$	$P = .0016$
Home Size	$b_1 = 2.39893$	$t = 9.75827$	$P < .0001$
Home Size 2	$b_2 = -0.00045$	$t = -7.617907$	$P = .0001$

KW Hrs/Mnth = $-1216.143887 + 2.39893 \text{ Home Size} - 0.00045 \text{ Home Size}^2$

Analysis of variance from regression

Source of variation	Sum Squares	DF	Mean Square
Regression	831069.546371	2	415534.773185
Residual	15332.553629	7	2190.364804
Total (corrected)	846402.1	9	

Root MSE = 46.801333

F = 189.710304 P < .0001

Multiple correlation coefficient	$(R) = 0.990901$
	$R^2 = 98.188502\%$
	$R^2 = 97.670932\%$

Durbin-Watson test statistic = 1.63341

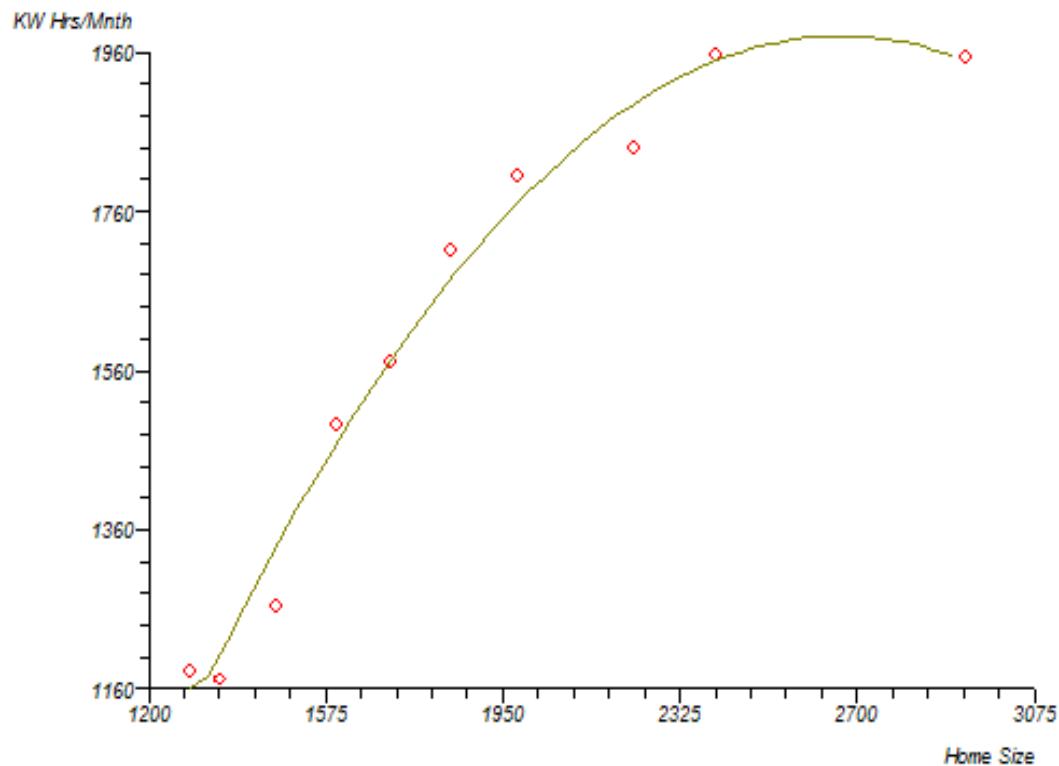
Polynomial regression - area under curve

AUC (polynomial function) = 2855413.374801

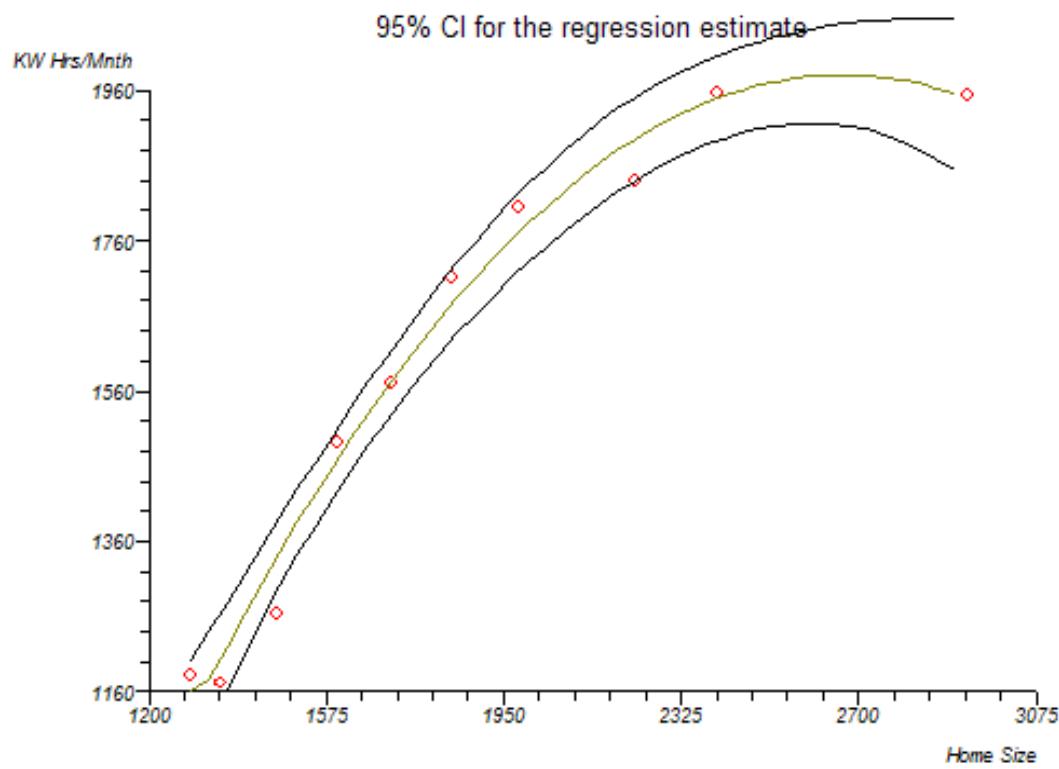
AUC (by trapezoidal rule) = 2838195

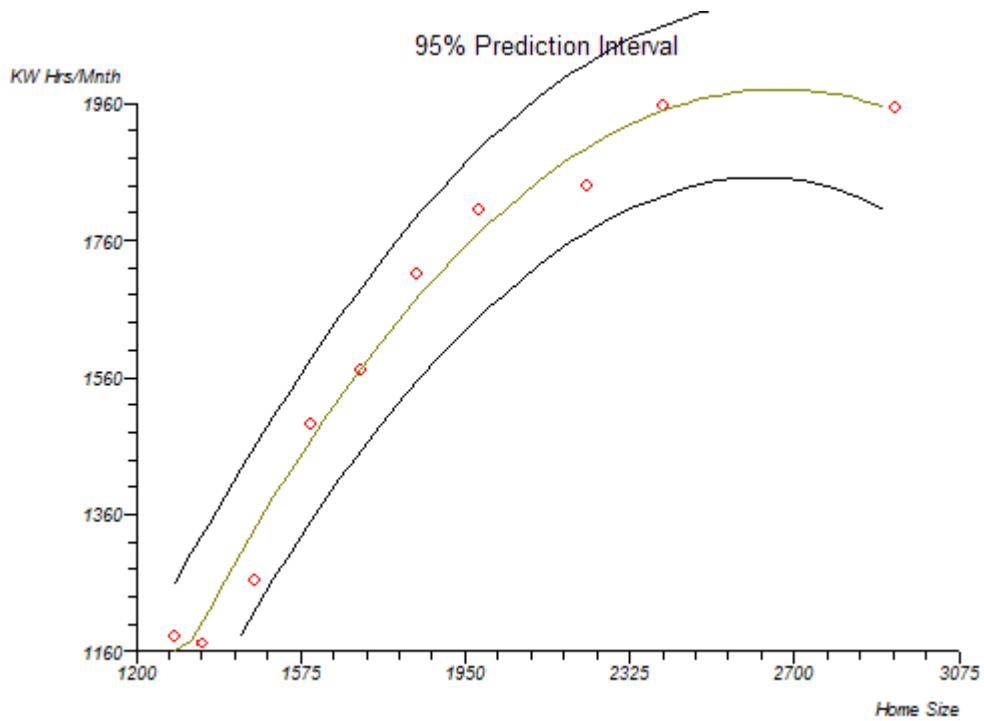
Thus, the overall regression and both degree coefficients are highly significant.

Plots



STATISTICAL MEASURES





In the above curve the right hand end shows a very sharp decline. If you were to extrapolate beyond the data, you have observed then you might conclude that very large houses have very low electricity consumption. This is obviously wrong. Polynomials are frequently illogical for some parts of a fitted curve. You must blend common sense, art and mathematics when fitting these models!

Generalization

Generalization in Machine Learning is a very important element when using machine learning algorithms. For example, the key goal of a machine learning classification algorithm is to create a learning model that accurately predict the class labels of previously unknown data items. Hence the created learning model should work very well in getting future data elements correctly classified. Experts refer to this as generalizing well on 'out of sample data' while the learning algorithm learns from the available 'in sample data'. The term 'in sample data' refers to the data available to us for learning. The term 'out of sample data' is unseen data, unknown data, and in many cases the future data a learning model will face.

The above terms are related to learning theory and the theory of generalization that includes the expectations that the 'out of sample data performance' tracks 'in sample data performance'. This in turn is the first building block of the theory of generalization with the meaning that if we reduce the error of 'in sample data', it is likely that the error of 'out of sample data' will be also reduced and approximately the same. The second building block of generalization theory is then that the learning algorithms will practically reduce the error of 'in sample data' and bring it as close to zero as possible. The latter might lead to a problem called overfitting whereby we memorize data instead of learning from it. A learning model that is overfitting the 'in sample data' is less likely to generalize well on 'out of sample data'.

What Do Machine Learning Algorithms Do?

When we fit a machine learning algorithm, we require a training dataset.

This training dataset includes a set of input patterns and the corresponding output patterns. The goal of the machine learning algorithm is to learn a reasonable approximation of the mapping from input patterns to output patterns.

Here are some examples to make this concrete:

- Mapping from emails to whether they are spam or not for email spam classification.
- Mapping from house details to house sale price for house sale price regression.
- Mapping from photograph to text to describe the photo in photo caption generation.

The list could go on.

We can summarize this mapping that machine learning algorithms learn as a function (f) that predicts the output (y) given the input (X), or restated:

1

$$y = f(X)$$

Our goal in fitting the machine learning algorithms is to get the best possible $f()$ for our purposes.

We are training the model to make predictions in the future given inputs for cases where we do not have the outputs. Where the outputs are unknown. This requires that the algorithm learn in general how to take observations from the domain and make a prediction, not just the specifics of the training data.

This is called generalization.

Generalization is Hard, but Powerful **99240 / 7730997544**

A machine learning algorithm must generalize from training data to the entire domain of all unseen observations in the domain so that it can make accurate predictions when you use the model.

This is really hard.

This approach of generalization requires that the data that we use to train the model (X) is a good and reliable sample of the observations in the mapping we want the algorithm to learn. The higher the quality and the more representative, the easier it will be for the model to learn the unknown and underlying "true" mapping that exists from inputs to outputs.

To generalize means to go from something specific to something broad.

It is the way humans we learn.

- We don't memorize specific roads when we learn to drive; we learn to drive in general so that we can drive on any road or set of conditions.
- We don't memorize specific computer programs when learning to code; we learn general ways to solve problems with code for any business case that might come up.
- We don't memorize the specific word order in natural language; we learn general meanings for words and put them together in new sequences as needed.

The list could go on.

Machine learning algorithms are procedures to automatically generalize from historical observations. And they can generalize on more data than a human could consider, faster than a human could consider it.

It is the speed and scale with which these automated generalization machines operate that is what is so exciting in the field of machine learning.

We Prefer Simpler Models

The machine learning model is the result of the automated generalization procedure called the machine learning algorithm.

The model could be said to be a generalization of the mapping from training inputs to training outputs.

There may be many ways to map inputs to outputs for a specific problem and we can navigate these ways by testing different algorithms, different algorithm configurations, different training data, and so on.

We cannot know which approach will result in the most skillful model beforehand, therefore we must test a suite of approaches, configurations, and framings of the problem to discover what works and what the limits of learning are on the problem before selecting a final model to use.

The skill of the model at making predictions determines the quality of the generalization and can help as a guide during the model selection process.

Out of the millions of possible mappings, we prefer simpler mappings over complex mappings. Put another way, we prefer the simplest possible hypothesis that explains the data.

The simpler model is often (but not always) easier to understand and maintain and is more robust. In practice, you may want to choose the best performing simplest model.

Machine Learning is Not for Every Problem

The ability to automatically learn by generalization is powerful, but is not suitable for all problems.

- Some problems require a precise solution, such as arithmetic on a bank account balance.
- Some problems can be solved by generalization, but simpler solutions exist, such as calculating the square root of positive numbers.
- Some problems look like they could be solved by generalization but there exists no structured underlying relationship to generalize from the data, or such a function is too complex, such as predicting security prices.

Key to the effective use of machine learning is learning where it can and cannot (or should not) be used.

Sometimes this is obvious, but often it is not. Again, you must use experience and experimentation to help tease out whether a problem is a good fit for being solved by generalization.

Regularization:

Regularization is a technique used in an attempt to solve the **overfitting** problem in statistical models.

First of all, I want to clarify how this problem of **overfitting** arises.

When someone wants to model a problem, let's say trying to predict the wage of someone based on his age, he will first try a linear regression model with age as an independent variable and wage as a dependent one.

This model will mostly fail, since it is **too simple**.

Then, you might think: well, I also have the age, the sex and the education of each individual in my data set. I could add these as explaining variables.

Your model becomes **more interesting and more complex**. You measure its accuracy regarding a **loss metric** $L(X, Y)$ where X is your design matrix and Y is the observations (also denoted targets) vector (here the wages).

You find out that your result are quite good but not as perfect as you wish.

So you add more variables: location, profession of parents, social background, number of children, weight, number of books, preferred color, best meal, last holidays destination and so on and so forth.

Your model will do good but it is probably **overfitting**, i.e. it will probably have poor prediction and generalization power: it sticks too much to the data and the model has probably **learned the background noise** while being fit. This isn't of course acceptable.

So how do you solve this?

It is here where the **regularization technique** comes in handy.

You penalize your loss function by adding a multiple of an L1 (LASSO) or an L2 (Ridge) norm of your weights vector w (it is the vector of the learned parameters in your linear regression).

You get the following equation:

$$L(X, Y) + \lambda N(w)$$

(N is either the L1, L2 or any other norm)

This will help you avoid **overfitting** and will perform, at the same time, features selection for certain regularization norms (the L1 in the LASSO does the job).

Finally you might ask: OK I have everything now. How can I tune in the **regularization term** λ ?

One possible answer is to use **cross-validation**: you divide your training data, you train your model for a fixed value of λ and test it on the remaining subsets and repeat this procedure while varying λ . Then you select the best λ that minimizes your loss function.

UnderFitting & OverFitting

Approximate a Target Function in Machine Learning

Supervised machine learning is best understood as approximating a target function (f) that maps input variables (X) to an output variable (Y).

$$Y = f(X)$$

This characterization describes the range of classification and prediction problems and the machine algorithms that can be used to address them.

An important consideration in learning the target function from the training data is how well the model generalizes to new data. Generalization is important because the data we collect is only a sample, it is incomplete and noisy.

Generalization in Machine Learning

In machine learning we describe the learning of the target function from training data as inductive learning.

Induction refers to learning general concepts from specific examples which is exactly the problem that supervised machine learning problems aim to solve. This is different from deduction that is the other way around and seeks to learn specific concepts from general rules.

Generalization refers to how well the concepts learned by a machine learning model apply to specific examples not seen by the model when it was learning.

The goal of a good machine learning model is to generalize well from the training data to any data from the problem domain. This allows us to make predictions in the future on data the model has never seen.

There is a terminology used in machine learning when we talk about how well a machine learning model learns and generalizes to new data, namely overfitting and underfitting.

Overfitting and underfitting are the two biggest causes for poor performance of machine learning algorithms.

Statistical Fit

In statistics, a fit refers to how well you approximate a target function.

This is good terminology to use in machine learning, because supervised machine learning algorithms seek to approximate the unknown underlying mapping function for the output variables given the input variables.

Statistics often describe the goodness of fit which refers to measures used to estimate how well the approximation of the function matches the target function.

Some of these methods are useful in machine learning (e.g. calculating the residual errors), but some of these techniques assume we know the form of the target function we are approximating, which is not the case in machine learning.

If we knew the form of the target function, we would use it directly to make predictions, rather than trying to learn an approximation from samples of noisy training data.

Overfitting in Machine Learning

Overfitting refers to a model that models the training data too well.

Overfitting happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data. This means that the noise or random fluctuations in the training data is picked up and learned as concepts by the model. The problem is that these concepts do not apply to new data and negatively impact the model's ability to generalize.

Overfitting is more likely with nonparametric and nonlinear models that have more flexibility when learning a target function. As such, many nonparametric machine learning algorithms also include parameters or techniques to limit and constrain how much detail the model learns.

For example, decision trees are a nonparametric machine learning algorithm that is very flexible and is subject to overfitting training data. This problem can be addressed by pruning a tree after it has learned in order to remove some of the detail it has picked up.

Underfitting in Machine Learning

Underfitting refers to a model that can neither model the training data nor generalize to new data.

An underfit machine learning model is not a suitable model and will be obvious as it will have poor performance on the training data.

Underfitting is often not discussed as it is easy to detect given a good performance metric. The remedy is to move on and try alternate machine learning algorithms. Nevertheless, it does provide a good contrast to the problem of overfitting.

A Good Fit in Machine Learning

Ideally, you want to select a model at the sweet spot between underfitting and overfitting.

This is the goal, but is very difficult to do in practice.

To understand this goal, we can look at the performance of a machine learning algorithm over time as it is learning a training data. We can plot both the skill on the training data and the skill on a test dataset we have held back from the training process.

Over time, as the algorithm learns, the error for the model on the training data goes down and so does the error on the test dataset. If we train for too long, the performance on the training dataset may continue to decrease because the model is overfitting and learning the irrelevant detail and noise in the training dataset. At the same time the error for the test set starts to rise again as the model's ability to generalize decreases.

The sweet spot is the point just before the error on the test dataset starts to increase where the model has good skill on both the training dataset and the unseen test dataset.

You can perform this experiment with your favorite machine learning algorithms. This is often not useful technique in practice, because by choosing the stopping point for training using the skill on the test dataset it means that the testset is no longer "unseen" or a

standalone objective measure. Some knowledge (a lot of useful knowledge) about that data has leaked into the training procedure.

There are two additional techniques you can use to help find the sweet spot in practice: resampling methods and a validation dataset.

How To Limit Overfitting

Both overfitting and underfitting can lead to poor model performance. But by far the most common problem in applied machine learning is overfitting.

Overfitting is such a problem because the evaluation of machine learning algorithms on training data is different from the evaluation we actually care the most about, namely how well the algorithm performs on unseen data.

There are two important techniques that you can use when evaluating machine learning algorithms to limit overfitting:

1. Use a resampling technique to estimate model accuracy.
2. Hold back a validation dataset.

The most popular resampling technique is k-fold cross validation. It allows you to train and test your model k-times on different subsets of training data and build up an estimate of the performance of a machine learning model on unseen data.

A validation dataset is simply a subset of your training data that you hold back from your machine learning algorithms until the very end of your project. After you have selected and tuned your machine learning algorithms on your training dataset you can evaluate the learned models on the validation dataset to get a final objective idea of how the models might perform on unseen data.

Using cross validation is a gold standard in applied machine learning for estimating model accuracy on unseen data. If you have the data, using a validation dataset is also an excellent practice.

- **Overfitting:** Good performance on the training data, poor generalization to other data.
- **Underfitting:** Poor performance on the training data and poor generalization to other data

Cross Validation

Cross-validation is a statistical method used to estimate the skill of machine learning models.

It is commonly used in applied machine learning to compare and select a model for a given predictive modeling problem because it is easy to understand, easy to implement, and results in skill estimates that generally have a lower bias than other methods.

k-Fold Cross-Validation

Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample.

The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k -fold cross-validation. When a specific value for k is chosen, it may be used in place of k in the reference to the model, such as $k=10$ becoming 10-fold cross-validation.

Cross-validation is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data. That is, to use a limited sample in order to estimate how the model is expected to perform in general when used to make predictions on data not used during the training of the model.

It is a popular method because it is simple to understand and because it generally results in a less biased or less optimistic estimate of the model skill than other methods, such as a simple train/test split.

The general procedure is as follows:

1. Shuffle the dataset randomly.
2. Split the dataset into k groups
3. For each unique group:
 4. Take the group as a hold out or test data set
 5. Take the remaining groups as a training data set
 6. Fit a model on the training set and evaluate it on the test set
 7. Retain the evaluation score and discard the model
 8. Summarize the skill of the model using the sample of model evaluation scores

Importantly, each observation in the data sample is assigned to an individual group and stays in that group for the duration of the procedure. This means that each sample is given the opportunity to be used in the hold out set 1 time and used to train the model $k-1$ times.

Note: This approach involves randomly dividing the set of observations into k groups, or folds, of approximately equal size. The first fold is treated as a validation set, and the method is fit on the remaining $k-1$ folds.

It is also important that any preparation of the data prior to fitting the model occur on the CV-assigned training dataset within the loop rather than on the broader data set. This also applies to any tuning of hyperparameters. A failure to perform these operations within the loop may result in data leakage and an optimistic estimate of the model skill.

Note: Despite the best efforts of statistical methodologists, users frequently invalidate their results by inadvertently peeking at the test data.

The results of a k -fold cross-validation run are often summarized with the mean of the model skill scores. It is also good practice to include a measure of the variance of the skill scores, such as the standard deviation or standard error.

Configuration of k

The k value must be chosen carefully for your data sample.

A poorly chosen value for k may result in a mis-representative idea of the skill of the model, such as a score with a high variance (that may change a lot based on the data used to fit the model), or a high bias, (such as an overestimate of the skill of the model).

Three common tactics for choosing a value for k are as follows:

- **Representative:** The value for k is chosen such that each train/test group of data samples is large enough to be statistically representative of the broader dataset.
- **k=10:** The value for k is fixed to 10, a value that has been found through experimentation to generally result in a model skill estimate with low bias and modest variance.
- **k=n:** The value for k is fixed to n, where n is the size of the dataset to give each test sample an opportunity to be used in the hold out dataset. This approach is called leave-one-out cross-validation.

Note: The choice of k is usually 5 or 10, but there is no formal rule. As k gets larger, the difference in size between the training set and the resampling subsets gets smaller. As this difference decreases, the bias of the technique becomes smaller.

A value of k=10 is very common in the field of applied machine learning, and is recommended if you are struggling to choose a value for your dataset.

Note: To summarize, there is a bias-variance trade-off associated with the choice of k in k-fold cross-validation. Typically, given these considerations, one performs k-fold cross-validation using k = 5 or k = 10, as these values have been shown empirically to yield test error rate estimates that suffer neither from excessively high bias nor from very high variance.

If a value for k is chosen that does not evenly split the data sample, then one group will contain a remainder of the examples. It is preferable to split the data sample into k groups with the same number of samples, such that the sample of model skill scores are all equivalent.

To make the cross-validation procedure concrete, let's look at a worked example. Imagine we have a data sample with 6 observations:

[0.1, 0.2, 0.3, 0.4, 0.5, 0.6]

Ameerpet / Kondapur

The first step is to pick a value for k in order to determine the number of folds used to split the data. Here, we will use a value of k=3. That means we will shuffle the data and then split the data into 3 groups. Because we have 6 observations, each group will have an equal number of 2 observations.

For example:

Fold1: [0.5, 0.2]

Fold2: [0.1, 0.3]

Fold3: [0.4, 0.6]

We can then make use of the sample, such as to evaluate the skill of a machine learning algorithm.

Three models are trained and evaluated with each fold given a chance to be the held out test set.

For example:

- **Model1:** Trained on Fold1 + Fold2, Tested on Fold3
- **Model2:** Trained on Fold2 + Fold3, Tested on Fold1
- **Model3:** Trained on Fold1 + Fold3, Tested on Fold2

The models are then discarded after they are evaluated as they have served their purpose.

The skill scores are collected for each model and summarized for use.

Cross-Validation API

We do not have to implement k-fold cross-validation manually. The scikit-learn library provides an implementation that will split a given data sample up.

The KFold() scikit-learn class can be used. It takes as arguments the number of splits, whether or not to shuffle the sample, and the seed for the pseudorandom number generator used prior to the shuffle.

For example, we can create an instance that splits a dataset into 3 folds, shuffles prior to the split, and uses a value of 1 for the pseudorandom number generator.

kfold = KFold(3, True, 1)

The split() function can then be called on the class where the data sample is provided as an argument. Called repeatedly, the split will return each group of train and test sets. Specifically, arrays are returned containing the indexes into the original data sample of observations to use for train and test sets on each iteration.

For example, we can enumerate the splits of the indices for a data sample using the created KFold instance as follows:

```
# enumerate splits
for train, test in kfold.split(data):
    print('train: %s, test: %s' % (train, test))
```

We can tie all of this together with our small dataset used in the worked example of the prior section.

```
# scikit-learn k-fold cross-validation
from numpy import array
from sklearn.model_selection import KFold
# data sample
data = array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6])
# prepare cross validation
kfold = KFold(3, True, 1)
# enumerate splits
for train, test in kfold.split(data):
    print('train: %s, test: %s' % (data[train], data[test])))
```

Running the example prints the specific observations chosen for each train and test set. The indices are used directly on the original data array to retrieve the observation values.

```
train: [0.1 0.4 0.5 0.6], test: [0.2 0.3]
train: [0.2 0.3 0.4 0.6], test: [0.1 0.5]
train: [0.1 0.2 0.3 0.5], test: [0.4 0.6]
```

Usefully, the k-fold cross validation implementation in scikit-learn is provided as a component operation within broader methods, such as grid-searching model hyperparameters and scoring a model on a dataset.

Nevertheless, the KFold class can be used directly in order to split up a dataset prior to modeling such that all models will use the same data splits. This is especially helpful if you are working with very large data samples. The use of the same splits across algorithms can have benefits for statistical tests that you may wish to perform on the data later.

Variations on Cross-Validation

There are a number of variations on the k-fold cross validation procedure. Three commonly used variations are as follows:

- **Train/Test Split:** Taken to one extreme, k may be set to 1 such that a single train/test split is created to evaluate the model.
- **LOOCV:** Taken to another extreme, k may be set to the total number of observations in the dataset such that each observation is given a chance to be the held out of the dataset. This is called leave-one-out cross-validation, or LOOCV for short.
- **Stratified:** The splitting of data into folds may be governed by criteria such as ensuring that each fold has the same proportion of observations with a given categorical value, such as the class outcome value. This is called stratified cross-validation.
- **Repeated:** This is where the k-fold cross-validation procedure is repeated n times, where importantly, the data sample is shuffled prior to each repetition, which results in a different split of the sample.

Ridge Regression

One of the major aspects of training your machine learning model is avoiding overfitting. The model will have a low accuracy if it is overfitting. This happens because your model is trying too hard to capture the noise in your training dataset. **By noise we mean the data points that don't really represent the true properties of your data, but random chance.** Learning such data points, makes your model more flexible, at the risk of overfitting.

The concept of balancing bias and variance, is helpful in understanding the phenomenon of overfitting.

One of the ways of avoiding overfitting is using cross validation, that helps in estimating the error over test set, and in deciding what parameters work best for your model.

Regularization

This is a form of regression, that constrains/ regularizes or shrinks the coefficient estimates towards zero. In other words, **this technique discourages learning a more complex or flexible model, so as to avoid the risk of overfitting.**

A simple relation for linear regression looks like this. Here Y represents the learned relation and β represents the coefficient estimates for different variables or predictors(X).

$$Y \approx \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p$$

The fitting procedure involves a loss function, known as residual sum of squares or RSS. The coefficients are chosen, such that they minimize this loss function.

$$RSS = \sum_{i=1}^n \left(y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2.$$

Now, this will adjust the coefficients based on your training data. If there is noise in the training data, then the estimated coefficients won't generalize well to the future data. This is where regularization comes in and shrinks or regularizes these learned estimates towards zero.

Ridge Regression

$$\sum_{i=1}^n \left(y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^p \beta_j^2 = RSS + \lambda \sum_{j=1}^p \beta_j^2$$

Above image shows ridge regression, where the **RSS is modified by adding the shrinkage quantity**. Now, the coefficients are estimated by minimizing this function. Here, λ is the tuning parameter that decides how much we want to penalize the flexibility of our model. The increase in flexibility of a model is represented by increase in its coefficients, and if we want to minimize the above function, then these coefficients need to be small. This is how the Ridge regression technique prevents coefficients from rising too high. Also, notice that we shrink the estimated association of each variable with the response, except the intercept β_0 . This intercept is a measure of the mean value of the response when ~~load~~

$$x_{i1} = x_{i2} = \dots = x_{ip} = 0.$$

When $\lambda = 0$, the penalty term has no effect, and the estimates produced by ridge regression will be equal to least squares. However, as $\lambda \rightarrow \infty$, the impact of the shrinkage penalty grows, and the ridge regression coefficient estimates will approach zero. As can be seen, selecting a good value of λ is critical. Cross validation comes in handy for this purpose. The coefficient estimates produced by this method are **also known as the L2 norm**.

The coefficients that are produced by the standard least squares method are scale equivariant, i.e. if we multiply each input by c then the corresponding coefficients are scaled by a factor of $1/c$. Therefore, regardless of how the predictor is scaled, the multiplication of predictor and coefficient($x_j \beta_j$) remains the same. **However, this is not the case with ridge regression, and therefore, we need to standardize the predictors or bring the predictors to the same scale before performing ridge regression.** The formula used to do this is given below.

$$\tilde{x}_{ij} = \frac{x_{ij}}{\sqrt{\frac{1}{n} \sum_{i=1}^n (x_{ij} - \bar{x}_j)^2}},$$

Ridge Regression : In ridge regression, the cost function is altered by adding a penalty equivalent to square of the magnitude of the coefficients.

$$\sum_{i=1}^M (y_i - \hat{y}_i)^2 = \sum_{i=1}^M \left(y_i - \sum_{j=0}^p w_j \times x_{ij} \right)^2 + \lambda \sum_{j=0}^p w_j^2 \quad (1.3)$$

Cost function for ridge regression

This is equivalent to saying minimizing the cost function in equation 1.2 under the condition as below

For some $c > 0$, $\sum_{j=0}^p w_j^2 < c$

Supplement 1: Constrain on Ridge regression coefficients

So ridge regression puts constraint on the coefficients (w). The penalty term (lambda) regularizes the coefficients such that if the coefficients take large values the optimization function is penalized. So, **ridge regression shrinks the coefficients and it helps to reduce the model complexity and multi-collinearity**. Going back to eq. 1.3 one can see that when lambda tends to zero, the cost function becomes similar to the linear regression cost function (eq. 1.2). So lower the constraint (low lambda) on the features, the model will resemble linear regression model. Let's see an example using Boston house data and below is the code I used to depict linear regression as a limiting case of Ridge regression-

```

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import matplotlib
matplotlib.rcParams.update({'font.size': 12})

from sklearn.datasets import load_boston
from sklearn.cross_validation import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge

boston=load_boston()
boston_df=pd.DataFrame(boston.data,columns=boston.feature_names)
#print boston_df.info()
# add another column that contains the house prices which in scikit learn datasets are
# considered as target
boston_df['Price']=boston.target

```

```
#print boston_df.head(3)
newX=boston_df.drop('Price',axis=1)
print newX[0:3] # check
newY=boston_df['Price']
#print type(newY)# pandas core frame
X_train,X_test,y_train,y_test=train_test_split(newX,newY,test_size=0.3,random_state=3)
print len(X_test), len(y_test)
lr = LinearRegression()
lr.fit(X_train, y_train)
rr = Ridge(alpha=0.01) # higher the alpha value, more restriction on the coefficients; low alpha >
more generalization, coefficients are barely
# restricted and in this case linear and ridge regression resembles
rr.fit(X_train, y_train)
rr100 = Ridge(alpha=100) # comparison with alpha value
rr100.fit(X_train, y_train)
train_score=lr.score(X_train, y_train)
test_score=lr.score(X_test, y_test)
Ridge_train_score = rr.score(X_train,y_train)
Ridge_test_score = rr.score(X_test, y_test)
Ridge_train_score100 = rr100.score(X_train,y_train)
Ridge_test_score100 = rr100.score(X_test, y_test)
print "linear regression train score:", train_score
print "linear regression test score:", test_score
print "ridge regression train score low alpha:", Ridge_train_score
print "ridge regression test score low alpha:", Ridge_test_score
print "ridge regression train score high alpha:", Ridge_train_score100
print "ridge regression test score high alpha:", Ridge_test_score100
plt.plot(rr.coef_,alpha=0.7,linestyle='none',marker='*',markersize=5,color='red',label=r'Ridge;
$\alpha = 0.01$',zorder=7) # zorder for ordering the markers
plt.plot(rr100.coef_,alpha=0.5,linestyle='none',marker='d',markersize=6,color='blue',label=r'Ridge;
$\alpha = 100$') # alpha here is for transparency
plt.plot(lr.coef_,alpha=0.4,linestyle='none',marker='o',markersize=7,color='green',label='Linear
Regression')
plt.xlabel('Coefficient Index',fontsize=16)
plt.ylabel('Coefficient Magnitude',fontsize=16)
plt.legend(fontsize=13,loc=4)
plt.show()
```

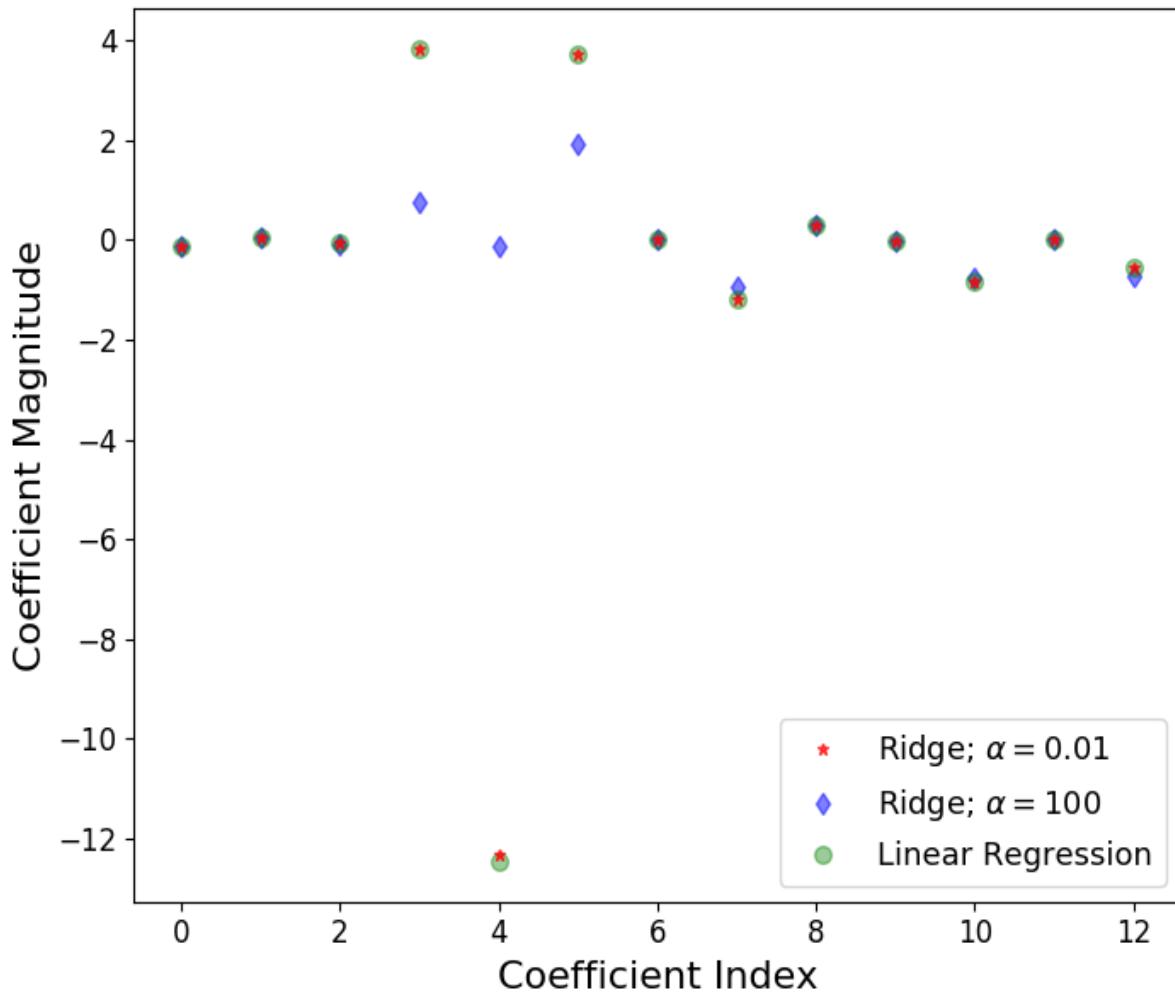


Figure 1: Ridge regression for different values of alpha is plotted to show linear regression as limiting case of ridge regression.

Let's understand the figure above. In X axis we plot the coefficient index and for Boston data there are 13 features (for Python 0th index refers to 1st feature). For low value of alpha (0.01), when the coefficients are less restricted, the coefficient magnitudes are almost same as of linear regression. For higher value of alpha (100), we see that for coefficient indices 3,4,5 the magnitudes are considerably less compared to linear regression case. This is an example of shrinking coefficient magnitude using Ridge regression

Lasso Regression

The cost function for Lasso (least absolute shrinkage and selection operator) regression can be written as

$$\sum_{i=1}^M (y_i - \hat{y}_i)^2 = \sum_{i=1}^M \left(y_i - \sum_{j=0}^p w_j \times x_{ij} \right)^2 + \lambda \sum_{j=0}^p |w_j| \quad (1.4)$$

Cost function for Lasso regression

For some $t > 0$, $\sum_{j=0}^p |w_j| < t$

Supplement 2: Lasso regression coefficients; subject to similar constrain as Ridge, shown before.

Just like Ridge regression cost function, for $\lambda = 0$, the equation above reduces to equation 1.2. The only difference is instead of taking the square of the coefficients, magnitudes are taken into account. This type of regularization (L1) can lead to zero coefficients i.e. some of the features are completely neglected for the evaluation of output. **So Lasso regression not only helps in reducing over-fitting but it can help us in feature selection.** Just like Ridge regression the regularization parameter (λ) can be controlled and we will see the effect below using cancer data set in sklearn. Reason I am using cancer data instead of Boston house data, that I have used before, is, cancer data-set have 30 features compared to only 13 features of Boston house data. So feature selection using Lasso regression can be depicted well by changing the regularization parameter.

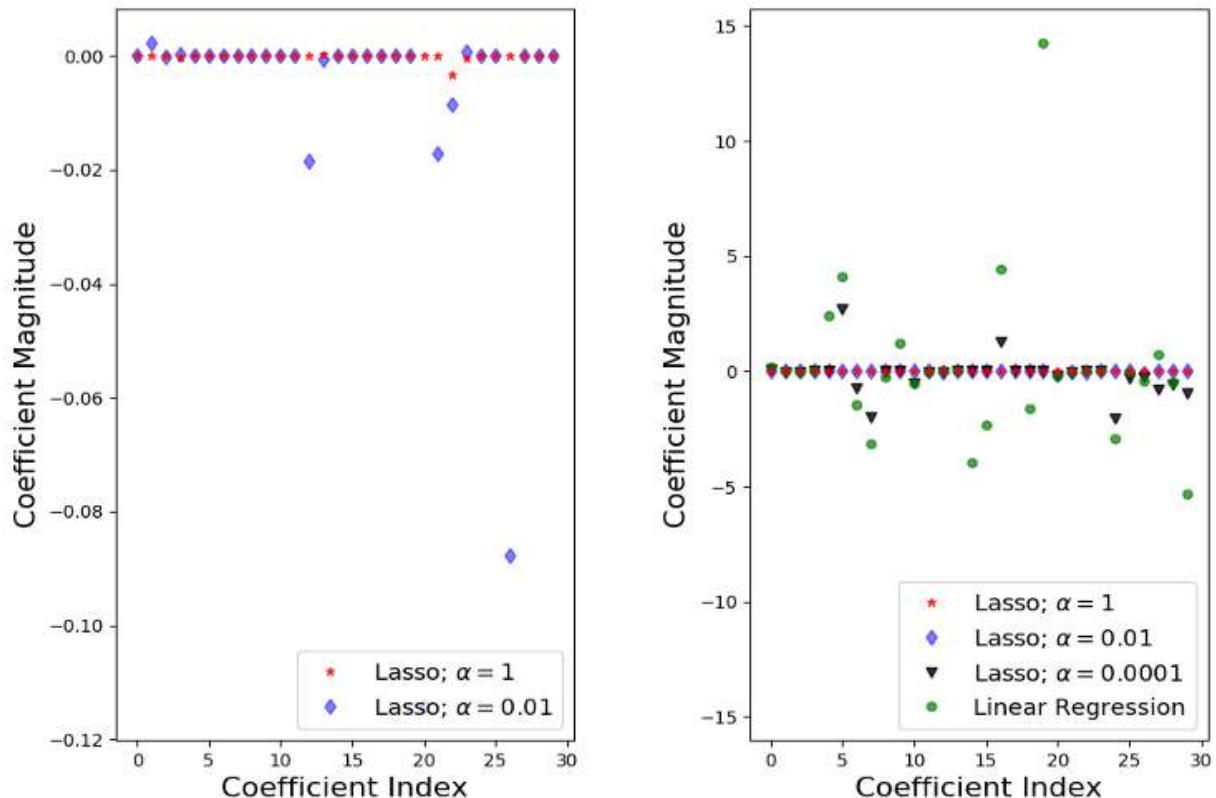


Figure 2: Lasso regression and feature selection dependence on the regularization parameter value.

The code I used to make these plots is as below

```
import math

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np
# difference of lasso and ridge regression is that some of the coefficients can be zero i.e. some
# of the features are
# completely neglected
from sklearn.linear_model import Lasso
from sklearn.linear_model import LinearRegression
from sklearn.datasets import load_breast_cancer
from sklearn.cross_validation import train_test_split
cancer = load_breast_cancer()
#print cancer.keys()
cancer_df = pd.DataFrame(cancer.data, columns=cancer.feature_names)
#print cancer_df.head(3)
X = cancer.data
Y = cancer.target
X_train,X_test,y_train,y_test=train_test_split(X,Y, test_size=0.3, random_state=31)
lasso = Lasso()
lasso.fit(X_train,y_train)
train_score=lasso.score(X_train,y_train)
test_score=lasso.score(X_test,y_test)
coeff_used = np.sum(lasso.coef_!=0)
print "training score:", train_score
print "test score: ", test_score
print "number of features used: ", coeff_used
lasso001 = Lasso(alpha=0.01, max_iter=10e5)
lasso001.fit(X_train,y_train)
train_score001=lasso001.score(X_train,y_train)
test_score001=lasso001.score(X_test,y_test)
coeff_used001 = np.sum(lasso001.coef_!=0)
print "training score for alpha=0.01:", train_score001
print "test score for alpha =0.01: ", test_score001
print "number of features used: for alpha =0.01:", coeff_used001
lasso00001 = Lasso(alpha=0.0001, max_iter=10e5)
lasso00001.fit(X_train,y_train)
train_score00001=lasso00001.score(X_train,y_train)
test_score00001=lasso00001.score(X_test,y_test)
coeff_used00001 = np.sum(lasso00001.coef_!=0)
print "training score for alpha=0.0001:", train_score00001
print "test score for alpha =0.0001: ", test_score00001
print "number of features used: for alpha =0.0001:", coeff_used00001
lr = LinearRegression()
lr.fit(X_train,y_train)
```

```
lr_train_score=lr.score(X_train,y_train)
lr_test_score=lr.score(X_test,y_test)
print "LR training score:", lr_train_score
print "LR test score: ", lr_test_score
plt.subplot(1,2,1)
plt.plot(lasso.coef_,alpha=0.7,linestyle='none',marker='*',markersize=5,color='red',label=r'Lasso;
$\alpha = 1$';zorder=7) # alpha here is for transparency
plt.plot(lasso001.coef_,alpha=0.5,linestyle='none',marker='d',markersize=6,color='blue',label=r'Las
so; $\alpha = 0.01$') # alpha here is for transparency

plt.xlabel('Coefficient Index',fontsize=16)
plt.ylabel('Coefficient Magnitude',fontsize=16)
plt.legend(fontsize=13,loc=4)
plt.subplot(1,2,2)
plt.plot(lasso.coef_,alpha=0.7,linestyle='none',marker='*',markersize=5,color='red',label=r'Lasso;
$\alpha = 1$';zorder=7) # alpha here is for transparency
plt.plot(lasso001.coef_,alpha=0.5,linestyle='none',marker='d',markersize=6,color='blue',label=r'Las
so; $\alpha = 0.01$') # alpha here is for transparency
plt.plot(lasso0001.coef_,alpha=0.8,linestyle='none',marker='v',markersize=6,color='black',label=r'L
asso; $\alpha = 0.00001$') # alpha here is for transparency
plt.plot(lr.coef_,alpha=0.7,linestyle='none',marker='o',markersize=5,color='green',label='Linear
Regression',zorder=2)
plt.xlabel('Coefficient Index',fontsize=16)
plt.ylabel('Coefficient Magnitude',fontsize=16)
plt.legend(fontsize=13,loc=4)
plt.tight_layout()
plt.show()

#output
training score: 0.5600974529893081
test score: 0.5832244618818156
number of features used: 4
training score for alpha=0.01: 0.7037865778498829
test score for alpha =0.01: 0.664183157772623
number of features used: for alpha =0.01: 10
training score for alpha=0.0001: 0.7754092006936697
test score for alpha =0.0001: 0.7318608210757904
number of features used: for alpha =0.0001: 22
LR training score: 0.7842206194055068
LR test score: 0.7329325010888681
```

Let's understand the plot and the code in a short summary.

- The default value of regularization parameter in Lasso regression (given by alpha) is 1.
- With this, out of 30 features in cancer data-set, only 4 features are used (non zero value of the coefficient).
- Both training and test score (with only 4 features) are low; conclude that the model is under-fitting the cancer data-set.

- Reduce this under-fitting by reducing alpha and increasing number of iterations. Now alpha = 0.01, non-zero features =10, training and test score increases.
- Comparison of coefficient magnitude for two different values of alpha are shown in the left panel of figure 2. For alpha =1, we can see most of the coefficients are zero or nearly zero, which is not the case for alpha=0.01.
- Further reduce alpha =0.0001, non-zero features = 22. Training and test scores are similar to basic linear regression case.
- In the right panel of figure, for alpha = 0.0001, coefficients for Lasso regression and linear regression show close resemblance.

So far we have gone through the basics of Ridge and Lasso regression and seen some examples to understand the applications. Now, I will try to explain why the Lasso regression can result in feature selection and Ridge regression only reduces the coefficients close to zero, but not zero. An illustrative figure below will help us to understand better, where we will assume a hypothetical data-set with only two features. Using the constrain for the coefficients of Ridge and Lasso regression (as shown above in the supplements 1 and 2), we can plot the figure below

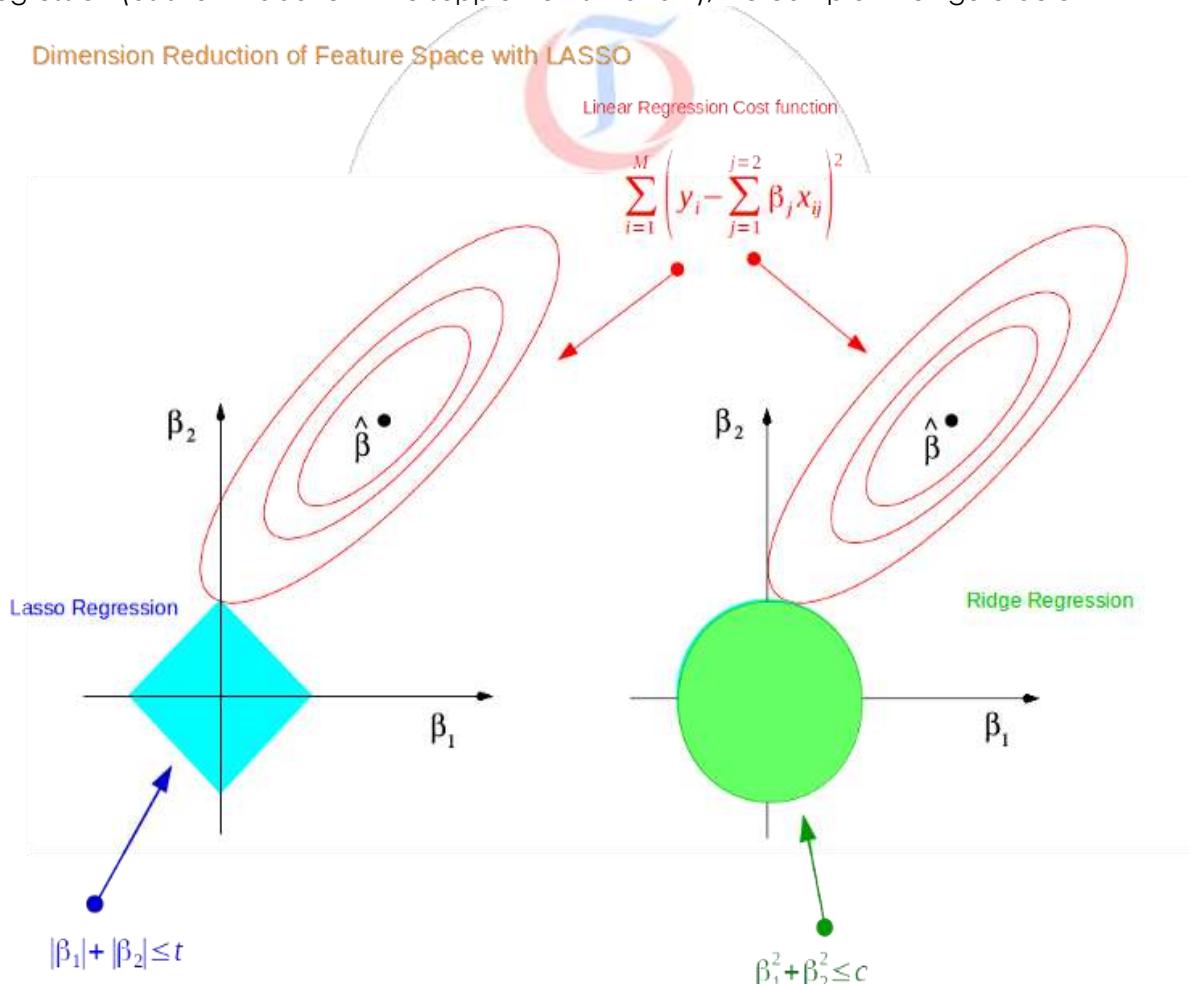


Figure 3: Why LASSO can reduce dimension of feature space? Example on 2D feature space. Modified from the plot used in 'The Elements of Statistical Learning'.

For a two dimensional feature space, the constraint regions (see supplement 1 and 2) are plotted for Lasso and Ridge regression with cyan and green colours. The elliptical contours are the cost function of linear regression (eq. 1.2). Now if we have relaxed conditions on the coefficients, then the constrained regions can get bigger and eventually they will hit the centre of the ellipse. This is the case when Ridge and Lasso regression resembles linear regression results. Otherwise, **both methods determine coefficients by finding the first point where the elliptical contours hit the region of constraints. The diamond (Lasso) has corners on the axes, unlike the disk, and whenever the elliptical region hits such point, one of the features completely vanishes!** For higher dimensional feature space there can be many solutions on the axis with Lasso regression and thus we get only the important features selected.

Finally to end this meditation, let's summarize what we have learnt so far

1. Cost function of Ridge and Lasso regression and importance of regularization term.
2. Went through some examples using simple data-sets to understand Linear regression as a limiting case for both Lasso and Ridge regression.
3. Understood why Lasso regression can lead to feature selection whereas Ridge can only shrink coefficients close to zero.

ElasticNet Regression

Elastic Net produces a regression model that is penalized with both the **L1-norm** and **L2-norm**. The consequence of this is to effectively shrink coefficients (like in ridge regression) and to set some coefficients to zero (as in LASSO).

Loading required R packages

- tidyverse for easy data manipulation and visualization
- caret for easy machine learning workflow
- glmnet, for computing penalized regression

```
library(tidyverse)  
library(caret)  
library(glmnet)
```

Preparing the data

We'll use the Boston data set [in MASS package], introduced in Chapter @ref(regression-analysis), for predicting the median house value (medv), in Boston Suburbs, based on multiple predictor variables.

We'll randomly split the data into training set (80% for building a predictive model) and test set (20% for evaluating the model). Make sure to set seed for reproducibility.

```
# Load the data  
data("Boston", package = "MASS")  
# Split the data into training and test set  
set.seed(123)  
training.samples <- Boston$medv %>%  
  createDataPartition(p = 0.8, list = FALSE)
```

```
train.data <- Boston[training.samples, ]  
test.data <- Boston[-training.samples, ]
```

Computing penalized linear regression

Additional data preparation

You need to create two objects:

- y for storing the outcome variable
- x for holding the predictor variables. This should be created using the function `model.matrix()` allowing to automatically transform any qualitative variables (if any) into dummy variables (Chapter @ref(regression-with-categorical-variables)), which is important because `glmnet()` can only take numerical, quantitative inputs. After creating the model matrix, we remove the intercept component at index = 1.

```
# Predictor variables  
x <- model.matrix(medv ~ ., train.data)[,-1]  
# Outcome variable  
y <- train.data$medv
```

Classification Models

In most of the problems in machine learning however we want to predict whether our output variable belongs to a particular category.

Let us take few examples as below :

1. Amazon wants to know whether a cash on delivery order be accepted by a customer or not when delivered (Yes or No)
2. Bank of America wants to know if a customer will pay back the loan on time , Prepay or not pay at all (OnTime Or Prepaid or Default)
3. Doctors want to know whether a patient will develop Coronary Heart Disease within next 10 years or not (Yes or No)
4. Optical Character Reader wants to read English Characters and find which character is read (A or B or C – Z or 1 or 2 or 3 -9)

All of these are classification problems which fall under the area of Supervised Learning.

Problem 1 and Problem 3 in RED are **Binary classification** problems since we are classifying the output into 2 classes in both the cases as Yes or No.

Problem 2 and Problem 4 in BLUE are **Multi Class Classification** problems since we want to classify output into more than one classes.

Binary Classification

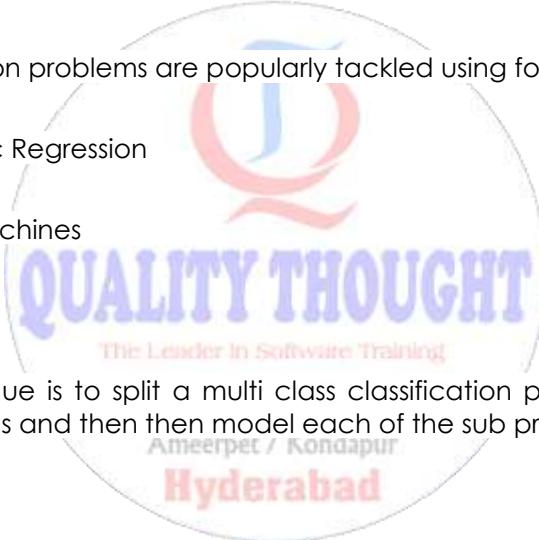
Following techniques are popular for Binary Classification problems.

- a. Logistic Regression
- b. Decision Trees
- c. Random Forests
- d. Neural Networks
- e. Support Vector Machines

Multi Class Classification

Multi class classification problems are popularly tackled using following techniques.

- a. Multinomial Logistic Regression
- b. Support Vector Machines
- c. Neural Networks
- d. A popular technique is to split a multi class classification problem into multiple binary classification problems and then then model each of the sub problem separately.

**Logistic Regression**

Logistic regression is another technique borrowed by machine learning from the field of statistics.

It is the go-to method for binary classification problems (problems with two class values). In this post you will discover the logistic regression algorithm for machine learning.

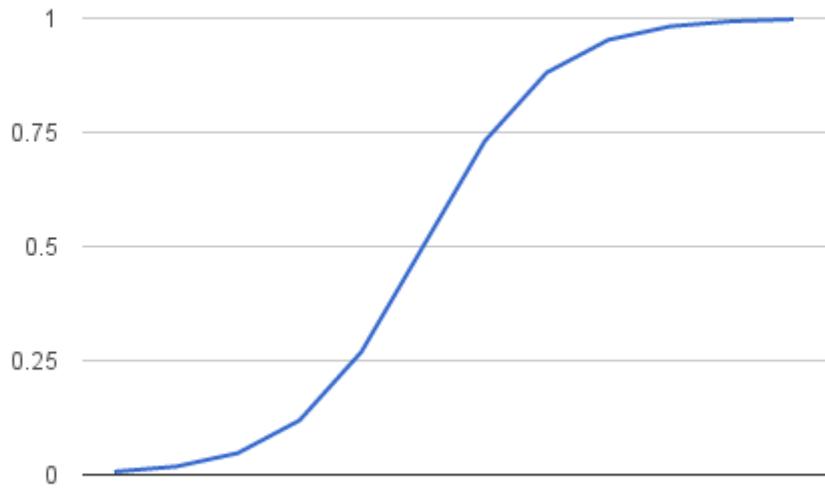
Logistic Function

Logistic regression is named for the function used at the core of the method, the logistic function.

The logistic function, also called the sigmoid function was developed by statisticians to describe properties of population growth in ecology, rising quickly and maxing out at the carrying capacity of the environment. It's an S-shaped curve that can take any real-valued number and map it into a value between 0 and 1, but never exactly at those limits.

$$1 / (1 + e^{-\text{value}})$$

Where e is the base of the natural logarithms (Euler's number or the EXP() function in your spreadsheet) and value is the actual numerical value that you want to transform. Below is a plot of the numbers between -5 and 5 transformed into the range 0 and 1 using the logistic function.



Now that we know what the logistic function is, let's see how it is used in logistic regression.

Representation Used for Logistic Regression

Logistic regression uses an equation as the representation, very much like linear regression.

Input values (x) are combined linearly using weights or coefficient values (referred to as the Greek capital letter Beta) to predict an output value (y). A key difference from linear regression is that the output value being modeled is a binary values (0 or 1) rather than a numeric value.

Below is an example logistic regression equation:

$$y = e^{(b_0 + b_1 \cdot x)} / (1 + e^{(b_0 + b_1 \cdot x)})$$

Where y is the predicted output, b_0 is the bias or intercept term and b_1 is the coefficient for the single input value (x). Each column in your input data has an associated b coefficient (a constant real value) that must be learned from your training data.

The actual representation of the model that you would store in memory or in a file are the coefficients in the equation (the beta value or b 's).

Logistic Regression Predicts Probabilities (Technical Interlude)

Logistic regression models the probability of the default class (e.g. the first class).

For example, if we are modeling people's sex as male or female from their height, then the first class could be male and the logistic regression model could be written as the probability of male given a person's height, or more formally:

$P(\text{sex}=\text{male} | \text{height})$

Written another way, we are modeling the probability that an input (X) belongs to the default class (Y=1), we can write this formally as:

$P(X) = P(Y=1 | X)$

We're predicting probabilities? I thought logistic regression was a classification algorithm?

Note that the probability prediction must be transformed into a binary values (0 or 1) in order to actually make a probability prediction. More on this later when we talk about making predictions.

Logistic regression is a linear method, but the predictions are transformed using the logistic function. The impact of this is that we can no longer understand the predictions as a linear combination of the inputs as we can with linear regression, for example, continuing on from above, the model can be stated as:

$$p(X) = e^{(b_0 + b_1 * X)} / (1 + e^{(b_0 + b_1 * X)})$$

I don't want to dive into the math too much, but we can turn around the above equation as follows (remember we can remove the e from one side by adding a natural logarithm (ln) to the other):

$$\ln(p(X) / 1 - p(X)) = b_0 + b_1 * X$$

This is useful because we can see that the calculation of the output on the right is linear again (just like linear regression), and the input on the left is a log of the probability of the default class.

This ratio on the left is called the odds of the default class (it's historical that we use odds, for example, odds are used in horse racing rather than probabilities). Odds are calculated as a ratio of the probability of the event divided by the probability of not the event, e.g. $0.8/(1-0.8)$ which has the odds of 4. So we could instead write:

$$\ln(\text{odds}) = b_0 + b_1 * X$$

Because the odds are log transformed, we call this left hand side the log-odds or the probit. It is possible to use other types of functions for the transform (which is out of scope, but as such it is common to refer to the transform that relates the linear regression equation to the probabilities as the link function, e.g. the probit link function).

We can move the exponent back to the right and write it as:

$$\text{odds} = e^{(b_0 + b_1 * X)}$$

All of this helps us understand that indeed the model is still a linear combination of the inputs, but that this linear combination relates to the log-odds of the default class.

Learning the Logistic Regression Model

The coefficients (Beta values b) of the logistic regression algorithm must be estimated from your training data. This is done using maximum-likelihood estimation.

Maximum-likelihood estimation is a common learning algorithm used by a variety of machine learning algorithms, although it does make assumptions about the distribution of your data (more on this when we talk about preparing your data).

The best coefficients would result in a model that would predict a value very close to 1 (e.g. male) for the default class and a value very close to 0 (e.g. female) for the other class. The intuition for maximum-likelihood for logistic regression is that a search procedure seeks values for the coefficients (Beta values) that minimize the error in the probabilities predicted by the model to those in the data (e.g. probability of 1 if the data is the primary class).

We are not going to go into the math of maximum likelihood. It is enough to say that a minimization algorithm is used to optimize the best values for the coefficients for your training data. This is often implemented in practice using efficient numerical optimization algorithm (like the Quasi-newton method).

When you are learning logistic, you can implement it yourself from scratch using the much simpler gradient descent algorithm.

Making Predictions with Logistic Regression

Making predictions with a logistic regression model is as simple as plugging in numbers into the logistic regression equation and calculating a result.

Let's make this concrete with a specific example.

Let's say we have a model that can predict whether a person is male or female based on their height (completely fictitious). Given a height of 150cm is the person male or female.

We have learned the coefficients of $b_0 = -100$ and $b_1 = 0.6$. Using the equation above we can calculate the probability of male given a height of 150cm or more formally $P(\text{male} | \text{height}=150)$. We will use `EXP()` for e, because that is what you can use if you type this example into your spreadsheet:

$$\begin{aligned}y &= e^{(b_0 + b_1 * X)} / (1 + e^{(b_0 + b_1 * X)}) \\y &= \exp(-100 + 0.6 * 150) / (1 + \exp(-100 + 0.6 * 150)) \\y &= 0.0000453978687\end{aligned}$$

Or a probability of near zero that the person is a male.

In practice we can use the probabilities directly. Because this is classification and we want a crisp answer, we can snap the probabilities to a binary class value, for example:

0 if $p(\text{male}) < 0.5$
1 if $p(\text{male}) \geq 0.5$

Now that we know how to make predictions using logistic regression, let's look at how we can prepare our data to get the most from the technique.

Prepare Data for Logistic Regression

The assumptions made by logistic regression about the distribution and relationships in your data are much the same as the assumptions made in linear regression.

Much study has gone into defining these assumptions and precise probabilistic and statistical language is used. My advice is to use these as guidelines or rules of thumb and experiment with different data preparation schemes.

Ultimately in predictive modeling machine learning projects you are laser focused on making accurate predictions rather than interpreting the results. As such, you can break some assumptions as long as the model is robust and performs well.

- **Binary Output Variable:** This might be obvious as we have already mentioned it, but logistic regression is intended for binary (two-class) classification problems. It will predict the probability of an instance belonging to the default class, which can be snapped into a 0 or 1 classification.
- **Remove Noise:** Logistic regression assumes no error in the output variable (y), consider removing outliers and possibly misclassified instances from your training data.
- **Gaussian Distribution:** Logistic regression is a linear algorithm (with a non-linear transform on output). It does assume a linear relationship between the input variables with the output. Data transforms of your input variables that better expose this linear relationship can result in a more accurate model. For example, you can use log, root, Box-Cox and other univariate transforms to better expose this relationship.
- **Remove Correlated Inputs:** Like linear regression, the model can overfit if you have multiple highly-correlated inputs. Consider calculating the pairwise correlations between all inputs and removing highly correlated inputs.
- **Fail to Converge:** It is possible for the expected likelihood estimation process that learns the coefficients to fail to converge. This can happen if there are many highly correlated inputs in your data or the data is very sparse (e.g. lots of zeros in your input data).

Knn Algorithm

K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions). KNN has been used in statistical estimation and pattern recognition already in the beginning of 1970's as a non-parametric technique.

Algorithm

A case is classified by a majority vote of its neighbors, with the case being assigned to the class most common amongst its K nearest neighbors measured by a distance function. If K = 1, then the case is simply assigned to the class of its nearest neighbor.

Distance functions

Euclidean

$$\sqrt{\sum_{i=1}^k (x_i - y_i)^2}$$

Manhattan

$$\sum_{i=1}^k |x_i - y_i|$$

Minkowski

$$\left(\sum_{i=1}^k (|x_i - y_i|)^q \right)^{1/q}$$

It should also be noted that all three distance measures are only valid for continuous variables. In the instance of categorical variables the Hamming distance must be used. It also brings up the issue of standardization of the numerical variables between 0 and 1 when there is a mixture of numerical and categorical variables in the dataset.

Hamming Distance

$$D_H = \sum_{i=1}^k |x_i - y_i|$$

$$x = y \Rightarrow D = 0$$

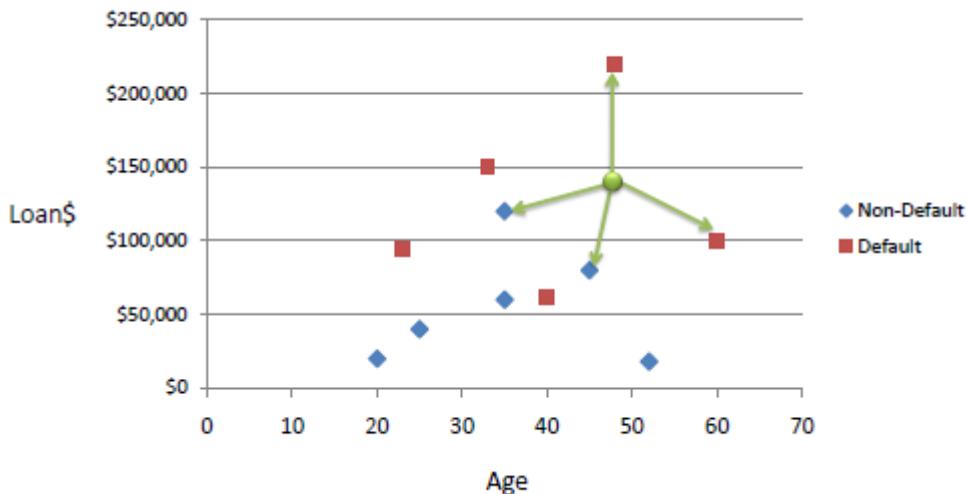
$$x \neq y \Rightarrow D = 1$$

X	Y	Distance
Male	Male	0
Male	Female	1

Choosing the optimal value for K is best done by first inspecting the data. In general, a large K value is more precise as it reduces the overall noise but there is no guarantee. Cross-validation is another way to retrospectively determine a good K value by using an independent dataset to validate the K value. Historically, the optimal K for most datasets has been between 3-10. That produces much better results than 1NN.

Example:

Consider the following data concerning credit default. Age and Loan are two numerical variables (predictors) and Default is the target.



We can now use the training set to classify an unknown case (Age=48 and Loan=\$142,000) using Euclidean distance. If K=1 then the nearest neighbor is the last case in the training set with Default=Y.

$$D = \text{Sqrt}[(48-33)^2 + (142000-150000)^2] = 8000.01 \gg \text{Default=Y}$$

Age	Loan	Default	Distance
25	\$40,000	N	102000
35	\$60,000	N	82000
45	\$80,000	N	62000
20	\$20,000	N	122000
35	\$120,000	N	22000
52	\$18,000	N	124000
23	\$95,000	Y	47000
40	\$62,000	Y	80000
60	\$100,000	Y	42000
48	\$220,000	Y	78000
33	\$150,000	Y	8000
48	\$142,000	?	

Euclidean Distance

$$D = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

2

3

1

With K=3, there are two Default=Y and one Default=N out of three closest neighbors. The prediction for the unknown case is again Default=Y.

Standardized Distance

One major drawback in calculating distance measures directly from the training set is in

the case where variables have different measurement scales or there is a mixture of numerical and categorical variables. For example, if one variable is based on annual income in dollars, and the other is based on age in years then income will have a much higher influence on the distance calculated. One solution is to standardize the training set as shown below.

Age	Loan	Default	Distance
0.125	0.11	N	0.7652
0.375	0.21	N	0.5200
0.625	0.31	N	0.3160
0	0.01	N	0.9245
0.375	0.50	N	0.3428
0.8	0.00	N	0.6220
0.075	0.38	Y	0.6669
0.5	0.22	Y	0.4437
1	0.41	Y	0.3650
0.7	1.00	Y	0.3861
0.325	0.65	Y	0.3771
0.7	0.61	?	

Standardized Variable

$$X_s = \frac{X - \text{Min}}{\text{Max} - \text{Min}}$$

Using the standardized distance on the same training set, the unknown case returned a different neighbor which is not a good sign of robustness.

When do we use KNN algorithm? 963799240 / 7730997544

KNN can be used for both classification and regression predictive problems. However, it is more widely used in classification problems in the industry. To evaluate any technique we generally look at 3 important aspects:

1. Ease to interpret output
2. Calculation time
3. Predictive Power

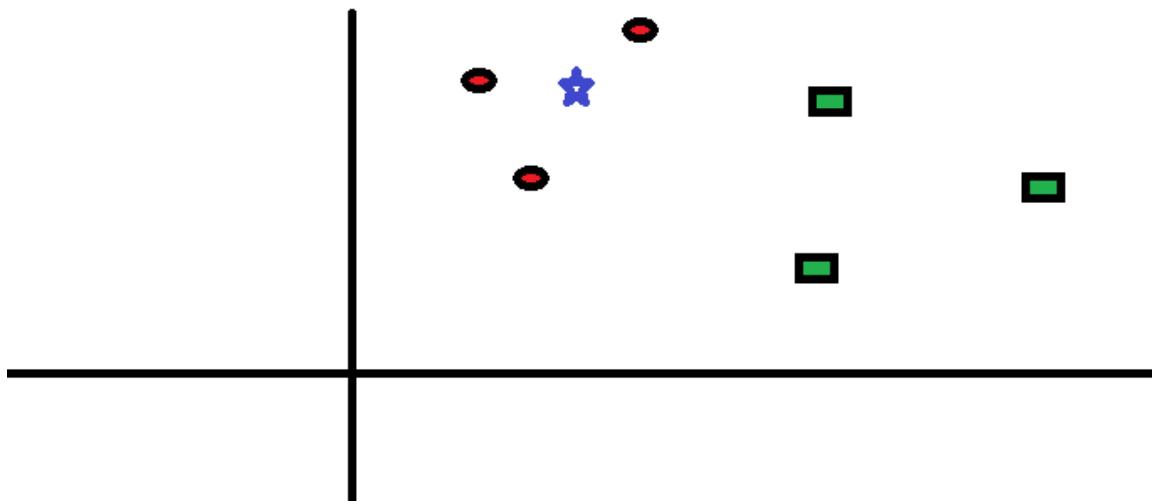
Let us take a few examples to place KNN in the scale :

	Logistic Regression	CART	Random Forest	KNN
1. Ease to interpret output	2	3	1	3
2. Calculation time	3	2	1	3
3. Predictive Power	2	2	3	2

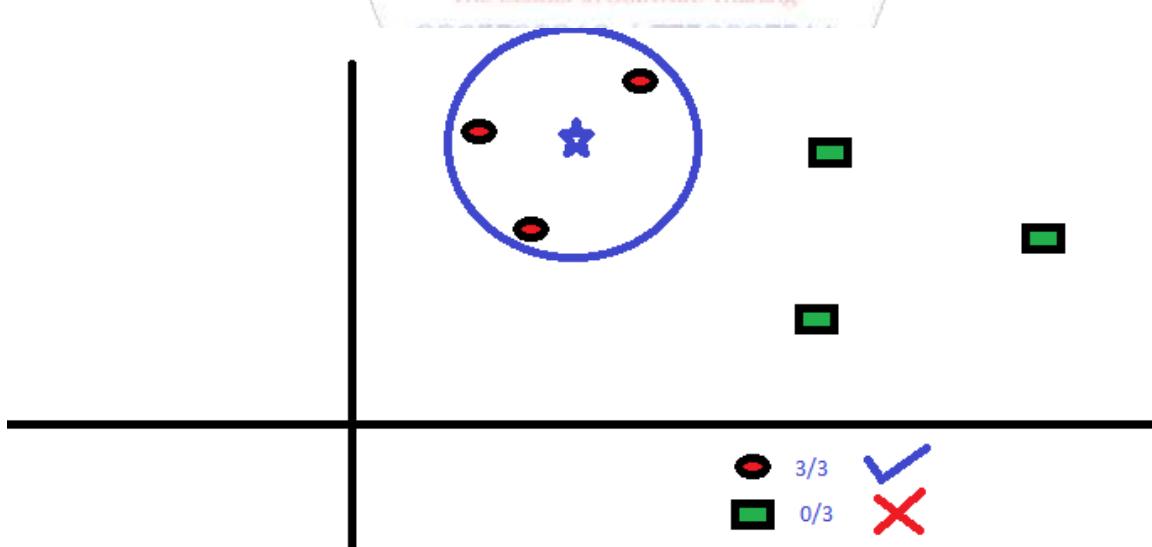
KNN algorithm fairs across all parameters of considerations. It is commonly used for its easy of interpretation and low calculation time.

How does the KNN algorithm work?

Let's take a simple case to understand this algorithm. Following is a spread of red circles (RC) and green squares (GS) :



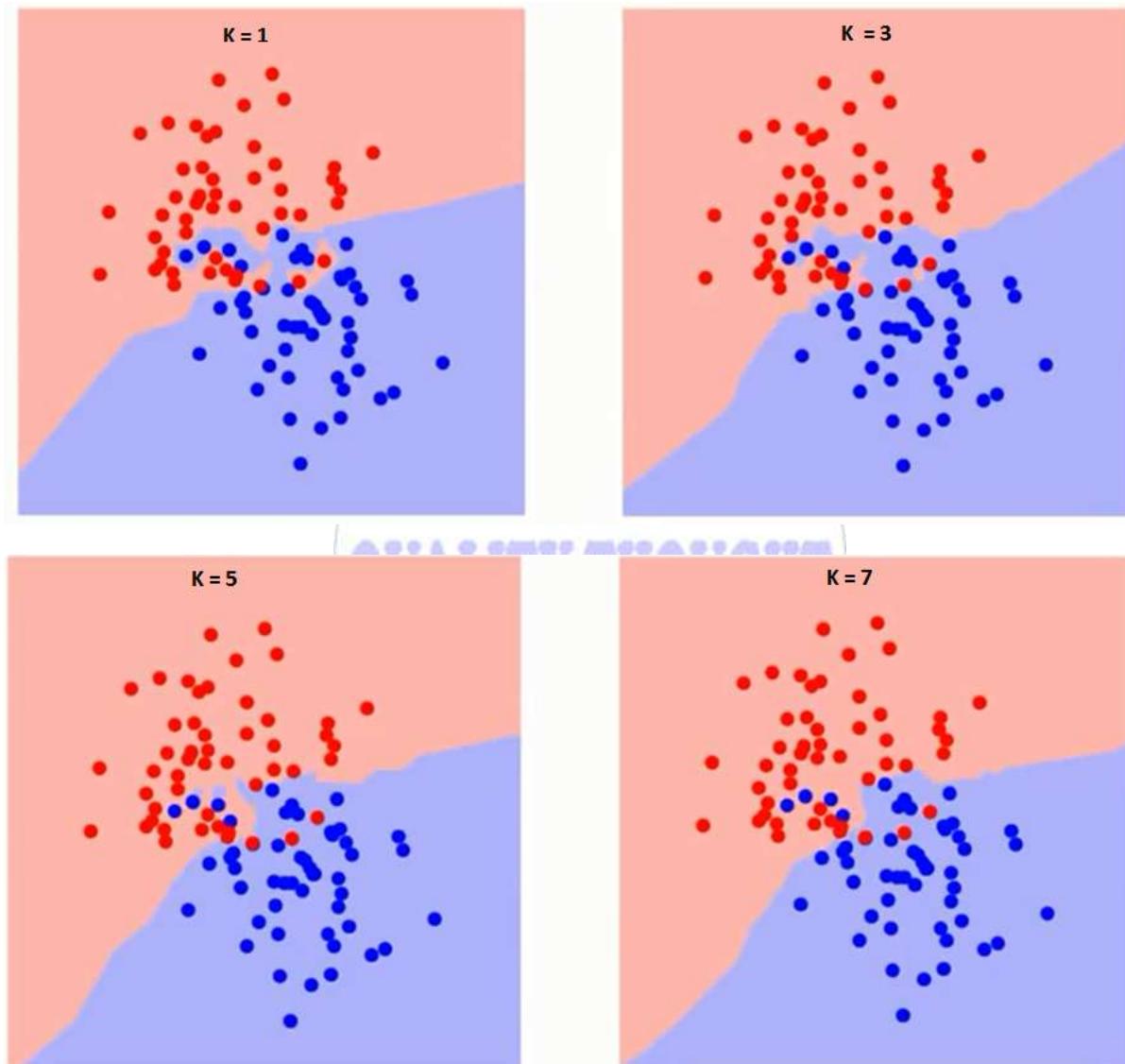
You intend to find out the class of the blue star (BS) . BS can either be RC or GS and nothing else. The "K" in KNN algorithm is the nearest neighbors we wish to take vote from. Let's say K = 3. Hence, we will now make a circle with BS as center just as big as to enclose only three datapoints on the plane. Refer to following diagram for more details:



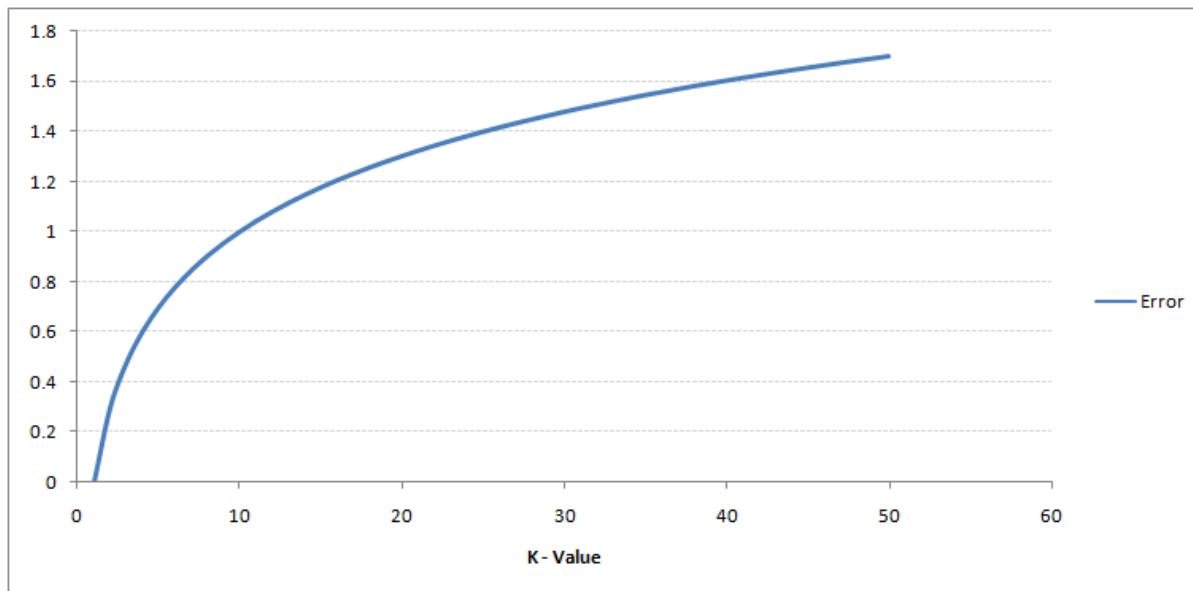
The three closest points to BS is all RC. Hence, with good confidence level we can say that the BS should belong to the class RC. Here, the choice became very obvious as all three votes from the closest neighbor went to RC. The choice of the parameter K is very crucial in this algorithm. Next we will understand what are the factors to be considered to conclude the best K.

How do we choose the factor K?

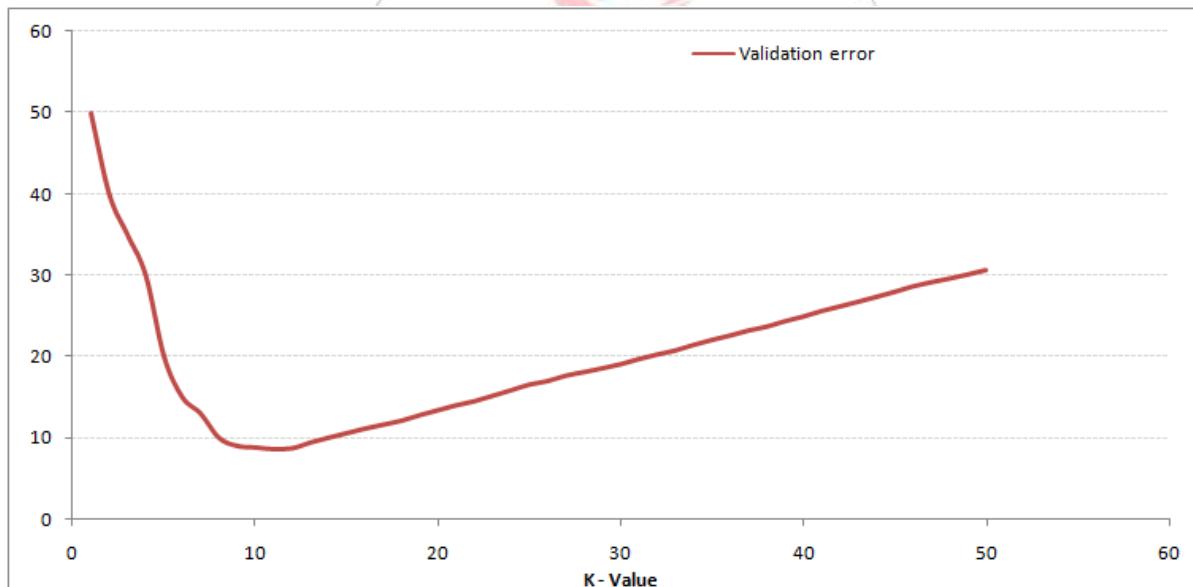
First let us try to understand what exactly does K influence in the algorithm. If we see the last example, given that all the 6 training observation remain constant, with a given K value we can make boundaries of each class. These boundaries will segregate RC from GS. The same way, let's try to see the effect of value "K" on the class boundaries. Following are the different boundaries separating the two classes with different values of K.



If you watch carefully, you can see that the boundary becomes smoother with increasing value of K. With K increasing to infinity it finally becomes all blue or all red depending on the total majority. The training error rate and the validation error rate are two parameters we need to access on different K-value. Following is the curve for the training error rate with varying value of K :



As you can see, the error rate at $K=1$ is always zero for the training sample. This is because the closest point to any training data point is itself. Hence the prediction is always accurate with $K=1$. If validation error curve would have been similar, our choice of K would have been 1. Following is the validation error curve with varying value of K :



This makes the story more clear. At $K=1$, we were overfitting the boundaries. Hence, error rate initially decreases and reaches a minima. After the minima point, it then increases with increasing K . To get the optimal value of K , you can segregate the training and validation from the initial dataset. Now plot the validation error curve to get the optimal value of K . This value of K should be used for all predictions.

Breaking it Down – Pseudo Code of KNN

We can implement a KNN model by following the below steps:

1. Load the data
2. Initialise the value of k
3. For getting the predicted class, iterate from 1 to total number of training data points
 1. Calculate the distance between test data and each row of training data. Here we will use Euclidean distance as our distance metric since it's the most popular method. The other metrics that can be used are Chebyshev, cosine, etc.
 2. Sort the calculated distances in ascending order based on distance values
 3. Get top k rows from the sorted array
 4. Get the most frequent class of these rows
 5. Return the predicted class

Implementation in Python from scratch

```
# Importing libraries
import pandas as pd
import numpy as np
import math
import operator
#### Start of STEP 1
# Importing data
data = pd.read_csv("iris.csv")
#### End of STEP 1
data.head()
```



	SepalLength	SepalWidth	PetalLength	PetalWidth	Name
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

```
# Defining a function which calculates euclidean distance between two data points
def euclideanDistance(data1, data2, length):
    distance = 0
    for x in range(length):
        distance += np.square(data1[x] - data2[x])
    return np.sqrt(distance)
# Defining our KNN model
def knn(trainingSet, testInstance, k):

    distances = {}
    sort = {}

    length = testInstance.shape[1]
```

```

##### Start of STEP 3
# Calculating euclidean distance between each row of training data and test data
for x in range(len(trainingSet)):

    ##### Start of STEP 3.1
    dist = euclideanDistance(testInstance, trainingSet.iloc[x], length)
    distances[x] = dist[0]
    ##### End of STEP 3.1

    ##### Start of STEP 3.2
    # Sorting them on the basis of distance
    sorted_d = sorted(distances.items(), key=operator.itemgetter(1))
    ##### End of STEP 3.2

    neighbors = []

    ##### Start of STEP 3.3
    # Extracting top k neighbors
    for x in range(k):
        neighbors.append(sorted_d[x][0])
    ##### End of STEP 3.3
    classVotes = {}

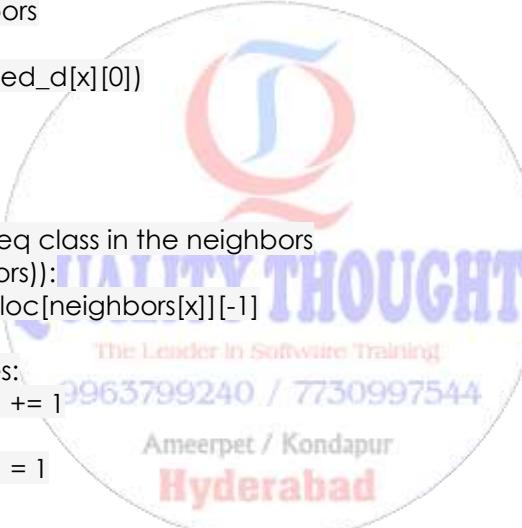
    ##### Start of STEP 3.4
    # Calculating the most freq class in the neighbors
    for x in range(len(neighbors)):
        response = trainingSet.iloc[neighbors[x]][-1]

        if response in classVotes:
            classVotes[response] += 1
        else:
            classVotes[response] = 1
    ##### End of STEP 3.4

    ##### Start of STEP 3.5
    sortedVotes = sorted(classVotes.items(), key=operator.itemgetter(1), reverse=True)
    return(sortedVotes[0][0], neighbors)
    ##### End of STEP 3.5

# Creating a dummy testset
testSet = [[7.2, 3.6, 5.1, 2.5]]
test = pd.DataFrame(testSet)
##### Start of STEP 2
# Setting number of neighbors = 1
k = 1
##### End of STEP 2
# Running KNN model
result,neigh = knn(data, test, k)
# Predicted class
print(result)
-> Iris-virginica

```



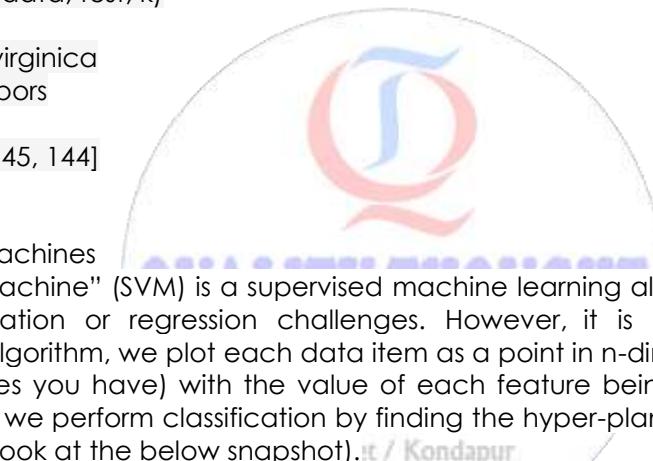
```
# Nearest neighbor  
print(neigh)  
-> [141]
```

Now we will try to alter the k values, and see how the prediction changes.

```
# Setting number of neighbors = 3  
k = 3  
# Running KNN model  
result,neigh = knn(data, test, k)  
# Predicted class  
print(result) -> Iris-virginica  
# 3 nearest neighbors  
print(neigh)  
-> [141, 139, 120]  
# Setting number of neighbors = 5  
k = 5  
# Running KNN model  
result,neigh = knn(data, test, k)  
# Predicted class  
print(result) -> Iris-virginica  
# 5 nearest neighbors  
print(neigh)  
-> [141, 139, 120, 145, 144]
```

Support Vector Machines

Support Vector Machine" (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In this algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiate the two classes very well (look at the below snapshot).[it / Kondapur](http://it/kondapur)



Support Vectors are simply the co-ordinates of individual observation. Support Vector Machine is a frontier which best segregates the two classes (hyper-plane/ line).

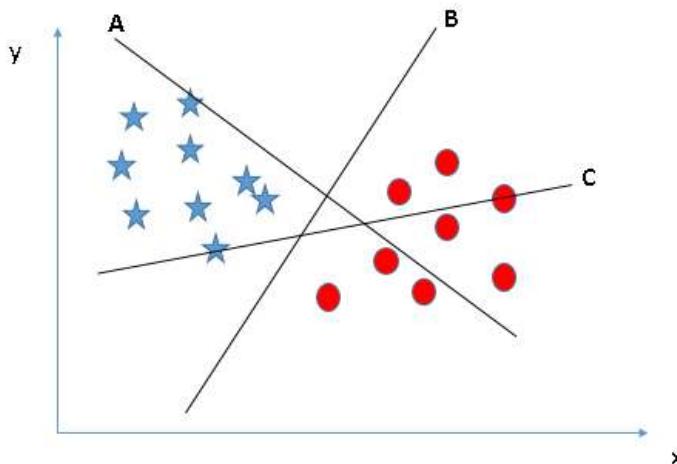
You can look at definition of support vectors and a few examples of its working here.

How does it work?

Above, we got accustomed to the process of segregating the two classes with a hyper-plane. Now the burning question is "How can we identify the right hyper-plane?". Don't worry, it's not as hard as you think!

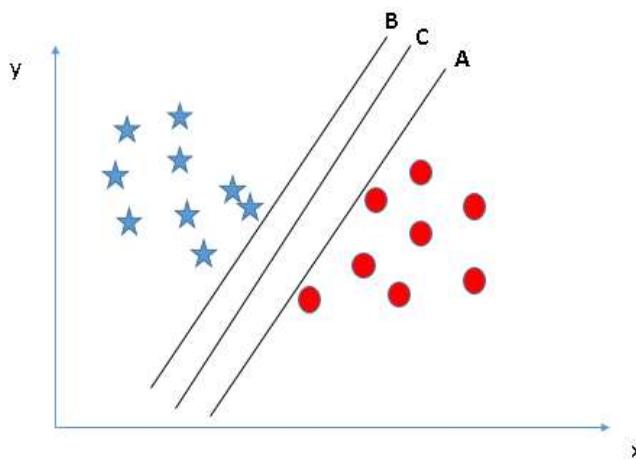
Let's understand:

- **Identify the right hyper-plane (Scenario-1):** Here, we have three hyper-planes (A, B and C). Now, identify the right hyper-plane to classify star and circle.

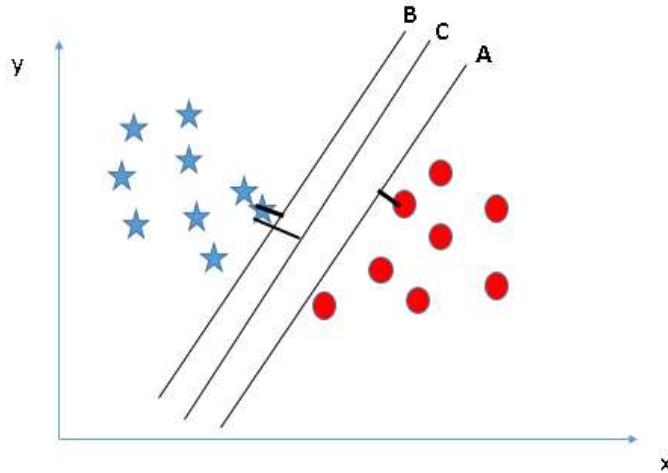


You need to remember a thumb rule to identify the right hyper-plane: "Select the hyper-plane which segregates the two classes better". In this scenario, hyper-plane "B" has excellently performed this job.

- **Identify the right hyper-plane (Scenario-2):** Here, we have three hyper-planes (A, B and C) and all are segregating the classes well. Now, How can we identify the right hyper-plane?

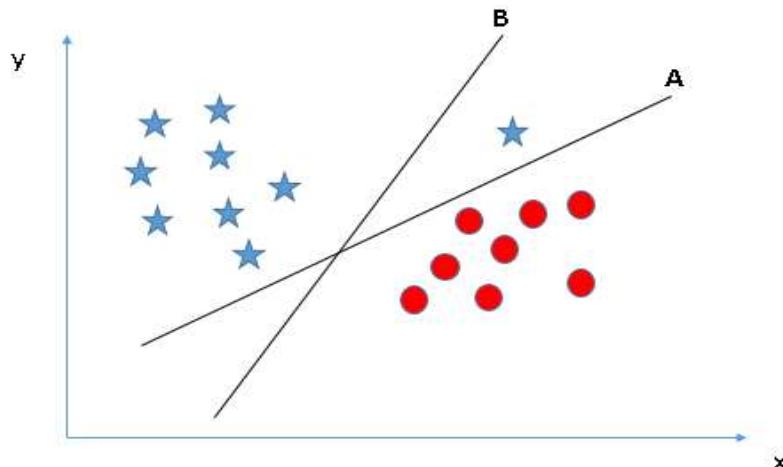


Here, maximizing the distances between nearest data point (either class) and hyper-plane will help us to decide the right hyper-plane. This distance is called as **Margin**. Let's look at the below snapshot:



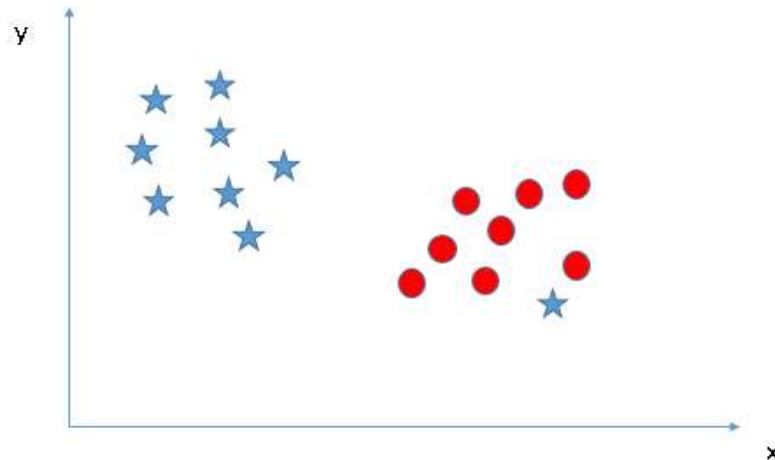
Above, you can see that the margin for hyper-plane C is high as compared to both A and B. Hence, we name the right hyper-plane as C. Another lightning reason for selecting the hyper-plane with higher margin is robustness. If we select a hyper-plane having low margin then there is high chance of miss-classification.

- **Identify the right hyper-plane (Scenario-3):** Hint: Use the rules as discussed in previous section to identify the right hyper-plane

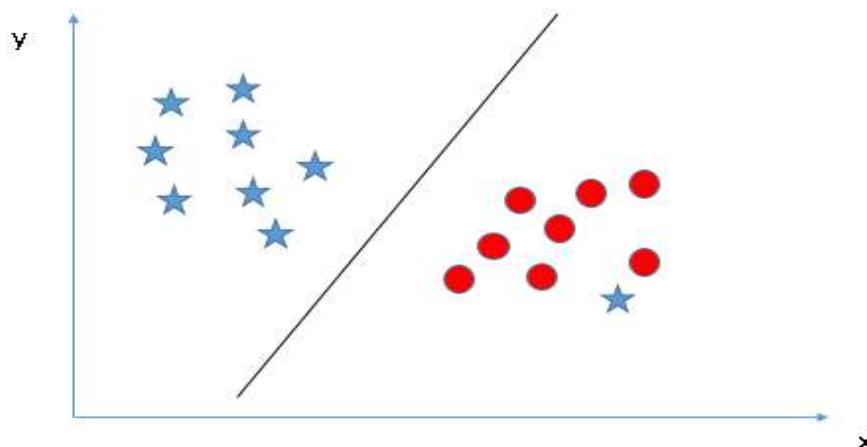


Some of you may have selected the hyper-plane **B** as it has higher margin compared to **A**. But, here is the catch, SVM selects the hyper-plane which classifies the classes accurately prior to maximizing margin. Here, hyper-plane B has a classification error and A has classified all correctly. Therefore, the right hyper-plane is **A**.

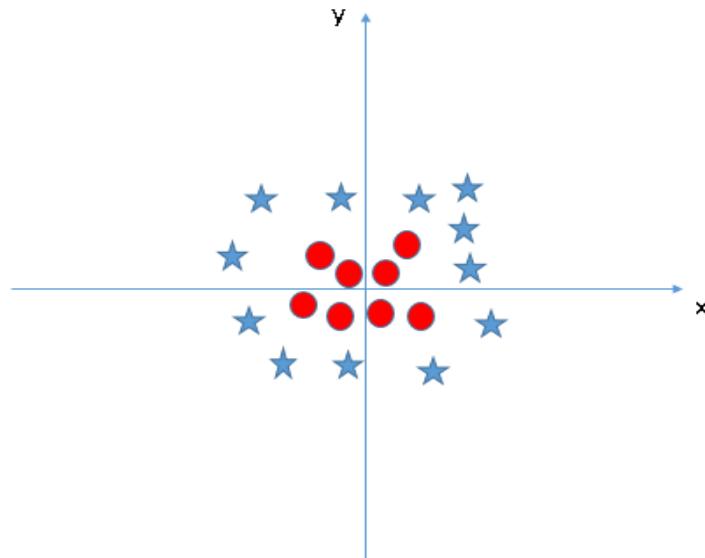
- **Can we classify two classes (Scenario-4)?:** Below, I am unable to segregate the two classes using a straight line, as one of star lies in the territory of other(circle) class as an outlier.



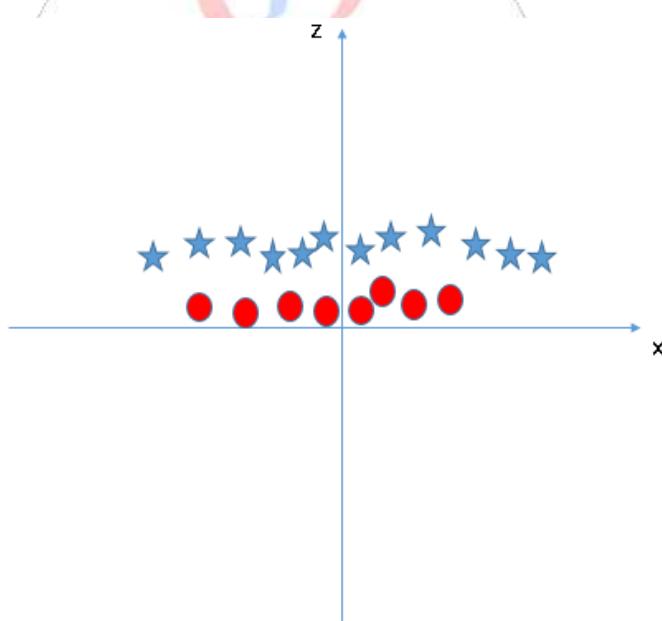
As I have already mentioned, one star at other end is like an outlier for star class. SVM has a feature to ignore outliers and find the hyper-plane that has maximum margin. Hence, we can say, SVM is robust to outliers.



- **Find the hyper-plane to segregate to classes (Scenario-5):** In the scenario below, we can't have linear hyper-plane between the two classes, so how does SVM classify these two classes? Till now, we have only looked at the linear hyper-plane.



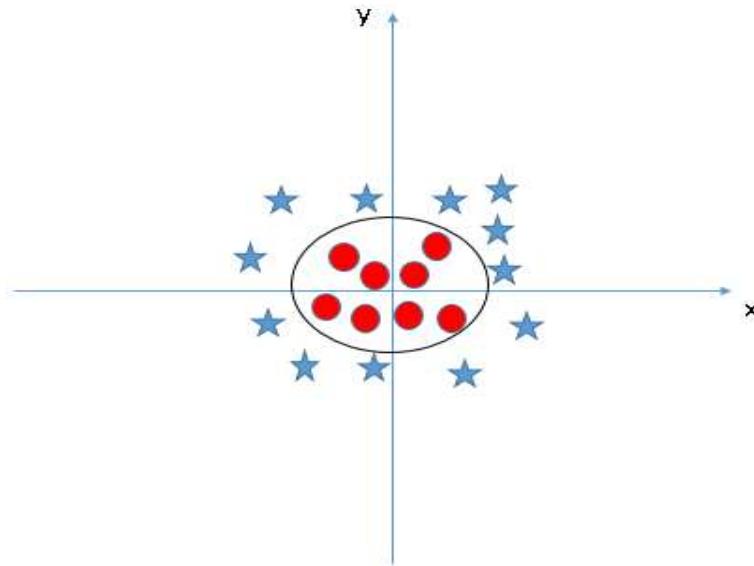
SVM can solve this problem. Easily! It solves this problem by introducing additional feature. Here, we will add a new feature $z=x^2+y^2$. Now, let's plot the data points on axis x and z:



In above plot, points to consider are:

- All values for z would be positive always because z is the squared sum of both x and y
 - In the original plot, red circles appear close to the origin of x and y axes, leading to lower value of z and star relatively away from the origin result to higher value of z.
- In SVM, it is easy to have a linear hyper-plane between these two classes. But, another burning question which arises is, should we need to add this feature manually to have a hyper-plane. No, SVM has a technique called the **kernel trick**. These are functions which takes low dimensional input space and transform it to a higher dimensional space i.e. it converts not separable problem to separable problem, these functions are called kernels. It is mostly useful in non-linear separation

problem. Simply put, it does some extremely complex data transformations, then find out the process to separate the data based on the labels or outputs you've defined. When we look at the hyper-plane in original input space it looks like a circle:



Now, let's look at the methods to apply SVM algorithm in a data science challenge.

How to implement SVM in Python and R?

In Python, scikit-learn is a widely used library for implementing machine learning algorithms, SVM is also available in scikit-learn library and follow the same structure (Import library, object creation, fitting model and prediction). Let's look at the below code:

```
#Import Library
from sklearn import svm
#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predictor) of
test_dataset
# Create SVM classification object
model = svm.SVC(kernel='linear', C=1, gamma=1)
# there are various option associated with it, like changing kernel, gamma and C value. Will
discuss more about it in next section. Train the model using the training sets and check score
model.fit(X, y)
model.score(X, y)
#Predict Output
predicted = model.predict(x_test)
```

The e1071 package in R is used to create Support Vector Machines with ease. It has helper functions as well as code for the Naive Bayes Classifier. The creation of a support vector machine in R and Python follow similar approaches, let's take a look now at the following code:

```
#Import Library
require(e1071) #Contains the SVM
Train <- read.csv(file.choose())
Test <- read.csv(file.choose())
```

```
# there are various options associated with SVM training; like changing kernel, gamma and C
value.
# create model
model <-
svm(Target~Predictor1+Predictor2+Predictor3,data=Train,kernel='linear',gamma=0.2, cost=100)
#Predict Output
preds <- predict(model,Test)
table(preds)
```

How to tune Parameters of SVM?

Tuning parameters value for machine learning algorithms effectively improves the model performance. Let's look at the list of parameters available with SVM.

```
sklearn.svm.SVC(C=1.0, kernel='rbf', degree=3, gamma=0.0, coef0=0.0, shrinking=True,
probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1,
random_state=None)
```

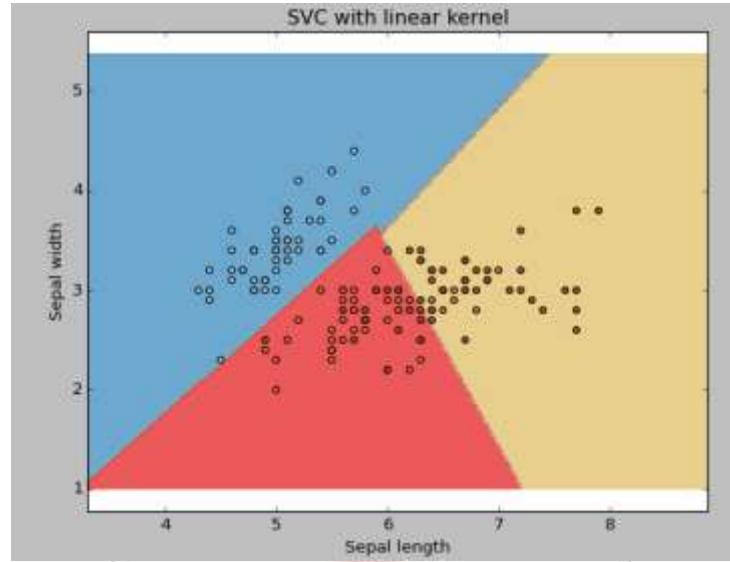
I am going to discuss about some important parameters having higher impact on model performance, "kernel", "gamma" and "C".

kernel: We have already discussed about it. Here, we have various options available with kernel like, "linear", "rbf", "poly" and others (default value is "rbf"). Here "rbf" and "poly" are useful for non-linear hyper-plane. Let's look at the example, where we've used linear kernel on two feature of iris data set to classify their class.

Example: Have linear kernel

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets
# import some data to play with
iris = datasets.load_iris()
X = iris.data[:, :2] # we only take the first two features. We could
# avoid this ugly slicing by using a two-dim dataset
y = iris.target
# we create an instance of SVM and fit out data. We do not scale our
# data since we want to plot the support vectors
C = 1.0 # SVM regularization parameter
svc = svm.SVC(kernel='linear', C=1, gamma=0).fit(X, y)
# create a mesh to plot in
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
h = (x_max / x_min)/100
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
np.arange(y_min, y_max, h))
plt.subplot(1, 1, 1)
Z = svc.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)
```

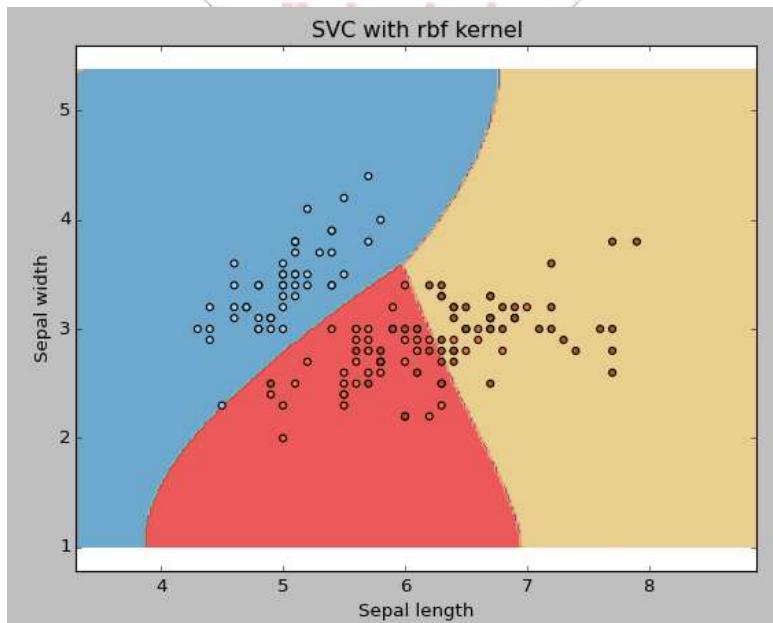
```
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Paired)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.xlim(xx.min(), xx.max())
plt.title('SVC with linear kernel')
plt.show()
```



Example: Have rbf kernel

Change the kernel type to rbf in below line and look at the impact.

```
svc = svm.SVC(kernel='rbf', C=1, gamma=0).fit(X, y)
```



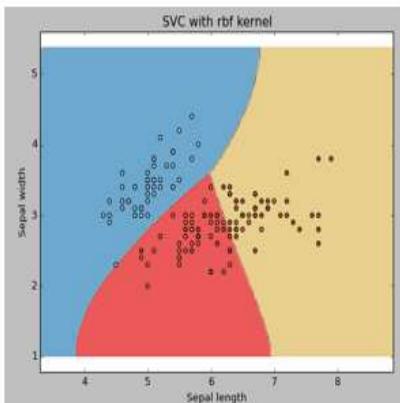
I would suggest you to go for linear kernel if you have large number of features (>1000) because it is more likely that the data is linearly separable in high dimensional space. Also, you can RBF but do not forget to cross validate for its parameters as to avoid over-fitting.

gamma: Kernel coefficient for 'rbf', 'poly' and 'sigmoid'. Higher the value of gamma, will try to exact fit the as per training data set i.e. generalization error and cause over-fitting problem.

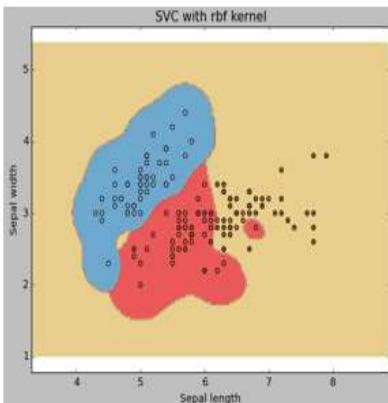
Example: Let's difference if we have gamma different gamma values like 0, 10 or 100.

```
svc = svm.SVC(kernel='rbf', C=1, gamma=0).fit(X, y)
```

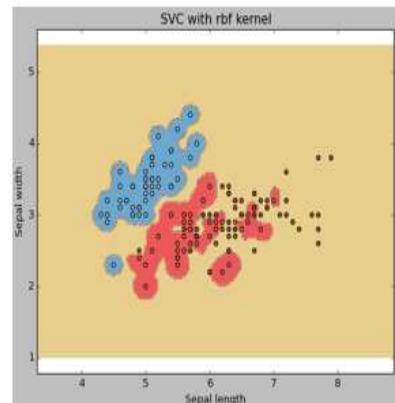
gamma = 0



gamma = 10

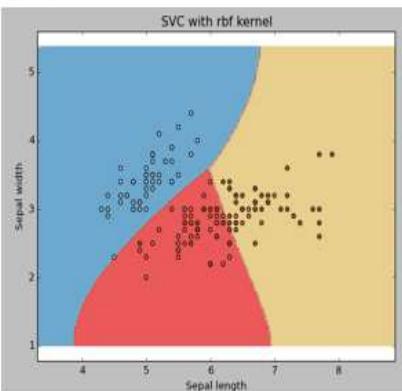


gamma = 100

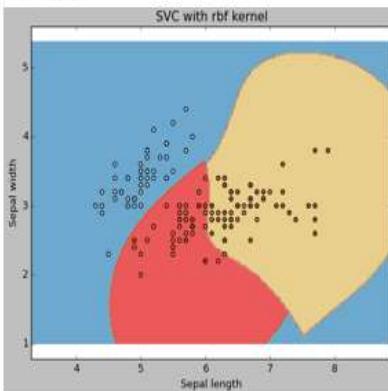


C: Penalty parameter C of the error term. It also controls the trade-off between smooth decision boundary and classifying the training points correctly.

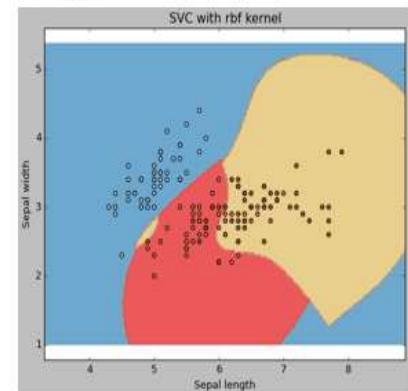
c = 1



c = 100



c = 1000



We should always look at the cross validation score to have effective combination of these parameters and avoid over-fitting.

In R, SVMs can be tuned in a similar fashion as they are in Python. Mentioned below are the respective parameters for e1071 package:

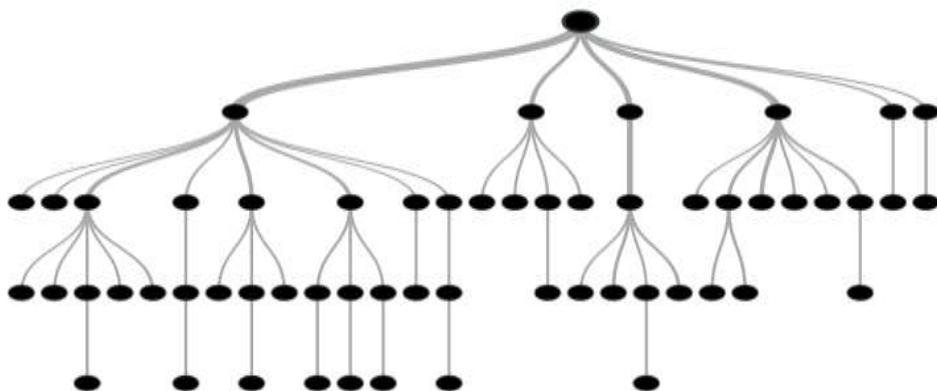
- The kernel parameter can be tuned to take "Linear", "Poly", "rbf" etc.
- The gamma value can be tuned by setting the "Gamma" parameter.
- The C value in Python is tuned by the "Cost" parameter in R.

Pros and Cons associated with SVM

- **Pros:**
 - It works really well with clear margin of separation
 - It is effective in high dimensional spaces.
 - It is effective in cases where number of dimensions is greater than the number of samples.
 - It uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- **Cons:**
 - It doesn't perform well, when we have large data set because the required training time is higher
 - It also doesn't perform very well, when the data set has more noise i.e. target classes are overlapping
 - SVM doesn't directly provide probability estimates, these are calculated using an expensive five-fold cross-validation. It is related SVC method of Python scikit-learn library.

Tree Based Algorithms**1. What is a Decision Tree ? How does it work ?**

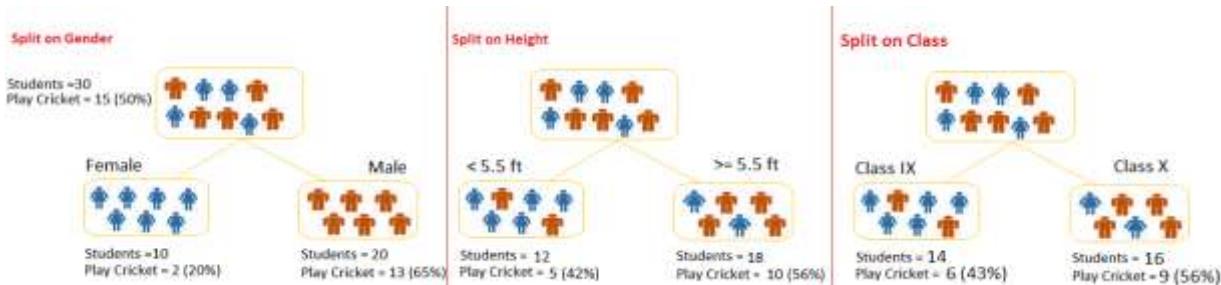
Decision tree is a type of supervised learning algorithm (having a pre-defined target variable) that is mostly used in classification problems. It works for both categorical and continuous input and output variables. In this technique, we split the population or sample into two or more homogeneous sets (or sub-populations) based on most significant splitter / differentiator in input variables.

**Example:-**

Let's say we have a sample of 30 students with three variables Gender (Boy/ Girl), Class(IX/ X) and Height (5 to 6 ft). 15 out of these 30 play cricket in leisure time. Now, I want to create a model to predict who will play cricket during leisure period? In this problem, we need to segregate students who play cricket in their leisure time based on highly significant input variable among all three.

This is where decision tree helps, it will segregate the students based on all values of three variable and identify the variable, which creates the best homogeneous sets of students (which

are heterogeneous to each other). In the snapshot below, you can see that variable Gender is able to identify best homogeneous sets compared to the other two variables.



As mentioned above, decision tree identifies the most significant variable and it's value that gives best homogeneous sets of population. Now the question which arises is, how does it identify the variable and the split? To do this, decision tree uses various algorithms, which we will shall discuss in the following section.

Types of Decision Trees

Types of decision tree is based on the type of target variable we have. It can be of two types:

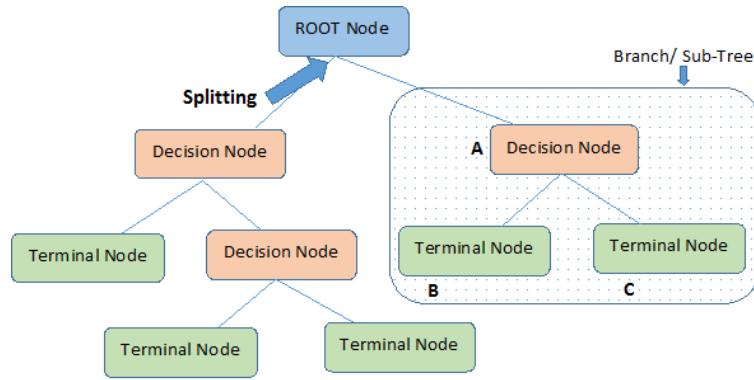
- Categorical Variable Decision Tree:** Decision Tree which has categorical target variable then it called as categorical variable decision tree. Example:- In above scenario of student problem, where the target variable was "Student will play cricket or not" i.e. YES or NO.
- Continuous Variable Decision Tree:** Decision Tree has continuous target variable then it is called as Continuous Variable Decision Tree.

Example:- Let's say we have a problem to predict whether a customer will pay his renewal premium with an insurance company (yes/ no). Here we know that income of customer is a significant variable but insurance company does not have income details for all customers. Now, as we know this is an important variable, then we can build a decision tree to predict customer income based on occupation, product and various other variables. In this case, we are predicting values for continuous variable.

Important Terminology related to Decision Trees

Let's look at the basic terminology used with Decision trees:

- Root Node:** It represents entire population or sample and this further gets divided into two or more homogeneous sets.
- Splitting:** It is a process of dividing a node into two or more sub-nodes.
- Decision Node:** When a sub-node splits into further sub-nodes, then it is called decision node.
- Leaf/ Terminal Node:** Nodes do not split is called Leaf or Terminal node.



Note:- A is parent node of B and C.

Pruning: When we remove sub-nodes of a decision node, this process is called pruning. You can say opposite process of splitting.

5. **Branch / Sub-Tree:** A sub section of entire tree is called branch or sub-tree.
6. **Parent and Child Node:** A node, which is divided into sub-nodes is called parent node of sub-nodes where as sub-nodes are the child of parent node.

These are the terms commonly used for decision trees. As we know that every algorithm has advantages and disadvantages, below are the important factors which one should know.

Advantages

1. **Easy to Understand:** Decision tree output is very easy to understand even for people from non-analytical background. It does not require any statistical knowledge to read and interpret them. Its graphical representation is very intuitive and users can easily relate their hypothesis.
2. **Useful in Data exploration:** Decision tree is one of the fastest way to identify most significant variables and relation between two or more variables. With the help of decision trees, we can create new variables / features that has better power to predict target variable. You can refer article (**Trick to enhance power of regression model**) for one such trick. It can also be used in data exploration stage. For example, we are working on a problem where we have information available in hundreds of variables, there decision tree will help to identify most significant variable.
3. **Less data cleaning required:** It requires less data cleaning compared to some other modeling techniques. It is not influenced by outliers and missing values to a fair degree.
4. **Data type is not a constraint:** It can handle both numerical and categorical variables.
5. **Non Parametric Method:** Decision tree is considered to be a non-parametric method. This means that decision trees have no assumptions about the space distribution and the classifier structure.

Disadvantages

1. **Over fitting:** Over fitting is one of the most practical difficulty for decision tree models. This problem gets solved by setting constraints on model parameters and pruning (discussed in detailed below).
2. **Not fit for continuous variables:** While working with continuous numerical variables, decision tree loses information when it categorizes variables in different categories.

2. Regression Trees vs Classification Trees

We all know that the terminal nodes (or leaves) lies at the bottom of the decision tree. This means that decision trees are typically drawn upside down such that leaves are the the bottom & roots are the tops (shown below).



Both the trees work almost similar to each other, let's look at the primary differences & similarity between classification and regression trees:

1. Regression trees are used when dependent variable is continuous. Classification trees are used when dependent variable is categorical.
2. In case of regression tree, the value obtained by terminal nodes in the training data is the mean response of observation falling in that region. Thus, if an unseen data observation falls in that region, we'll make its prediction with mean value.
3. In case of classification tree, the value (class) obtained by terminal node in the training data is the mode of observations falling in that region. Thus, if an unseen data observation falls in that region, we'll make its prediction with mode value.
4. Both the trees divide the predictor space (independent variables) into distinct and non-overlapping regions. For the sake of simplicity, you can think of these regions as high dimensional boxes or boxes.
5. Both the trees follow a top-down greedy approach known as recursive binary splitting. We call it as 'top-down' because it begins from the top of tree when all the observations are available in a single region and successively splits the predictor space into two new branches down the tree. It is known as 'greedy' because, the algorithm cares (looks for best variable available) about only the current split, and not about future splits which will lead to a better tree.

6. This splitting process is continued until a user defined stopping criteria is reached. For example: we can tell the the algorithm to stop once the number of observations per node becomes less than 50.
7. In both the cases, the splitting process results in fully grown trees until the stopping criteria is reached. But, the fully grown tree is likely to overfit data, leading to poor accuracy on unseen data. This bring 'pruning'. Pruning is one of the technique used tackle overfitting. We'll learn more about it in following section.

3. How does a tree decide where to split?

The decision of making strategic splits heavily affects a tree's accuracy. The decision criteria is different for classification and regression trees.

Decision trees use multiple algorithms to decide to split a node in two or more sub-nodes. The creation of sub-nodes increases the homogeneity of resultant sub-nodes. In other words, we can say that purity of the node increases with respect to the target variable. Decision tree splits the nodes on all available variables and then selects the split which results in most homogeneous sub-nodes.

The algorithm selection is also based on type of target variables. Let's look at the four most commonly used algorithms in decision tree:

Gini Index

Gini index says, if we select two items from a population at random then they must be of same class and probability for this is 1 if population is pure.

1. It works with categorical target variable "Success" or "Failure".
2. It performs only Binary splits
3. Higher the value of Gini higher the homogeneity.
4. CART (Classification and Regression Tree) uses Gini method to create binary splits.

Steps to Calculate Gini for a split

1. Calculate Gini for sub-nodes, using formula sum of square of probability for success and failure (p^2+q^2).
2. Calculate Gini for split using weighted Gini score of each node of that split

Example: – Referring to example used above, where we want to segregate the students based on target variable (playing cricket or not). In the snapshot below, we split the population using two input variables Gender and Class. Now, I want to identify which split is producing more homogeneous sub-nodes using Gini index.

Split on Gender

Students = 30
Play Cricket = 15 (50%)



Female



Students = 10

Play Cricket = 2 (20%)

Male



Students = 20

Play Cricket = 13 (65%)

Split on Class



Class IX



Students = 14

Play Cricket = 6 (43%)

Class X



Students = 16

Play Cricket = 9 (56%)

Split on Gender:

1. Calculate, Gini for sub-node Female = $(0.2)*(0.2)+(0.8)*(0.8)=0.68$
2. Gini for sub-node Male = $(0.65)*(0.65)+(0.35)*(0.35)=0.55$
3. Calculate weighted Gini for Split Gender = $(10/30)*0.68+(20/30)*0.55 = 0.59$

Similar for Split on Class:

1. Gini for sub-node Class IX = $(0.43)*(0.43)+(0.57)*(0.57)=0.51$
2. Gini for sub-node Class X = $(0.56)*(0.56)+(0.44)*(0.44)=0.51$
3. Calculate weighted Gini for Split Class = $(14/30)*0.51+(16/30)*0.51 = 0.51$

Above, you can see that Gini score for Split on Gender is higher than Split on Class, hence, the node split will take place on Gender.

Chi-Square



It is an algorithm to find out the statistical significance between the differences between sub-nodes and parent node. We measure it by sum of squares of standardized differences between observed and expected frequencies of target variable.

1. It works with categorical target variable "Success" or "Failure".
2. It can perform two or more splits.
3. Higher the value of Chi-Square higher the statistical significance of differences between sub-node and Parent node.
4. Chi-Square of each node is calculated using formula,
5. Chi-square = $((Actual - Expected)^2 / Expected)^{1/2}$
6. It generates tree called CHAID (Chi-square Automatic Interaction Detector)

Steps to Calculate Chi-square for a split:

1. Calculate Chi-square for individual node by calculating the deviation for Success and Failure both
2. Calculated Chi-square of Split using Sum of all Chi-square of success and Failure of each node of the split

Example: Let's work with above example that we have used to calculate Gini.

Split on Gender:

1. First we are populating for node Female, Populate the actual value for "Play Cricket" and "Not Play Cricket", here these are 2 and 8 respectively.
2. Calculate expected value for "Play Cricket" and "Not Play Cricket", here it would be 5 for both because parent node has probability of 50% and we have applied same probability on Female count(10).
3. Calculate deviations by using formula, Actual – Expected. It is for "Play Cricket" (2 – 5 = -3) and for "Not play cricket" (8 – 5 = 3).
4. Calculate Chi-square of node for "Play Cricket" and "Not Play Cricket" using formula with formula, $= ((Actual - Expected)^2 / Expected)^{1/2}$. You can refer below table for calculation.
5. Follow similar steps for calculating Chi-square value for Male node.
6. Now add all Chi-square values to calculate Chi-square for split Gender.

Node	Play Cricket	Not Play Cricket	Total	Expected Play Cricket	Expected Not Play Cricket	Deviation Play Cricket	Deviation Not Play Cricket	Chi-Square	
								Play Cricket	Not Play Cricket
Female	2	8	10	5	5	-3	3	1.34	1.34
Male	13	7	20	10	10	3	-3	0.95	0.95
Total Chi-Square								4.58	

Split on Class:

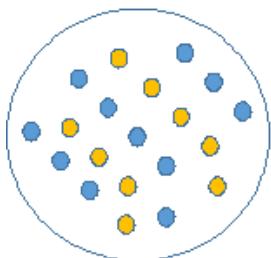
Perform similar steps of calculation for split on Class and you will come up with below table.

Node	Play Cricket	Not Play Cricket	Total	Expected Play Cricket	Expected Not Play Cricket	Deviation Play Cricket	Deviation Not Play Cricket	Chi-Square	
								Play Cricket	Not Play Cricket
IX	6	8	14	7	7	-1	1	0.38	0.38
X	9	7	16	8	8	1	-1	0.35	0.35
Total Chi-Square								1.46	

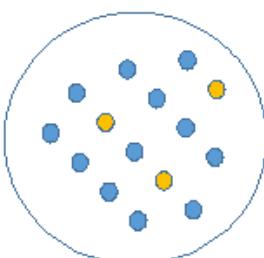
Above, you can see that Chi-square also identify the Gender split is more significant compare to Class.

Information Gain:

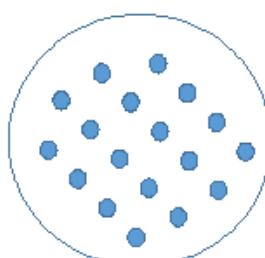
Look at the image below and think which node can be described easily. I am sure, your answer is C because it requires less information as all values are similar. On the other hand, B requires more information to describe it and A requires the maximum information. In other words, we can say that C is a Pure node, B is less Impure and A is more impure.



A



B



C

Now, we can build a conclusion that less impure node requires less information to describe it. And, more impure node requires more information. Information theory is a measure to define this degree of disorganization in a system known as Entropy. If the sample is completely homogeneous, then the entropy is zero and if the sample is an equally divided (50% – 50%), it has entropy of one.

Entropy can be calculated using formula:-

$$\text{Entropy} = -p \log_2 p - q \log_2 q$$

Here p and q is probability of success and failure respectively in that node. Entropy is also used with categorical target variable. It chooses the split which has lowest entropy compared to parent node and other splits. The lesser the entropy, the better it is.

Steps to calculate entropy for a split:

1. Calculate entropy of parent node
2. Calculate entropy of each individual node of split and calculate weighted average of all sub-nodes available in split.

Example: Let's use this method to identify best split for student example.

1. Entropy for parent node = $-(15/30) \log_2 (15/30) - (15/30) \log_2 (15/30) = 1$. Here 1 shows that it is a impure node.
2. Entropy for Female node = $-(2/10) \log_2 (2/10) - (8/10) \log_2 (8/10) = 0.72$ and for male node, $-(13/20) \log_2 (13/20) - (7/20) \log_2 (7/20) = 0.93$
3. Entropy for split Gender = Weighted entropy of sub-nodes = $(10/30)*0.72 + (20/30)*0.93 = 0.86$
4. Entropy for Class IX node, $-(6/14) \log_2 (6/14) - (8/14) \log_2 (8/14) = 0.99$ and for Class X node, $-(9/16) \log_2 (9/16) - (7/16) \log_2 (7/16) = 0.99$.
5. Entropy for split Class = $(14/30)*0.99 + (16/30)*0.99 = 0.99$

Above, you can see that entropy for Split on Gender is the lowest among all, so the tree will split on Gender. We can derive information gain from entropy as **1 - Entropy**.

Reduction in Variance

Till now, we have discussed the algorithms for categorical target variable. Reduction in variance is an algorithm used for continuous target variables (regression problems). This algorithm uses the standard formula of variance to choose the best split. The split with lower variance is selected as the criteria to split the population:

$$\text{Variance} = \frac{\sum(X - \bar{X})^2}{n}$$

Above \bar{X} is mean of the values, X is actual and n is number of values.

Steps to calculate Variance:

1. Calculate variance for each node.
2. Calculate variance for each split as weighted average of each node variance.

Example:- Let's assign numerical value 1 for play cricket and 0 for not playing cricket. Now follow the steps to identify the right split:

1. Variance for Root node, here mean value is $(15*1 + 15*0)/30 = 0.5$ and we have 15 one and 15 zero. Now variance would be $((1-0.5)^2 + (1-0.5)^2 + \dots + 15 \text{ times} + (0-0.5)^2 + (0-0.5)^2 + \dots + 15 \text{ times}) / 30$, this can be written as $(15*(1-0.5)^2 + 15*(0-0.5)^2) / 30 = 0.25$
2. Mean of Female node = $(2*1 + 8*0)/10 = 0.2$ and Variance = $(2*(1-0.2)^2 + 8*(0-0.2)^2) / 10 = 0.16$
3. Mean of Male Node = $(13*1 + 7*0)/20 = 0.65$ and Variance = $(13*(1-0.65)^2 + 7*(0-0.65)^2) / 20 = 0.23$
4. Variance for Split Gender = Weighted Variance of Sub-nodes = $(10/30)*0.16 + (20/30)*0.23 = 0.21$
5. Mean of Class IX node = $(6*1 + 8*0)/14 = 0.43$ and Variance = $(6*(1-0.43)^2 + 8*(0-0.43)^2) / 14 = 0.24$
6. Mean of Class X node = $(9*1 + 7*0)/16 = 0.56$ and Variance = $(9*(1-0.56)^2 + 7*(0-0.56)^2) / 16 = 0.25$
7. Variance for Split Gender = $(14/30)*0.24 + (16/30)*0.25 = 0.25$

Above, you can see that Gender split has lower variance compare to parent node, so the split would take place on Gender variable.

Until here, we learnt about the basics of decision trees and the decision making process involved to choose the best splits in building a tree model. As I said, decision tree can be applied both on regression and classification problems. Let's understand these aspects in detail.

4. What are the key parameters of tree modeling and how can we avoid over-fitting in decision trees?

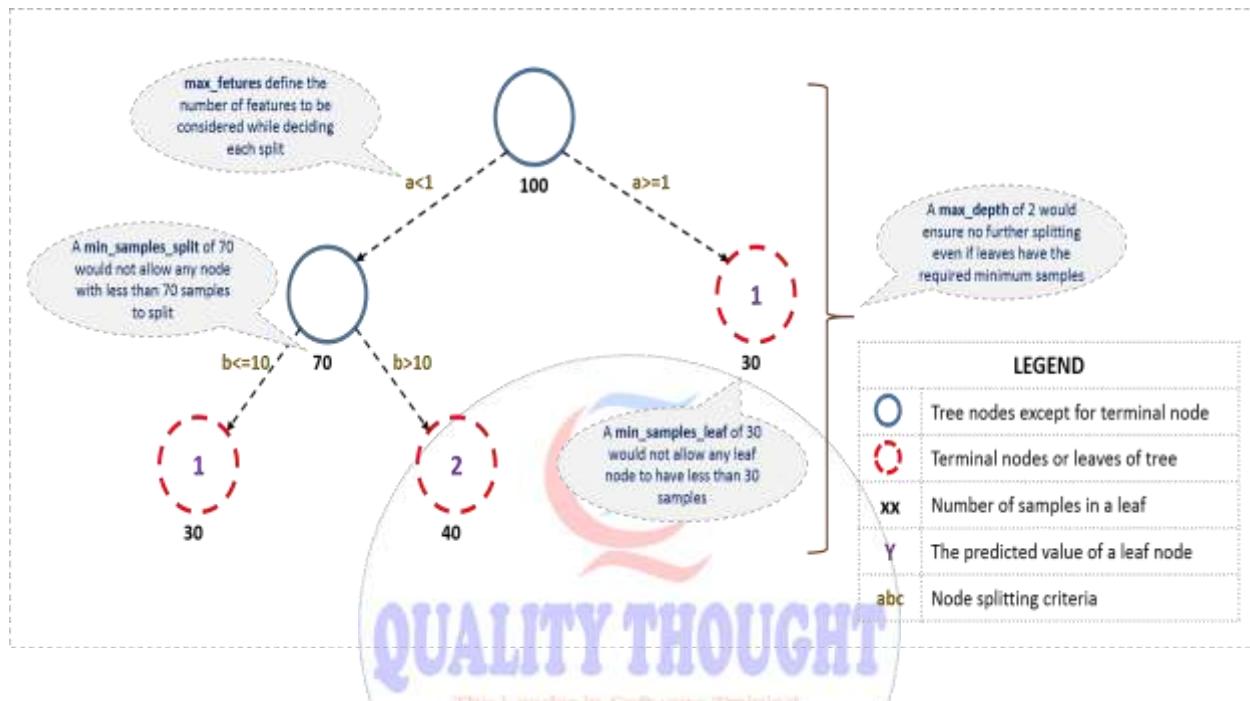
Overfitting is one of the key challenges faced while modeling decision trees. If there is no limit set of a decision tree, it will give you 100% accuracy on training set because in the worse case it will end up making 1 leaf for each observation. Thus, preventing overfitting is pivotal while modeling a decision tree and it can be done in 2 ways:

1. Setting constraints on tree size
2. Tree pruning

Lets discuss both of these briefly.

Setting Constraints on Tree Size

This can be done by using various parameters which are used to define a tree. First, lets look at the general structure of a decision tree:



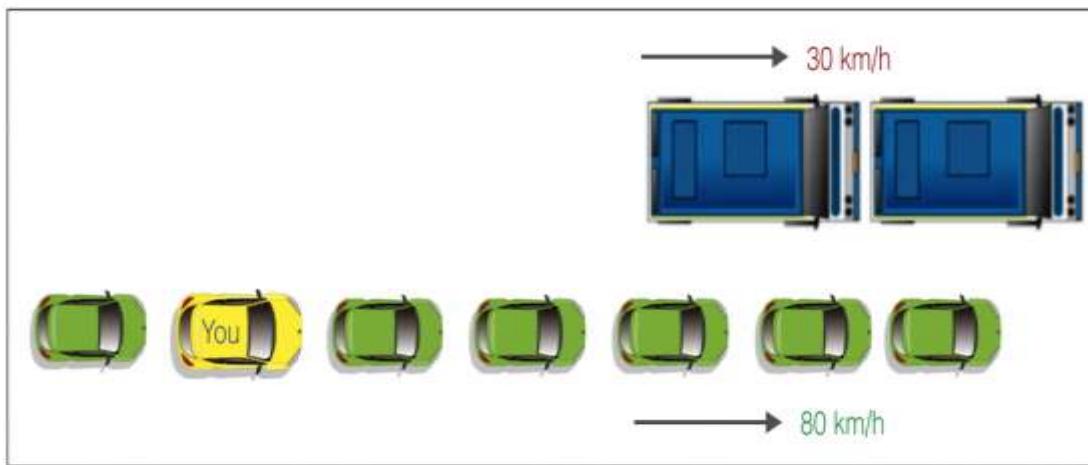
The parameters used for defining a tree are further explained below. The parameters described below are irrespective of tool. It is important to understand the role of parameters used in tree modeling. These parameters are available in R & Python.

- 1. Minimum samples for a node split**
 - Defines the minimum number of samples (or observations) which are required in a node to be considered for splitting.
 - Used to control over-fitting. Higher values prevent a model from learning relations which might be highly specific to the particular sample selected for a tree.
 - Too high values can lead to under-fitting hence, it should be tuned using CV.
- 2. Minimum samples for a terminal node (leaf)**
 - Defines the minimum samples (or observations) required in a terminal node or leaf.
 - Used to control over-fitting similar to `min_samples_split`.
 - Generally lower values should be chosen for imbalanced class problems because the regions in which the minority class will be in majority will be very small.
- 3. Maximum depth of tree (vertical depth)**
 - The maximum depth of a tree.
 - Used to control over-fitting as higher depth will allow model to learn relations very specific to a particular sample.
 - Should be tuned using CV.
- 4. Maximum number of terminal nodes**
 - The maximum number of terminal nodes or leaves in a tree.

- Can be defined in place of max_depth. Since binary trees are created, a depth of 'n' would produce a maximum of 2^n leaves.
5. Maximum features to consider for split
- The number of features to consider while searching for a best split. These will be randomly selected.
 - As a thumb-rule, square root of the total number of features works great but we should check upto 30-40% of the total number of features.
 - Higher values can lead to over-fitting but depends on case to case.

Tree Pruning

As discussed earlier, the technique of setting constraint is a greedy-approach. In other words, it will check for the best split instantaneously and move forward until one of the specified stopping condition is reached. Let's consider the following case when you're driving:



There are 2 lanes:

1. A lane with cars moving at 80km/h
2. A lane with trucks moving at 30km/h

At this instant, you are the yellow car and you have 2 choices:

1. Take a left and overtake the other 2 cars quickly
2. Keep moving in the present lane

Let's analyze these choice. In the former choice, you'll immediately overtake the car ahead and reach behind the truck and start moving at 30 km/h, looking for an opportunity to move back right. All cars originally behind you move ahead in the meanwhile. This would be the optimum choice if your objective is to maximize the distance covered in next say 10 seconds. In the later choice, you sail through at same speed, cross trucks and then overtake maybe depending on situation ahead. Greedy you!



This is exactly the difference between normal decision tree & pruning. A decision tree with constraints won't see the truck ahead and adopt a greedy approach by taking a left. On the other hand if we use pruning, we in effect look at a few steps ahead and make a choice.

So we know pruning is better. But how to implement it in decision tree? The idea is simple.

1. We first make the decision tree to a large depth.
2. Then we start at the bottom and start removing leaves which are giving us negative returns when compared from the top.
3. Suppose a split is giving us a gain of say -10 (loss of 10) and then the next split on that gives us a gain of 20. A simple decision tree will stop at step 1 but in pruning, we will see that the overall gain is +10 and keep both leaves.

Note that sklearn's decision tree classifier does not currently support pruning. Advanced packages like xgboost have adopted tree pruning in their implementation. But the library rpart in R, provides a function to prune. Good for R users!

5. Are tree based models better than linear models?

"If I can use logistic regression for classification problems and linear regression for regression problems, why is there a need to use trees"? Many of us have this question. And, this is a valid one too.

Actually, you can use any algorithm. It is dependent on the type of problem you are solving. Let's look at some key factors which will help you to decide which algorithm to use:

1. If the relationship between dependent & independent variable is well approximated by a linear model, linear regression will outperform tree based model.
2. If there is a high non-linearity & complex relationship between dependent & independent variables, a tree model will outperform a classical regression method.
3. If you need to build a model which is easy to explain to people, a decision tree model will always do better than a linear model. Decision tree models are even simpler to interpret than linear regression!

6. Working with Decision Trees in R and Python

For R users and Python users, decision tree is quite easy to implement. Let's quickly look at the set of codes which can get you started with this algorithm. For ease of use, I've shared standard codes where you'll need to replace your data set name and variables to get started.

For R users, there are multiple packages available to implement decision tree such as ctree, rpart, tree etc.

```
> library(rpart)
> x <- cbind(x_train,y_train)
# grow tree
> fit <- rpart(y_train ~ ., data = x,method="class")
> summary(fit)
#Predict Output
> predicted= predict(fit,x_test)
```

In the code above:

- y_train – represents dependent variable.
- x_train – represents independent variable
- x – represents training data.

For Python users, below is the code:

```
#Import Library
#Import other necessary libraries like pandas, numpy...
from sklearn import tree
#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predictor) of
test_dataset
# Create tree object
model = tree.DecisionTreeClassifier(criterion='gini') # for classification, here you can change the
algorithm as gini or entropy (information gain) by default it is gini
# model = tree.DecisionTreeRegressor() for regression
# Train the model using the training sets and check score
model.fit(X, y)
model.score(X, y)
#Predict Output
predicted= model.predict(x_test)
```

7. What are ensemble methods in tree based modeling ?

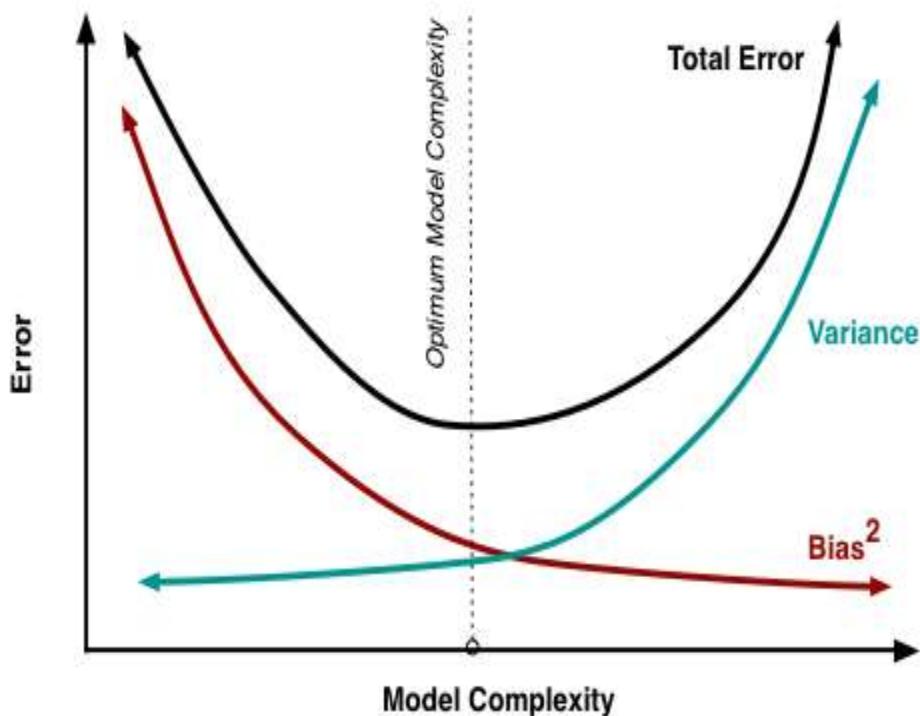
The literary meaning of word 'ensemble' is group. Ensemble methods involve group of predictive models to achieve a better accuracy and model stability. Ensemble methods are known to impart supreme boost to tree based models.

Like every other model, a tree based model also suffers from the plague of bias and variance. Bias means, 'how much on an average are the predicted values different from the actual value.' Variance means, 'how different will the predictions of the model be at the same point if different samples are taken from the same population'.

You build a small tree and you will get a model with low variance and high bias. How do you manage to balance the trade off between bias and variance ?

Normally, as you increase the complexity of your model, you will see a reduction in prediction error due to lower bias in the model. As you continue to make your model more complex, you end up over-fitting your model and your model will start suffering from high variance.

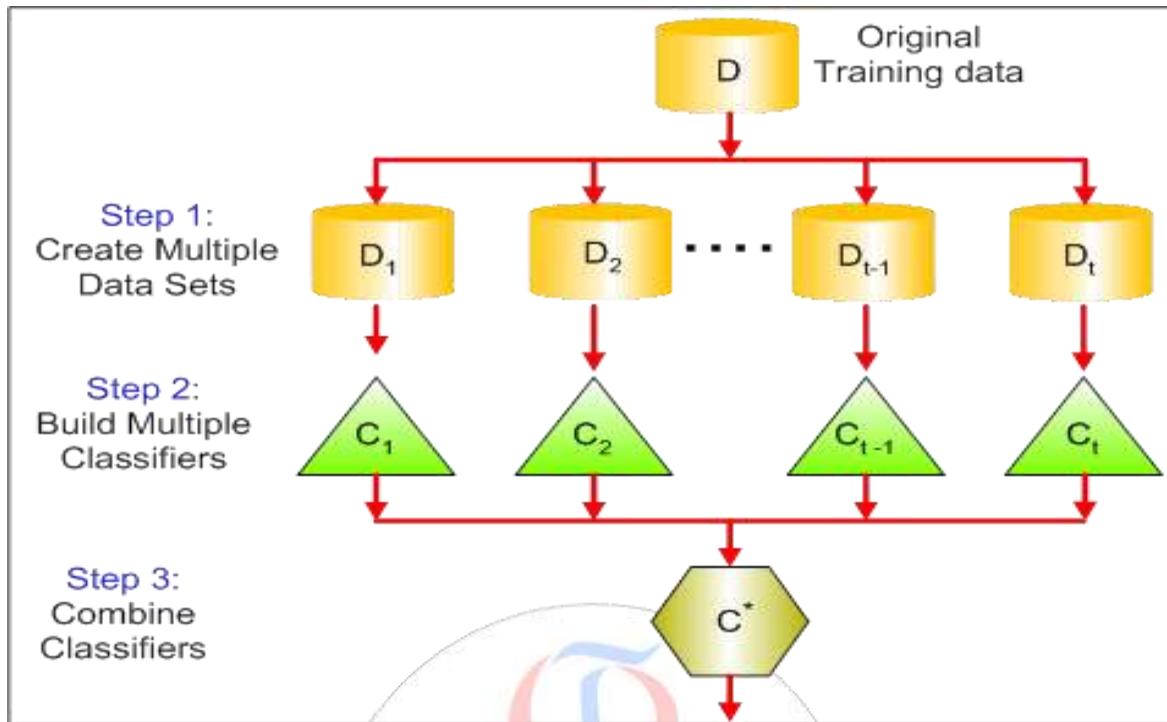
A champion model should maintain a balance between these two types of errors. This is known as the **trade-off management** of bias-variance errors. Ensemble learning is one way to execute this trade off analysis.



Some of the commonly used ensemble methods include: Bagging, Boosting and Stacking. In this tutorial, we'll focus on Bagging and Boosting in detail.

8. What is Bagging? How does it work?

Bagging is a technique used to reduce the variance of our predictions by combining the result of multiple classifiers modeled on different sub-samples of the same data set. The following figure will make it clearer:



The steps followed in bagging are:

- 1. Create Multiple DataSets:**
 - Sampling is done with replacement on the original data and new datasets are formed.
 - The new data sets can have a fraction of the columns as well as rows, which are generally hyper-parameters in a bagging model
 - Taking row and column fractions less than 1 helps in making robust models, less prone to overfitting
- 2. Build Multiple Classifiers:**
 - Classifiers are built on each data set.
 - Generally the same classifier is modeled on each data set and predictions are made.
- 3. Combine Classifiers:**
 - The predictions of all the classifiers are combined using a mean, median or mode value depending on the problem at hand.
 - The combined values are generally more robust than a single model.

Note that, here the number of models built is not a hyper-parameters. Higher number of models are always better or may give similar performance than lower numbers. It can be theoretically shown that the variance of the combined predictions are reduced to $1/n$ (n : number of classifiers) of the original variance, under some assumptions.

There are various implementations of bagging models. Random forest is one of them and we'll discuss it next.

9. What is Random Forest ? How does it work?

Random Forest is considered to be a panacea of all data science problems. On a funny note, when you can't think of any algorithm (irrespective of situation), use random forest!

Random Forest is a versatile machine learning method capable of performing both regression and classification tasks. It also undertakes dimensional reduction methods, treats missing values, outlier values and other essential steps of data exploration, and does a fairly good job. It is a type of ensemble learning method, where a group of weak models combine to form a powerful model.

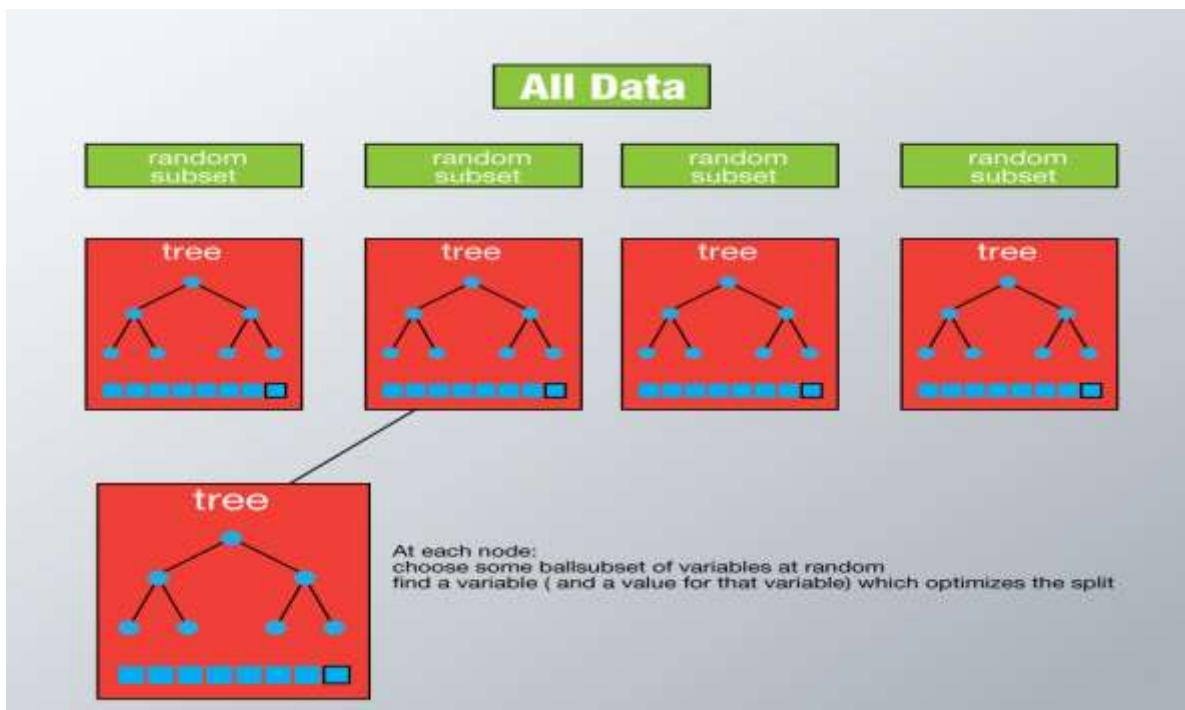
How does it work?

In Random Forest, we grow multiple trees as opposed to a single tree in CART model (see comparison between CART and Random Forest here, part1 and part2). To classify a new object based on attributes, each tree gives a classification and we say the tree "votes" for that class. The forest chooses the classification having the most votes (over all the trees in the forest) and in case of regression, it takes the average of outputs by different trees.



It works in the following manner. Each tree is planted & grown as follows:

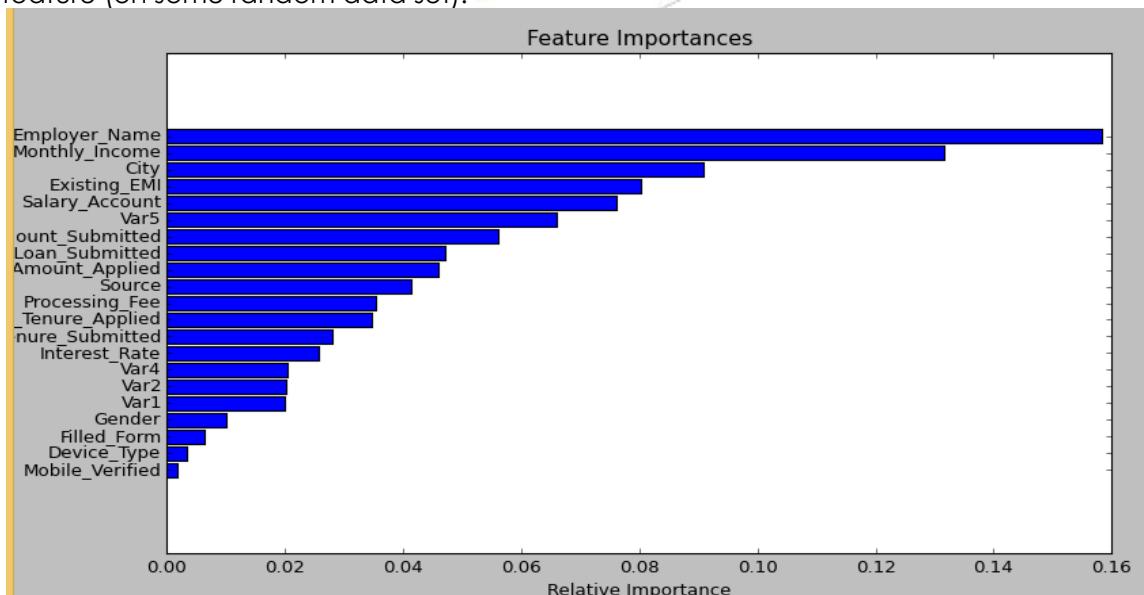
1. Assume number of cases in the training set is N. Then, sample of these N cases is taken at random but with replacement. This sample will be the training set for growing the tree.
2. If there are M input variables, a number $m < M$ is specified such that at each node, m variables are selected at random out of the M. The best split on these m is used to split the node. The value of m is held constant while we grow the forest.
3. Each tree is grown to the largest extent possible and there is no pruning.
4. Predict new data by aggregating the predictions of the ntree trees (i.e., majority votes for classification, average for regression).



To understand more in detail about this algorithm using a case study, please read this article "Introduction to Random forest – Simplified".

Advantages of Random Forest

- This algorithm can solve both type of problems i.e. classification and regression and does a decent estimation at both fronts.
- One of benefits of Random forest which excites me most is, the power of handle large data set with higher dimensionality. It can handle thousands of input variables and identify most significant variables so it is considered as one of the dimensionality reduction methods. Further, the model outputs **Importance of variable**, which can be a very handy feature (on some random data set).



- It has an effective method for estimating missing data and maintains accuracy when a large proportion of the data are missing.
- It has methods for balancing errors in data sets where classes are imbalanced.
- The capabilities of the above can be extended to unlabeled data, leading to unsupervised clustering, data views and outlier detection.
- Random Forest involves sampling of the input data with replacement called as bootstrap sampling. Here one third of the data is not used for training and can be used to testing. These are called the **out of bag** samples. Error estimated on these out of bag samples is known as out of bag error. Study of error estimates by Out of bag, gives evidence to show that the out-of-bag estimate is as accurate as using a test set of the same size as the training set. Therefore, using the out-of-bag error estimate removes the need for a set aside test set.

Disadvantages of Random Forest

- It surely does a good job at classification but not as good as for regression problem as it does not give precise continuous nature predictions. In case of regression, it doesn't predict beyond the range in the training data, and that they may over-fit data sets that are particularly noisy.
- Random Forest can feel like a black box approach for statistical modelers – you have very little control on what the model does. You can at best – try different parameters and random seeds!

Python & R implementation

Random forests have commonly known implementations in R packages and Python scikit-learn. Let's look at the code of loading random forest model in R and Python below:

Python

```
#Import Library
from sklearn.ensemble import RandomForestClassifier #use RandomForestRegressor for
regression problem
#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predictor) of
test_dataset
# Create Random Forest object
model= RandomForestClassifier(n_estimators=1000)
# Train the model using the training sets and check score
model.fit(X, y)
#Predict Output
predicted= model.predict(x_test)
```

R Code

```
> library(randomForest)
> x <- cbind(x_train,y_train)
# Fitting model
> fit <- randomForest(Species ~ ., x,ntree=500)
```

```
> summary(fit)
#Predict Output
> predicted= predict(fit,x_test)
```

10. What is Boosting ? How does it work?

Definition: The term 'Boosting' refers to a family of algorithms which converts weak learner to strong learners.

Let's understand this definition in detail by solving a problem of spam email identification:

How would you classify an email as SPAM or not? Like everyone else, our initial approach would be to identify 'spam' and 'not spam' emails using following criteria. If:

1. Email has only one image file (promotional image), It's a SPAM
2. Email has only link(s), It's a SPAM
3. Email body consist of sentence like "You won a prize money of \$ xxxxx", It's a SPAM
4. Email from our official domain "Analyticsvidhya.com" , Not a SPAM
5. Email from known source, Not a SPAM

Above, we've defined multiple rules to classify an email into 'spam' or 'not spam'. But, do you think these rules individually are strong enough to successfully classify an email? No.

Individually, these rules are not powerful enough to classify an email into 'spam' or 'not spam'. Therefore, these rules are called as **weak learner**.

To convert weak learner to strong learner, we'll combine the prediction of each weak learner using methods like:

- Using average/ weighted average
- Considering prediction has higher vote

For example: Above, we have defined 5 weak learners. Out of these 5, 3 are voted as 'SPAM' and 2 are voted as 'Not a SPAM'. In this case, by default, we'll consider an email as SPAM because we have higher(3) vote for 'SPAM'.

How does it work?

Now we know that, boosting combines weak learner a.k.a. base learner to form a strong rule. An immediate question which should pop in your mind is, 'How boosting identify weak rules?'

To find weak rule, we apply base learning (ML) algorithms with a different distribution. Each time base learning algorithm is applied, it generates a new weak prediction rule. This is an iterative process. After many iterations, the boosting algorithm combines these weak rules into a single strong prediction rule.

Here's another question which might haunt you, 'How do we choose different distribution for each round?'

For choosing the right distribution, here are the following steps:

Step 1: The base learner takes all the distributions and assign equal weight or attention to each observation.

Step 2: If there is any prediction error caused by first base learning algorithm, then we pay higher attention to observations having prediction error. Then, we apply the next base learning algorithm.

Step 3: Iterate Step 2 till the limit of base learning algorithm is reached or higher accuracy is achieved.

Finally, it combines the outputs from weak learner and creates a strong learner which eventually improves the prediction power of the model. Boosting pays higher focus on examples which are mis-classified or have higher errors by preceding weak rules.

There are many boosting algorithms which impart additional boost to model's accuracy. In this tutorial, we'll learn about the two most commonly used algorithms i.e. Gradient Boosting (GBM) and XGboost.

11. Which is more powerful: GBM or Xgboost?

I've always admired the boosting capabilities that xgboost algorithm. At times, I've found that it provides better result compared to GBM implementation, but at times you might find that the gains are just marginal. When I explored more about its performance and science behind its high accuracy, I discovered many advantages of Xgboost over GBM:

1. Regularization:
 - Standard GBM implementation has no regularization like XGBoost, therefore it also helps to reduce overfitting.
 - In fact, XGBoost is also known as 'regularized boosting' technique.
2. Parallel Processing:
 - XGBoost implements parallel processing and is blazingly faster as compared to GBM.
 - But hang on, we know that boosting is sequential process so how can it be parallelized? We know that each tree can be built only after the previous one, so what stops us from making a tree using all cores? I hope you get where I'm coming from. Check this link out to explore further.
 - XGBoost also supports implementation on Hadoop.
3. High Flexibility
 - XGBoost allow users to define custom optimization objectives and evaluation criteria.
 - This adds a whole new dimension to the model and there is no limit to what we can do.
4. Handling Missing Values
 - XGBoost has an in-built routine to handle missing values.
 - User is required to supply a different value than other observations and pass that as a parameter. XGBoost tries different things as it encounters a missing value on each node and learns which path to take for missing values in future.
5. Tree Pruning:
 - A GBM would stop splitting a node when it encounters a negative loss in the split. Thus it is more of a greedy algorithm.
 - XGBoost on the other hand make splits upto the max_depth specified and then start pruning the tree backwards and remove splits beyond which there is no positive gain.

- Another advantage is that sometimes a split of negative loss say -2 may be followed by a split of positive loss +10. GBM would stop as it encounters -2. But XGBoost will go deeper and it will see a combined effect of +8 of the split and keep both.
6. Built-in Cross-Validation
 - XGBoost allows user to run a cross-validation at each iteration of the boosting process and thus it is easy to get the exact optimum number of boosting iterations in a single run.
 - This is unlike GBM where we have to run a grid-search and only a limited values can be tested.
 7. Continue on Existing Model
 - User can start training an XGBoost model from its last iteration of previous run. This can be of significant advantage in certain specific applications.
 - GBM implementation of sklearn also has this feature so they are even on this point.

12. Working with GBM in R and Python

Before we start working, let's quickly understand the important parameters and the working of this algorithm. This will be helpful for both R and Python users. Below is the overall pseudo-code of GBM algorithm for 2 classes:

1. Initialize the outcome
2. Iterate from 1 to total number of trees
 - 2.1 Update the weights for targets based on previous run (higher for the ones mis-classified)
 - 2.2 Fit the model on selected subsample of data
 - 2.3 Make predictions on the full set of observations
 - 2.4 Update the output with current results taking into account the learning rate
3. Return the final output.

This is an extremely simplified (probably naive) explanation of GBM's working. But, it will help every beginners to understand this algorithm.

Let's consider the important GBM parameters used to improve model performance in Python:

1. learning_rate
 - This determines the impact of each tree on the final outcome (step 2.4). GBM works by starting with an initial estimate which is updated using the output of each tree. The learning parameter controls the magnitude of this change in the estimates.
 - Lower values are generally preferred as they make the model robust to the specific characteristics of tree and thus allowing it to generalize well.
 - Lower values would require higher number of trees to model all the relations and will be computationally expensive.
2. n_estimators
 - The number of sequential trees to be modeled (step 2)
 - Though GBM is fairly robust at higher number of trees but it can still overfit at a point. Hence, this should be tuned using CV for a particular learning rate.
3. subsample
 - The fraction of observations to be selected for each tree. Selection is done by random sampling.
 - Values slightly less than 1 make the model robust by reducing the variance.
 - Typical values ~0.8 generally work fine but can be fine-tuned further.

Apart from these, there are certain miscellaneous parameters which affect overall functionality:

1. loss
 - o It refers to the loss function to be minimized in each split.
 - o It can have various values for classification and regression case. Generally the default values work fine. Other values should be chosen only if you understand their impact on the model.
2. init
 - o This affects initialization of the output.
 - o This can be used if we have made another model whose outcome is to be used as the initial estimates for GBM.
3. random_state
 - o The random number seed so that same random numbers are generated every time.
 - o This is important for parameter tuning. If we don't fix the random number, then we'll have different outcomes for subsequent runs on the same parameters and it becomes difficult to compare models.
 - o It can potentially result in overfitting to a particular random sample selected. We can try running models for different random samples, which is computationally expensive and generally not used.
4. verbose
 - o The type of output to be printed when the model fits. The different values can be:
 - 0: no output generated (default)
 - 1: output generated for trees in certain intervals
 - >1: output generated for all trees
5. warm_start
 - o This parameter has an interesting application and can help a lot if used judiciously.
 - o Using this, we can fit additional trees on previous fits of a model. It can save a lot of time and you should explore this option for advanced applications
6. presort
 - o Select whether to presort data for faster splits.
 - o It makes the selection automatically by default but it can be changed if needed.

I know its a long list of parameters but I have simplified it for you in an excel file which you can download from this GitHub repository.

For R users, using caret package, there are 3 main tuning parameters:

1. n.trees – It refers to number of iterations i.e. tree which will be taken to grow the trees
2. interaction.depth – It determines the complexity of the tree i.e. total number of splits it has to perform on a tree (starting from a single node)
3. shrinkage – It refers to the learning rate. This is similar to learning_rate in python (shown above).
4. n.minobsinnode – It refers to minimum number of training samples required in a node to perform splitting

GBM in R (with cross validation)

I've shared the standard codes in R and Python. At your end, you'll be required to change the value of dependent variable and data set name used in the codes below. Considering the ease of implementing GBM in R, one can easily perform tasks like cross validation and grid search with this package.

```
> library(caret)
```

```
> fitControl <- trainControl(method = "cv",
    number = 10, #5folds)
> tune_Grid <- expand.grid(interaction.depth = 2,
    n.trees = 500,
    shrinkage = 0.1,
    n.minobsinnode = 10)
> set.seed(825)
> fit <- train(y_train ~ ., data = train,
    method = "gbm",
    trControl = fitControl,
    verbose = FALSE,
    tuneGrid = gbmGrid)
> predicted = predict(fit, test, type = "prob")[,2]
```

GBM in Python

```
#import libraries
from sklearn.ensemble import GradientBoostingClassifier #For Classification
from sklearn.ensemble import GradientBoostingRegressor #For Regression
#use GBM function
clf = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0, max_depth=1)
clf.fit(X_train, y_train)
```

Clustering

Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group than those in other groups. In simple words, the aim is to segregate groups with similar traits and assign them into clusters.

Let's understand this with an example. Suppose, you are the head of a rental store and wish to understand preferences of your costumers to scale up your business. Is it possible for you to look at details of each costumer and devise a unique business strategy for each one of them? Definitely not. But, what you can do is to cluster all of your costumers into say 10 groups based on their purchasing habits and use a separate strategy for costumers in each of these 10 groups. And this is what we call clustering.

Now, that we understand what is clustering. Let's take a look at the types of clustering.

Types of Clustering

Broadly speaking, clustering can be divided into two subgroups :

- **Hard Clustering:** In hard clustering, each data point either belongs to a cluster completely or not. For example, in the above example each customer is put into one group out of the 10 groups.

- **Soft Clustering:** In soft clustering, instead of putting each data point into a separate cluster, a probability or likelihood of that data point to be in those clusters is assigned. For example, from the above scenario each customer is assigned a probability to be in either of 10 clusters of the retail store.

Types of clustering algorithms

Since the task of clustering is subjective, the means that can be used for achieving this goal are plenty. Every methodology follows a different set of rules for defining the 'similarity' among data points. In fact, there are more than 100 clustering algorithms known. But few of the algorithms are used popularly, let's look at them in detail:

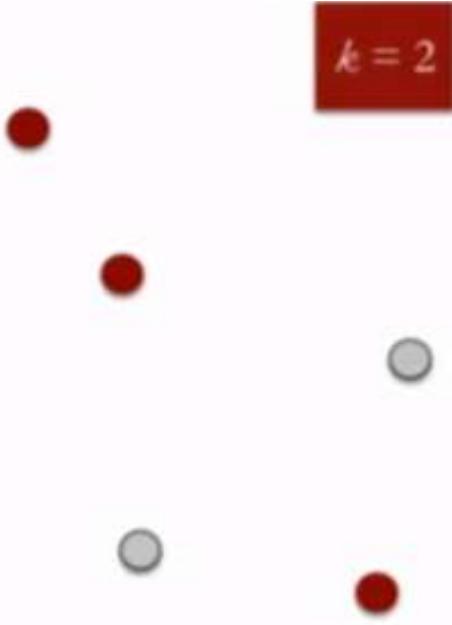
- **Connectivity models:** As the name suggests, these models are based on the notion that the data points closer in data space exhibit more similarity to each other than the data points lying farther away. These models can follow two approaches. In the first approach, they start with classifying all data points into separate clusters & then aggregating them as the distance decreases. In the second approach, all data points are classified as a single cluster and then partitioned as the distance increases. Also, the choice of distance function is subjective. These models are very easy to interpret but lacks scalability for handling big datasets. Examples of these models are hierarchical clustering algorithm and its variants.
- **Centroid models:** These are iterative clustering algorithms in which the notion of similarity is derived by the closeness of a data point to the centroid of the clusters. K-Means clustering algorithm is a popular algorithm that falls into this category. In these models, the no. of clusters required at the end have to be mentioned beforehand, which makes it important to have prior knowledge of the dataset. These models run iteratively to find the local optima.
- **Distribution models:** These clustering models are based on the notion of how probable is it that all data points in the cluster belong to the same distribution (For example: Normal, Gaussian). These models often suffer from overfitting. A popular example of these models is Expectation-maximization algorithm which uses multivariate normal distributions.
- **Density Models:** These models search the data space for areas of varied density of data points in the data space. It isolates various different density regions and assign the data points within these regions in the same cluster. Popular examples of density models are DBSCAN and OPTICS.

Now I will be taking you through two of the most popular clustering algorithms in detail – K Means clustering and Hierarchical clustering. Let's begin.

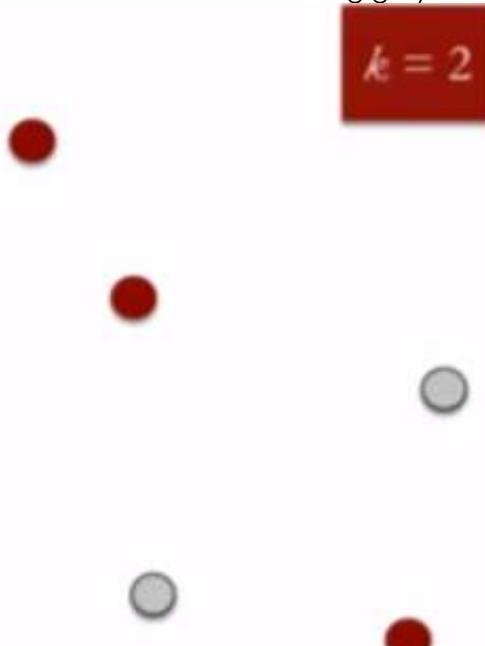
K Means Clustering

K means is an iterative clustering algorithm that aims to find local maxima in each iteration. This algorithm works in these 5 steps :

1. Specify the desired number of clusters K : Let us choose k=2 for these 5 data points in 2-D space.

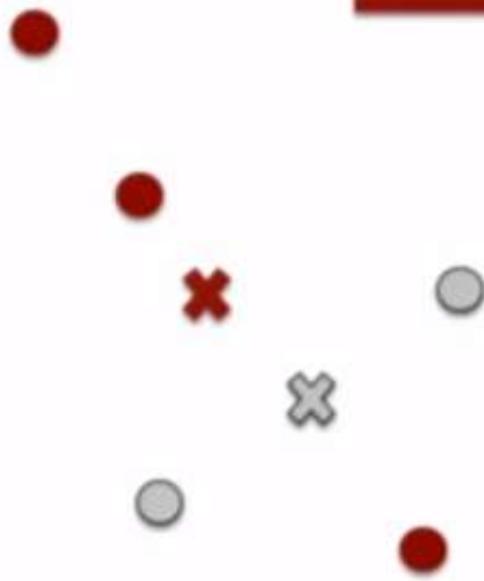


2. Randomly assign each data point to a cluster : Let's assign three points in cluster 1 shown using red color and two points in cluster 2 shown using grey color.



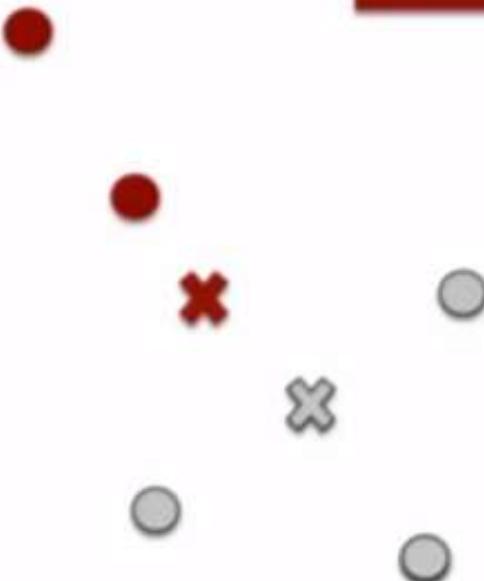
3. Compute cluster centroids : The centroid of data points in the red cluster is shown using red cross and those in grey cluster using grey cross.

$k = 2$

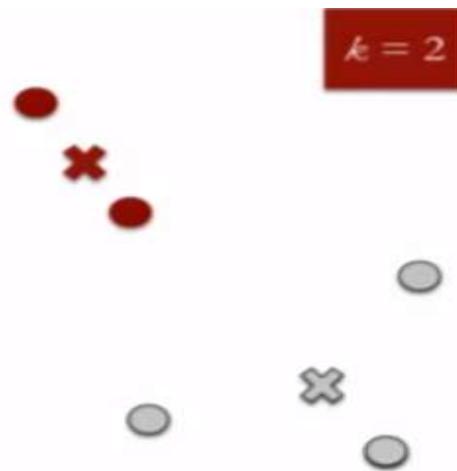


4. Re-assign each point to the closest cluster centroid : Note that only the data point at the bottom is assigned to the red cluster even though its closer to the centroid of grey cluster. Thus, we assign that data point into grey cluster

$k = 2$



5. Re-compute cluster centroids : Now, re-computing the centroids for both the clusters.

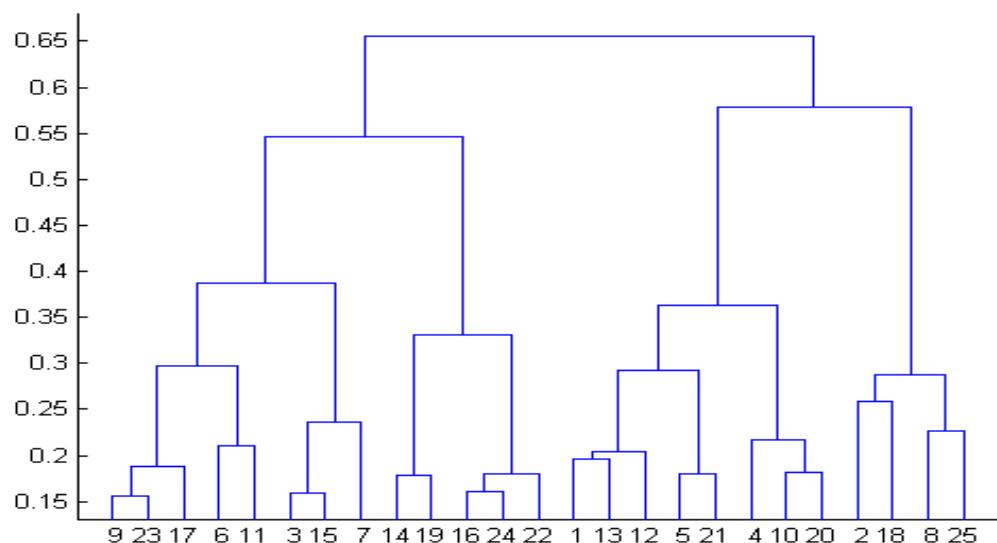


6. Repeat steps 4 and 5 until no improvements are possible : Similarly, we'll repeat the 4th and 5th steps until we'll reach global optima. When there will be no further switching of data points between two clusters for two successive repeats. It will mark the termination of the algorithm if not explicitly mentioned.

Hierarchical Clustering

Hierarchical clustering, as the name suggests is an algorithm that builds hierarchy of clusters. This algorithm starts with all the data points assigned to a cluster of their own. Then two nearest clusters are merged into the same cluster. In the end, this algorithm terminates when there is only a single cluster left.

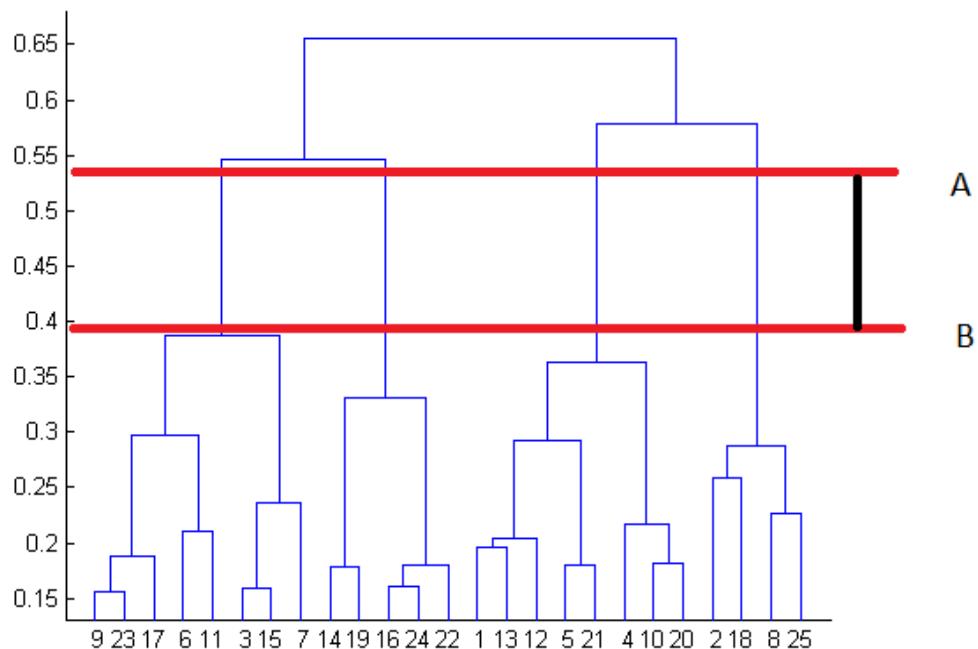
The results of hierarchical clustering can be shown using dendrogram. The dendrogram can be interpreted as:



At the bottom, we start with 25 data points, each assigned to separate clusters. Two closest clusters are then merged till we have just one cluster at the top. The height in the dendrogram at which two clusters are merged represents the distance between two clusters in the data space.

The decision of the no. of clusters that can best depict different groups can be chosen by observing the dendrogram. The best choice of the no. of clusters is the no. of vertical lines in the dendrogram cut by a horizontal line that can transverse the maximum distance vertically without intersecting a cluster.

In the above example, the best choice of no. of clusters will be 4 as the red horizontal line in the dendrogram below covers maximum vertical distance AB.



Two important things that you should know about hierarchical clustering are:

- This algorithm has been implemented above using bottom up approach. It is also possible to follow top-down approach starting with all data points assigned in the same cluster and recursively performing splits till each data point is assigned a separate cluster.
- The decision of merging two clusters is taken on the basis of closeness of these clusters. There are multiple metrics for deciding the closeness of two clusters :
 - Euclidean distance: $\|a-b\|_2 = \sqrt{\sum (a_i - b_i)^2}$
 - Squared Euclidean distance: $\|a-b\|_2^2 = \sum (a_i - b_i)^2$
 - Manhattan distance: $\|a-b\|_1 = \sum |a_i - b_i|$
 - Maximum distance: $\|a-b\|_{\infty} = \max_i |a_i - b_i|$
 - Mahalanobis distance: $\sqrt{(a-b)^T S^{-1} (-b)}$ {where, s : covariance matrix}

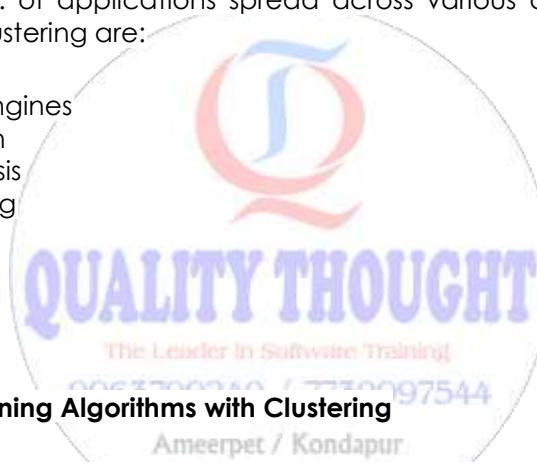
Difference between K Means and Hierarchical clustering

- Hierarchical clustering can't handle big data well but K Means clustering can. This is because the time complexity of K Means is linear i.e. $O(n)$ while that of hierarchical clustering is quadratic i.e. $O(n^2)$.
- In K Means clustering, since we start with random choice of clusters, the results produced by running the algorithm multiple times might differ. While results are reproducible in Hierarchical clustering.
- K Means is found to work well when the shape of the clusters is hyper spherical (like circle in 2D, sphere in 3D).
- K Means clustering requires prior knowledge of K i.e. no. of clusters you want to divide your data into. But, you can stop at whatever number of clusters you find appropriate in hierarchical clustering by interpreting the dendrogram

Applications of Clustering

Clustering has a large no. of applications spread across various domains. Some of the most popular applications of clustering are:

- Recommendation engines
- Market segmentation
- Social network analysis
- Search result grouping
- Medical imaging
- Image segmentation
- Anomaly detection

**Improving Supervised Learning Algorithms with Clustering**

Clustering is an unsupervised machine learning approach, but can it be used to improve the accuracy of supervised machine learning algorithms as well by clustering the data points into similar groups and using these cluster labels as independent variables in the supervised machine learning algorithm? Let's find out.

Let's check out the impact of clustering on the accuracy of our model for the classification problem using 3000 observations with 100 predictors of stock data to predicting whether the stock will go up or down using R. This dataset contains 100 independent variables from X1 to X100 representing profile of a stock and one outcome variable Y with two levels : 1 for rise in stock price and -1 for drop in stock price.

The dataset is available online

Let's first try applying randomforest without clustering.

```
#loading required libraries
library('randomForest')
library('Metrics')
```

```
#set random seed
set.seed(101)
#loading dataset
data<-read.csv("train.csv",stringsAsFactors= T)
#checking dimensions of data
dim(data)
## [1] 3000 101
#specifying outcome variable as factor
data$Y<-as.factor(data$Y)
#dividing the dataset into train and test
train<-data[1:2000,]
test<-data[2001:3000,]
#applying randomForest
model_rf<-randomForest(Y~.,data=train)
preds<-predict(object=model_rf,test[,-101])
table(preds)
## preds
## -1 1
## 453 547
#checking accuracy
auc(preds,test$Y)
## [1] 0.4522703
```

So, the accuracy we get is 0.45. Now let's create five clusters based on values of independent variables using k-means clustering and reapply randomforest.

```
#combing test and train
all<-rbind(train,test)
#creating 5 clusters using K- means clustering
Cluster <- kmeans(all[,-101], 5)
#adding clusters as independent variable to the dataset.
all$cluster<-as.factor(Cluster$cluster)
#dividing the dataset into train and test
train<-all[1:2000,]
test<-all[2001:3000,]
#applying randomforest
model_rf<-randomForest(Y~.,data=train)
preds2<-predict(object=model_rf,test[,-101])
table(preds2)
## preds2
## -1 1
## 548 452
auc(preds2,test$Y)
## [1] 0.5345908
```

Whoo! In the above example, even though the final accuracy is poor but clustering has given our model a significant boost from accuracy of 0.45 to slightly above 0.53.

This shows that clustering can indeed be helpful for supervised machine learning tasks.

Association Rule Learning

We live in a fast changing digital world. In today's age customers expect the sellers to tell what they might want to buy. I personally end up using Amazon's recommendations almost in all my visits to their site.

This creates an interesting threat / opportunity situation for the retailers.

If you can tell the customers what they might want to buy – it not only improves your sales, but also the customer experience and ultimately life time value.

On the other hand, if you are unable to predict the next purchase, the customer might not come back to your store.

In this article, we will learn one such algorithm which enables us to predict the items bought together frequently. Once we know this, we can use it to our advantage in multiple ways.

The Approach(Apriori Algorithm)

When you go to a store, would you not want the aisles to be ordered in such a manner that reduces your efforts to buy things?

For example, I would want the toothbrush, the paste, the mouthwash & other dental products on a single aisle – because when I buy, I tend to buy them together. This is done by a way in which we find associations between items.

In order to understand the concept better, let's take a simple dataset (let's name it as Coffee dataset) consisting of a few hypothetical transactions. We will try to understand this in simple English.

The Coffee dataset consisting of items purchased from a retail store.

Coffee dataset:

Transaction	Item 1	Item 2	Item 3
1	Milk	Sugar	Coffee Powder
2	Milk	Sugar	Coffee Powder
3	Milk	Sugar	Coffee Powder
4	Milk	Sugar	
5	Milk	Sugar	

The Association Rules:

For this dataset, we can write the following association rules: (Rules are just for illustrations and understanding of the concept. They might not represent the actuals).

Rule 1: If Milk is purchased, then Sugar is also purchased.

Rule 2: If Sugar is purchased, then Milk is also purchased.

Rule 3: If Milk and Sugar are purchased, Then Coffee powder is also purchased in 60% of the transactions.

Generally, association rules are written in "IF-THEN" format. We can also use the term "Antecedent" for IF (LHS) and "Consequent" for THEN (RHS).

From the above rules, we understand the following explicitly:

1. Whenever Milk is purchased, Sugar is also purchased or vice versa.
2. If Milk and Sugar are purchased then the coffee powder is also purchased. This is true in 3 out of the 5 transactions.

For example, if we see {Milk} as a set with one item and {Coffee} as another set with one item, we will use these to find sets with two items in the dataset such as {Milk,Coffee} and then later see which products are purchased with both of these in our basket.

Therefore now we will search for a suitable right hand side or Consequent. If someone buys Coffee with Milk, we will represent it as {Coffee} => {Milk} where Coffee becomes the LHS and Milk the RHS.

When we use these to explore more k-item sets, we might find that {Coffee,Milk} => {Tea}. That means the people who buy Coffee and Milk have a possibility of buying Tea as well.

Let us see how the item sets are actually built using the Apriori.

LHS	RHS	Count
Milk	-	300
Coffee	-	200
Tea	-	200
Sugar	-	150
Milk	Coffee	100
Tea	Sugar	80
Milk, Coffee	Tea	40
Milk, Coffee, Tea	Sugar	10

Apriori envisions an iterative approach where it uses k-Item sets to search for (k+1)-Item sets. The first 1-Item sets are found by gathering the count of each item in the set. Then the 1-Item sets are used to find 2-Item sets and so on **until no more k-Item sets can be explored; when all our items land up in one final observation as visible in our last row of the table above**. One exploration takes one scan of the complete dataset.

An Item set is a mathematical set of products in the basket.

1.1 Handling and Readyng The Dataset

The first part of any analysis is to bring in the dataset. We will be using an inbuilt dataset "Groceries" from the 'arules' package to simplify our analysis.

All stores and retailers store their information of transactions in a specific type of dataset called the "Transaction" type dataset.

The 'pacman' package is an assistor to help load and install the packages. we will be using pacman to load the arules package.

The p_load() function from "pacman" takes names of packages as arguments.

If your system has those packages, it will load them and if not, it will install and load them.

Example:

```
pacman::p_load(PACKAGE_NAME)
pacman::p_load(arules, arulesViz)
```

OR

```
Library(arules)
```

```
Library(arulesViz)
```

```
data("Groceries")
```

1.2 Structural Overview and Prerequisites

Before we begin applying the "Apriori" algorithm on our dataset, we need to make sure that it is of the type "Transactions".

```
str(Groceries)
```

```

Formal class 'transactions' [package "arules"] with 3 slots
..@ data      :Formal class 'ngCMatrix' [package "Matrix"] with 5 slots
... ... @ i    : int [1:43367] 13 60 69 78 14 29 98 24 15 29 ...
... ... @ p    : int [1:9836] 0 4 7 8 12 16 21 22 27 28 ...
... ... @ Dim   : int [1:2] 169 9835
... ... @ Dimnames:List of 2
... ... ... $ : NULL
... ... ... $ : NULL
... ... @ factors : list()
..@ itemInfo  :'data.frame': 169 obs. of  3 variables:
... ... $ labels: chr [1:169] "frankfurter" "sausage" "liver loaf" "ham" ...
... ... $ level2: Factor w/ 55 levels "baby food", "bags", ..: 44 44 44 44 44 44 42 42 41 ...
... ... $ level1: Factor w/ 10 levels "canned food", ..: 6 6 6 6 6 6 6 6 6 ...
..@ itemsetInfo:'data.frame': 0 obs. of  0 variables

```

The structure of our transaction type dataset shows us that it is internally divided into three slots: Data, itemInfo and itemsetInfo.

The slot "Data" contains the dimensions, dimension names and other numerical values of number of products sold by every transaction made.

```

> head(Groceries@itemInfo, n=12)
      labels    level2      level1
1  frankfurter sausage meat and sausage
2          sausage sausage meat and sausage
3    liver loaf sausage meat and sausage
4          ham sausage meat and sausage
5          meat sausage meat and sausage
6  finished products sausage meat and sausage
7    organic sausage sausage meat and sausage
8        chicken poultry meat and sausage
9        turkey poultry meat and sausage
10         pork    pork meat and sausage
11         beef    beef meat and sausage
12   hamburger meat    beef meat and sausage

```

These are the first 12 rows of the itemInfo list within the Groceries dataset. It gives specific names to our items under the column "labels". The "level2" column segregates into an easier to understand term, while "level1" makes the complete generalisation of Meat.

The slot itemInfo contains a Data Frame that has three vectors which categorizes the food items in the first vector "Labels".

The second & third vectors divide the food broadly into levels like "baby food", "bags" etc.

The third slot itemsetInfo will be generated by us and will store all associations.

```

> inspect(data)
  items
[1] {a,b,c}
[2] {a,b}
[3] {a,b,d}
[4] {b,e}
[5] {b,c,e}
[6] {a,d,e}
[7] {a,c}
[8] {a,b,d}
[9] {c,e}
[10] {a,b,d,e}
[11] {a,b,c,e}
> inspect(tl)
  items  transactionIDs
1 a      {1,2,3,6,7,8,10,11}
2 b      {1,2,3,4,5,8,10,11}
3 c      {1,5,7,9,11}
4 d      {3,6,8,10}
5 e      {4,5,6,9,10,11}
>

```

This is what the internal visual of any transaction dataset looks like and there is a dataframe containing products bought in each transaction in our first inspection. Then, we can group those products by TransactionID like we did in our second inspection to see how many times each is sold before we begin with associativity analysis.

The above datasets are just for a clearer visualisation on how to make a Transaction Dataset and can be reproduced using the following code:

```

data <- list(
  c("a","b","c"),
  c("a","b"),
  c("a","b","d"))

```

```
c("b","e"),  
c("b","c","e"),  
c("a","d","e"),  
c("a","c"),  
c("a","b","d"),  
c("c","e"),  
c("a","b","d","e"),  
c("a",'b','e','c')  
)
```

```
data <- as(data, "transactions")  
inspect(data)  
#Convert transactions to transaction ID lists  
tl <- as(data, "tidLists")  
inspect(tl)
```

Let us check the most frequently purchased products using the summary function.

```
summary(Groceries)
```



transactions as itemMatrix in sparse format with
9835 rows (elements/itemsets/transactions) and
169 columns (items) and a density of 0.02609146

most frequent items:

whole milk	other vegetables	rolls/buns	soda	yogurt
2513	1903	1809	1715	1372
(Other)				
34055				

element (itemset/transaction) length distribution:

sizes

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
2159	1643	1299	1005	855	645	545	438	350	246	182	117	78	77	55	46	29	14	14	9
21	22	23	24	26	27	28	29	32											
11	4	6	1	1	1	1	3	1											

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
1.000	2.000	3.000	4.409	6.000	32.000

includes extended item information - examples:

labels	level2	level1
1	frankfurter	sausage meat and sausage
2	sausage	sausage meat and sausage
3	liver loaf	sausage meat and sausage

The summary statistics show us the top 5 items sold in our transaction set as "Whole Milk","Other Vegetables","Rolls/Buns","Soda" and "Yogurt". (Further explained in Section 3)

To parse to Transaction type, make sure your dataset has similar slots and then use the as() function in R.

2. Implementing Apriori Algorithm and Key Terms and Usage

rules <- apriori(Groceries,

parameter = list(supp = 0.001, conf = 0.80))

We will set minimum support parameter (minSup) to .001.

We can set minimum confidence (minConf) to anywhere between 0.75 and 0.85 for varied results.

I have used support and confidence in my parameter list. Let me try to explain it:

Support: Support is the basic probability of an event to occur. If we have an event to buy product A, Support(A) is the number of transactions which includes A divided by total number of transactions.

Confidence: The confidence of an event is the conditional probability of the occurrence; the chances of A happening given B has already happened.

Lift: This is the ratio of confidence to expected confidence. The probability of all of the items in a rule occurring together (otherwise known as the support) divided by the product of the

probabilities of the items on the left and right side occurring as if there was no association between them.

The lift value tells us how much better a rule is at predicting something than randomly guessing. The higher the lift, the stronger the association.

Let's find out the top 10 rules arranged by lift.

```
inspect(rules[1:10])
```

lhs	rhs	support	confidence
[1] {liquor,red/blush wine}	=> {bottled beer}	0.001931876	0.9047619
[2] {curd,cereals}	=> {whole milk}	0.001016777	0.9090909
[3] {yogurt,cereals}	=> {whole milk}	0.001728521	0.8095238
[4] {butter,jam}	=> {whole milk}	0.001016777	0.8333333
[5] {soups,bottled beer}	=> {whole milk}	0.001118454	0.9166667
[6] {napkins,house keeping products}	=> {whole milk}	0.001321810	0.8125000
[7] {whipped/sour cream,house keeping products}	=> {whole milk}	0.001220132	0.9230769
[8] {pastry,sweet spreads}	=> {whole milk}	0.001016777	0.9090909
[9] {turkey,curd}	=> {other vegetables}	0.001220132	0.8000000
[10] {rice,sugar}	=> {whole milk}	0.001220132	1.0000000
lift			
[1] 11.235269			
[2] 3.557863			
[3] 3.168192			
[4] 3.261374			
[5] 3.587512			
[6] 3.179840			
[7] 3.612599			
[8] 3.557863			
[9] 4.134524			
[10] 3.913649			

Ameerpet / Kondapur

As we can see, these are the top 10 rules derived from our Groceries dataset by running the above code.

The first rule shows that if we buy Liquor and Red Wine, we are very likely to buy bottled beer. We can rank the rules based on top 10 from either lift, support or confidence.

Let's plot all our rules in certain visualisations first to see what goes with what item in our shop.

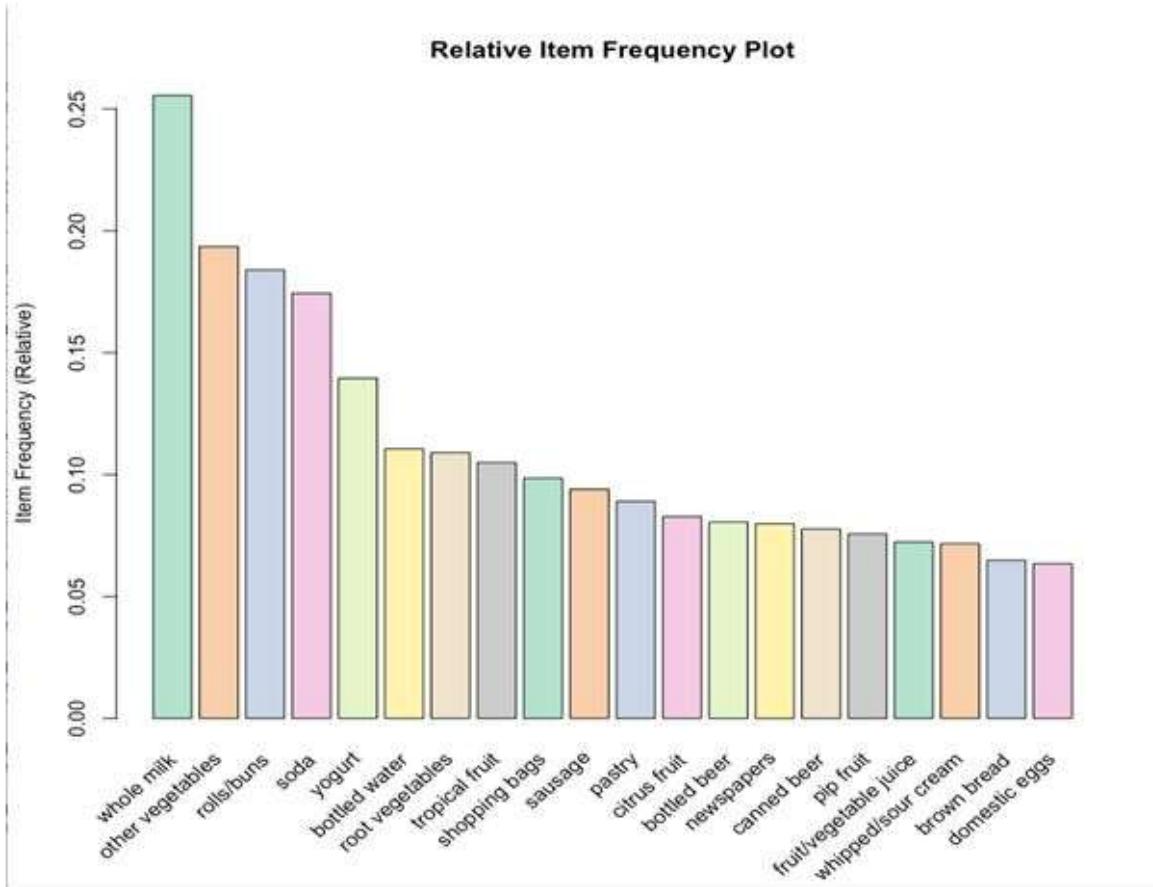
3. Interpretations and Analysis

Let us first identify which products were sold how frequently in our dataset.

3.1 The Item Frequency Histogram

These histograms depict how many times an item has occurred in our dataset as compared to the others.

The relative frequency plot accounts for the fact that "Whole Milk" and "Other Vegetables" constitute around half of the transaction dataset; half the sales of the store are of these items.



```
arules::itemFrequencyPlot(Groceries,topN=20,col=brewer.pal(8,'Pastel2'),main='Relative Item Frequency Plot',type="relative",ylab="Item Frequency (Relative)")
```

This would mean that a lot of people are buying milk and vegetables!

What other objects can we place around the more frequently purchased objects to enhance those sales too?

For example, to boost sales of eggs I can place it beside my milk and vegetables.

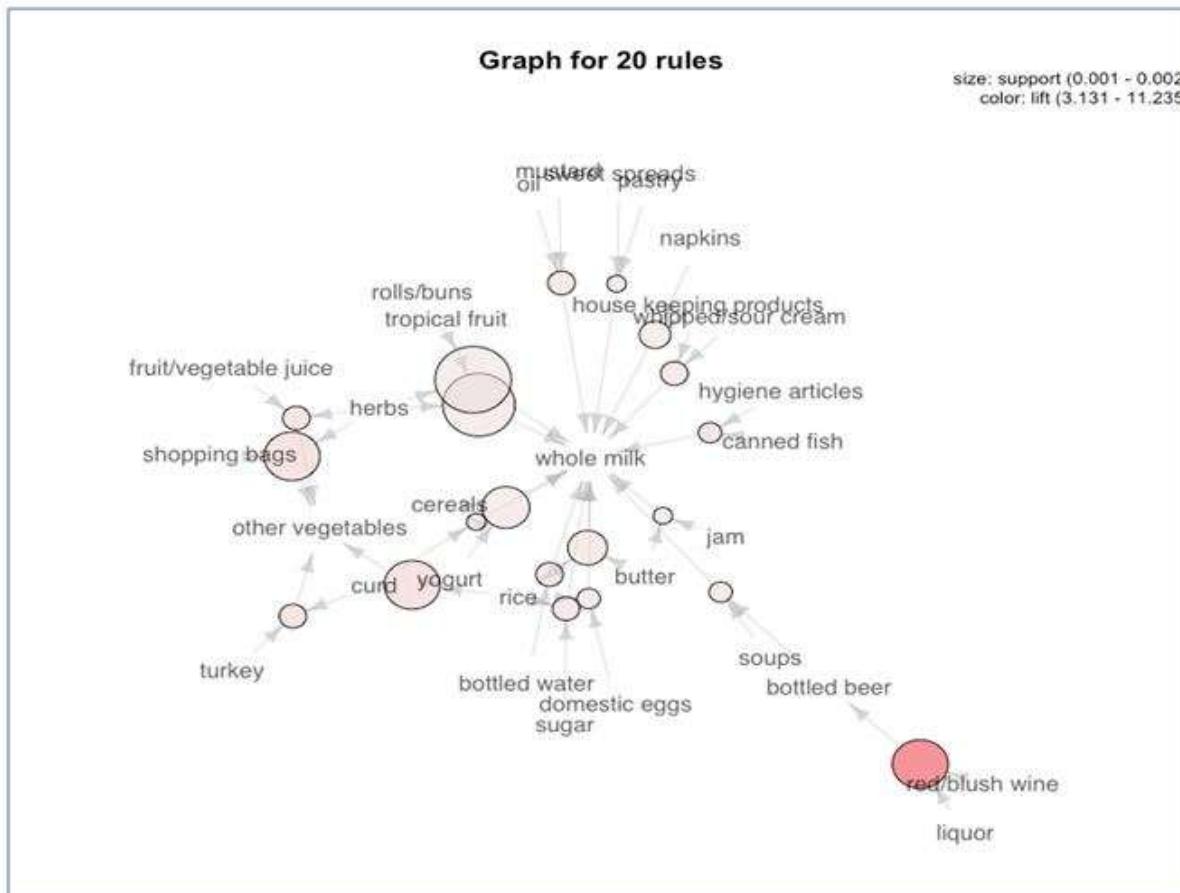
3.2 Graphical Representation

Moving forward in the visualisation, we can use a graph to highlight the support and lifts of various items in our repository but mostly to see which product is associated with which one in the sales environment.

```
plot(rules[1:20],  
      method = "graph",  
      control = list(type = "items"))
```

This representation gives us a graph model of items in our dataset.

The size of graph nodes is based on support levels and the colour on lift ratios. The incoming lines show the Antecedants or the LHS and the RHS is represented by names of items.



The above graph shows us that most of our transactions were consolidated around "Whole Milk".

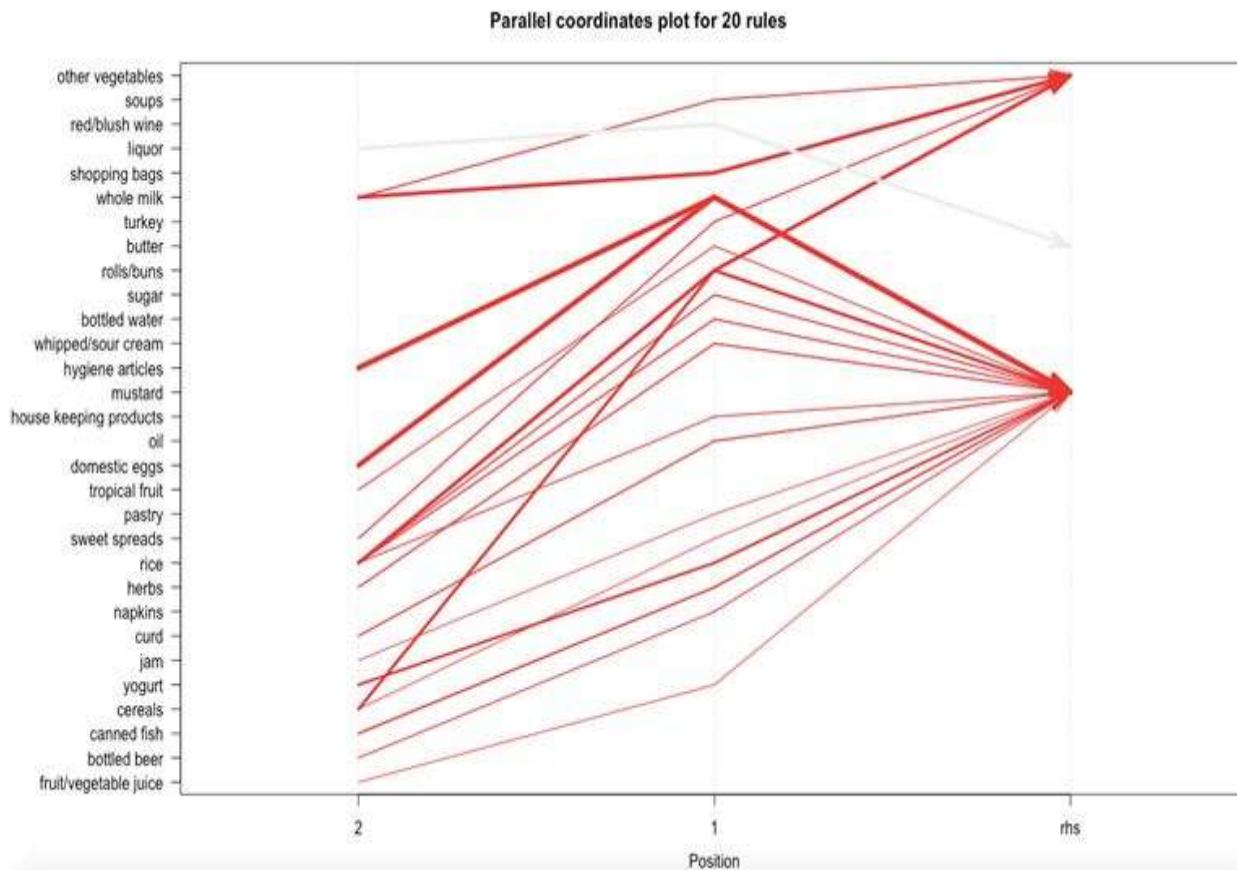
We also see that all liquor and wine are very strongly associated so we must place these together.

Another association we see from this graph is that the people who buy tropical fruits and herbs also buy rolls and buns. We should place these in an aisle together.

3.3 Individual Rule Representation

The next plot offers us a parallel coordinate system of visualisation. It would help us clearly see that which products along with which ones, result in what kinds of sales.

As mentioned above, the RHS is the Consequent or the item we propose the customer will buy; the positions are in the LHS where 2 is the most recent addition to our basket and 1 is the item we previously had.



The topmost rule shows us that when I have whole milk and soups in my shopping cart, I am highly likely to buy other vegetables to go along with those as well.

```
plot(rules[1:20],  
method = "paracoord",  
control = list(reorder = TRUE))
```

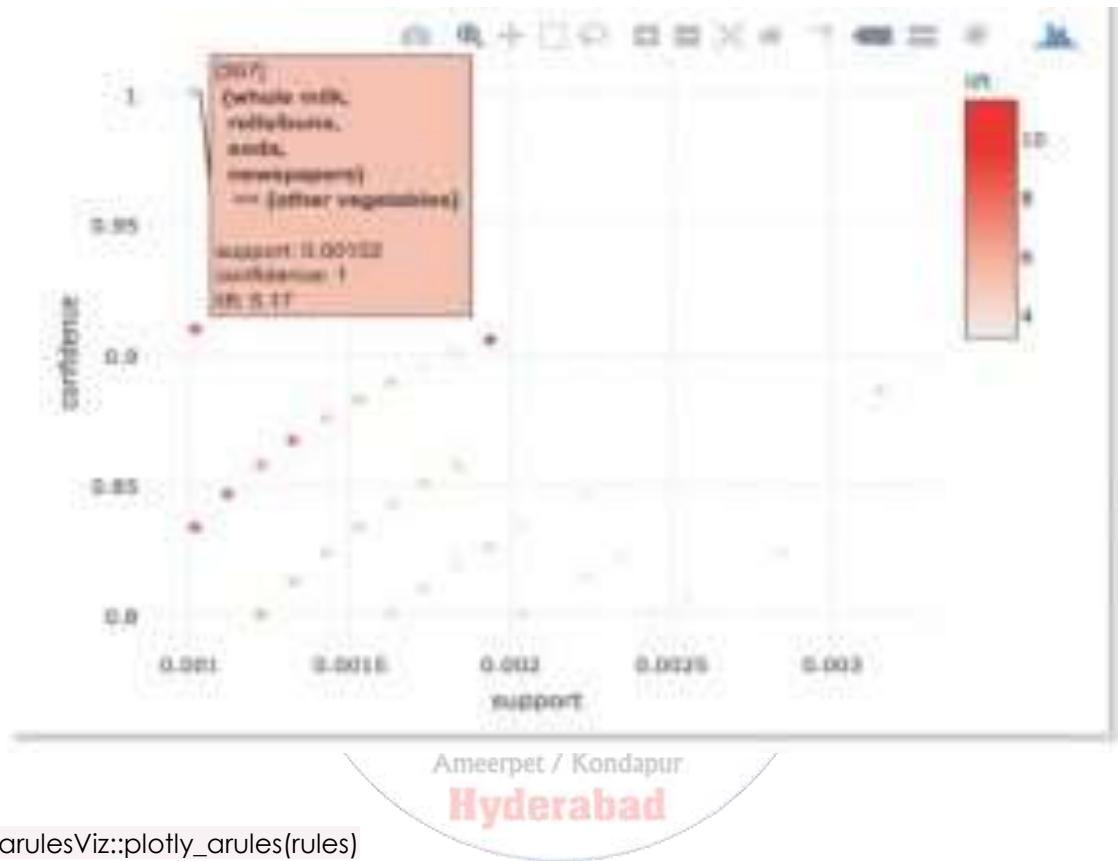
If we want a matrix representation, an alternate code option would be:

```
plot(rules[1:20],  
method = "matrix",
```

```
control = list(reorder = TRUE)
```

3.4 Interactive Scatterplot

These plots show us each and every rule visualised into a form of a scatterplot. The confidence levels are plotted on the Y axis and Support levels on the X axis for each rule. We can hover over them in our interactive plot to see the rule.



```
Plot: arulesViz::plotly_arules(rules)
```

The plot uses the arulesViz package and plotly to generate an interactive plot. We can hover over each rule and see the Support, Confidence and Lift.

As the interactive plot suggests, one rule that has a confidence of 1 is the one above. It has an exceptionally high lift as well, at 5.17.

Reinforcement Learning

What is Reinforcement learning?

Let us start with a simple analogy. If you have a pet at home, you may have used this technique with your pet.



A clicker (or whistle) is a technique to let your pet know some treat is just about to get served! This is essentially “reinforcing” your pet to practice good behavior. You click the “clicker” and follow up with a treat. And with time, your pet gets accustomed to this sound and responds every time he/she hears the click sound. With this technique, you can train your pet to do “good” deeds when required.

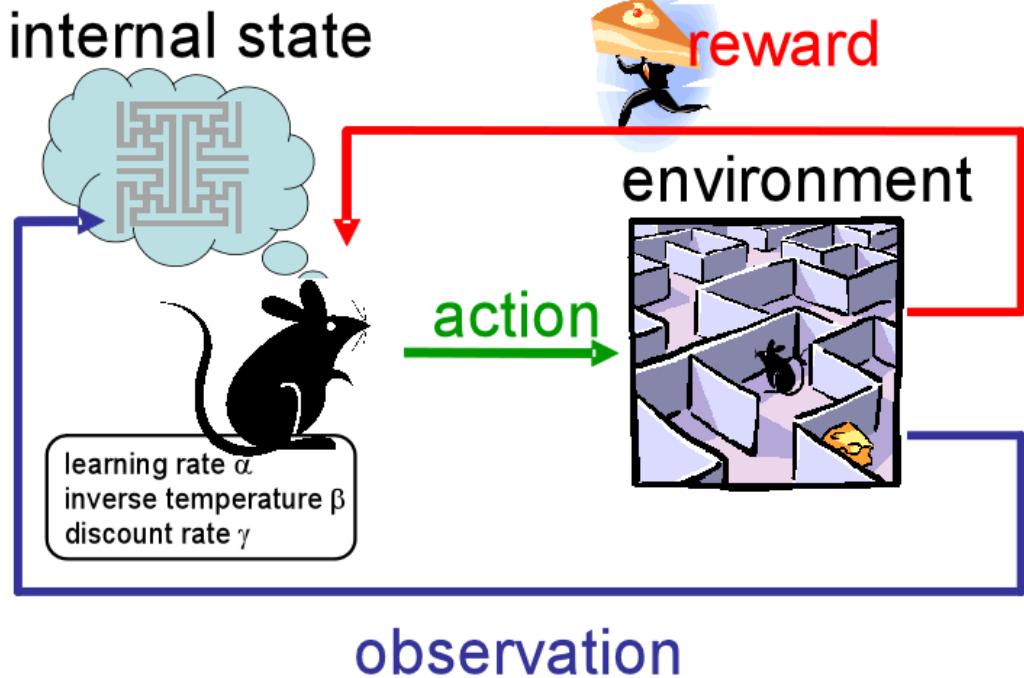
Now let's make these replacements in the example:

- The pet becomes the artificial agent
- The treat becomes the reward function
- The good behavior is the resultant action

The above example explains what reinforcement learning looks like. This is actually a classic example of reinforcement learning.

To apply this on an artificial agent, you have a kind of a feedback loop to reinforce your agent. It rewards when the actions performed is right and punishes in-case it was wrong. Basically what you have in your kitty is:

- an **internal state**, which is maintained by the agent to learn about the environment
- a **reward function**, which is used to train your agent how to behave
- an **environment**, which is a scenario the agent has to face
- an **action**, which is done by the agent in the environment
- and last but not the least, an **agent** which does all the deeds!



Source: UTCS RL Reading Group

2. Examples of Reinforcement Learning

Now, I am sure you must be thinking how the experiment conducted on animals can be relevant to people practicing machine learning. This is what I thought when I came across reinforcement learning first.

A lot of beginners tend to think that there are only 2 types of problems in machine learning – Supervised machine learning and Unsupervised machine learning. I don't know where this notion comes from, but the world of machine learning is much more than the 2 types of problems mentioned above. Reinforcement learning is one such class of problems.

Let's look at some real-life applications of reinforcement learning. Generally, we know the start state and the end state of an agent, but there could be multiple paths to reach the end state – reinforcement learning finds an application in these scenarios. **This essentially means that driverless cars, self navigating vacuum cleaners, scheduling of elevators are all applications of Reinforcement learning.**

Here is a video of a game bot trained to play flappy bird.

3. What is a Reinforcement Learning Platform?

Before we look into what a platform is, let's try to understand a reinforcement learning environment.

A reinforcement learning environment is what an agent can observe and act upon. The horizon of an agent is much bigger, but it is the task of the agent to perform actions on the environment which can help it maximize its reward. As per "A brief introduction to reinforcement learning" by Murphy (1998),

The environment is modeled as a stochastic finite state machine with inputs (actions sent from the agent) and outputs (observations and rewards sent to the agent).

Let's take an example,

This is a typical game of mario. Remember how you played this game. Now consider that you are the "agent" who is playing the game.

Now you have "access" to a land of opportunities, but you don't know what will happen when you do something, say smash a brick. You can see a limited amount of "environment", and until you traverse around the world you can't see everything. So you move around the world, trying to perceive what entails ahead of you, and at the same time try to increase your chances to attain your goal.

This whole "story" is not created by itself. You have to "render" it first. And that is the main task of the platform, viz to create everything required for a complete experience – the environment, the agent and the rewards.

4. Major Reinforcement Learning Platforms

i) Deepmind Lab

DeepMind Lab is a fully 3D game-like platform tailored for agent-based AI research

A recent release by Google Deepmind, **Deepmind lab** is an integrated agent-environment platform for general artificial intelligence research with a focus on first person perspective games. It was built to accommodate the research done at DeepMind. Deepmind lab is based on an open-source engine ioquake3 , which was modified to be a flexible interface for integration with artificial systems.

Things I liked

- It has richer and realistic visuals.
- Closer integration with the gaming environment

Things I did not like

- It still lacks variety in terms of a gaming environment, which would get built over time by open source contributions.

- Also at the moment, it supports only Linux, but has been tested on different OS. Bazel (which is a dependency for deepmind lab) is experimental for windows. So windows support for Deepmind lab is still not guaranteed.

Markov Decision Process:

The mathematical framework for defining a solution in reinforcement learning scenario is called **Markov Decision Process**. This can be designed as:

- Set of states, S
- Set of actions, A
- Reward function, R
- Policy, π
- Value, V

We have to take an action (A) to transition from our start state to our end state (S). In return getting rewards (R) for each action we take. Our actions can lead to a positive reward or negative reward.

The set of actions we took define our policy (π) and the rewards we get in return defines our value (V). Our task here is to maximize our rewards by choosing the correct policy. So we have to maximize

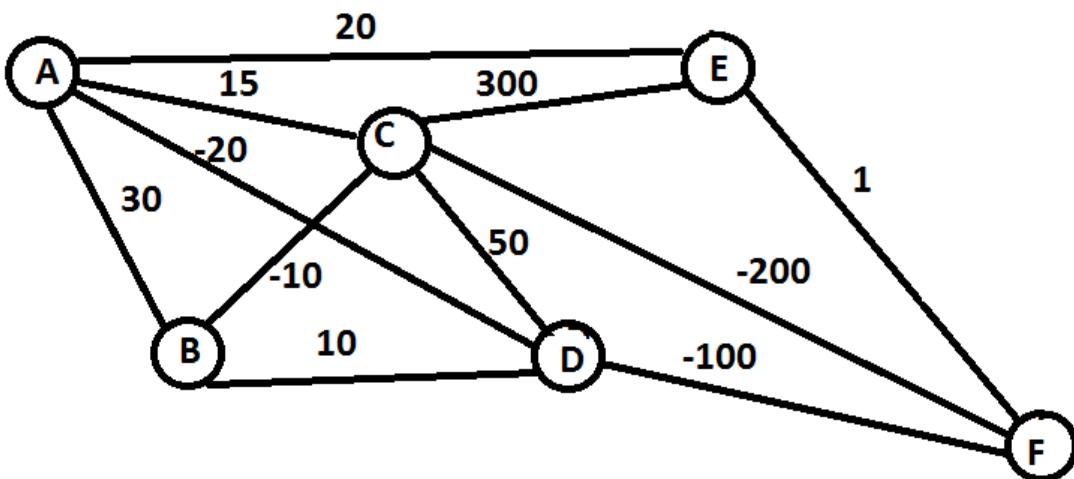
$$E(r_t | \pi, s_t)$$

for all possible values of S for a time t.

Shortest Path Problem



Let me take you through another example to make it clear.



This is a representation of a shortest path problem. The task is to go from place A to place F, with as low cost as possible. The numbers at each edge between two places represent the cost taken to traverse the distance. The negative cost are actually some earnings on the way. We define Value is the total cumulative reward when you do a policy.

Here,

- The set of states are the nodes, viz {A, B, C, D, E, F}
- The action to take is to go from one place to other, viz {A -> B, C -> D, etc}
- The reward function is the value represented by edge, i.e. cost
- The policy is the "way" to complete the task, viz {A -> C -> F}

Now suppose you are at place A, the only visible path is your next destination and anything beyond that is not known at this stage (a.k.a observable space).

You can take a greedy approach and take the best possible next step, which is going from {A -> D} from a subset of {A -> {B, C, D, E}}. Similarly now you are at place D and want to go to place F, you can choose from {D -> {B, C, F}}. We see that {D -> F} has the lowest cost and hence we take that path.

So here, our policy was to take {A -> D -> F} and our Value is -120.

Congratulations! You have just implemented a reinforcement learning algorithm. This algorithm is known as **epsilon greedy**, which is literally a greedy approach to solving the problem. Now if you (the salesman) want to go from place A to place F again, you would always choose the same policy.

Other ways of travelling?

Can you guess which category does our policy belong to i.e. (pure exploration vs pure exploitation)?

Notice that the policy we took is not an optimal policy. We would have to "explore" a little bit to find the optimal policy. The approach which we took here is policy based learning, and our task is to find the optimal policy among all the possible policies. There are different ways to solve this problem, I'll briefly list down the major categories

- **Policy based**, where our focus is to find optimal policy
- **Value based**, where our focus is to find optimal value, i.e. cumulative reward
- **Action based**, where our focus is on what optimal actions to take at each step

I would try to cover in-depth reinforcement learning algorithms in future articles. Till then, you can refer to this paper on a survey of reinforcement learning algorithms.

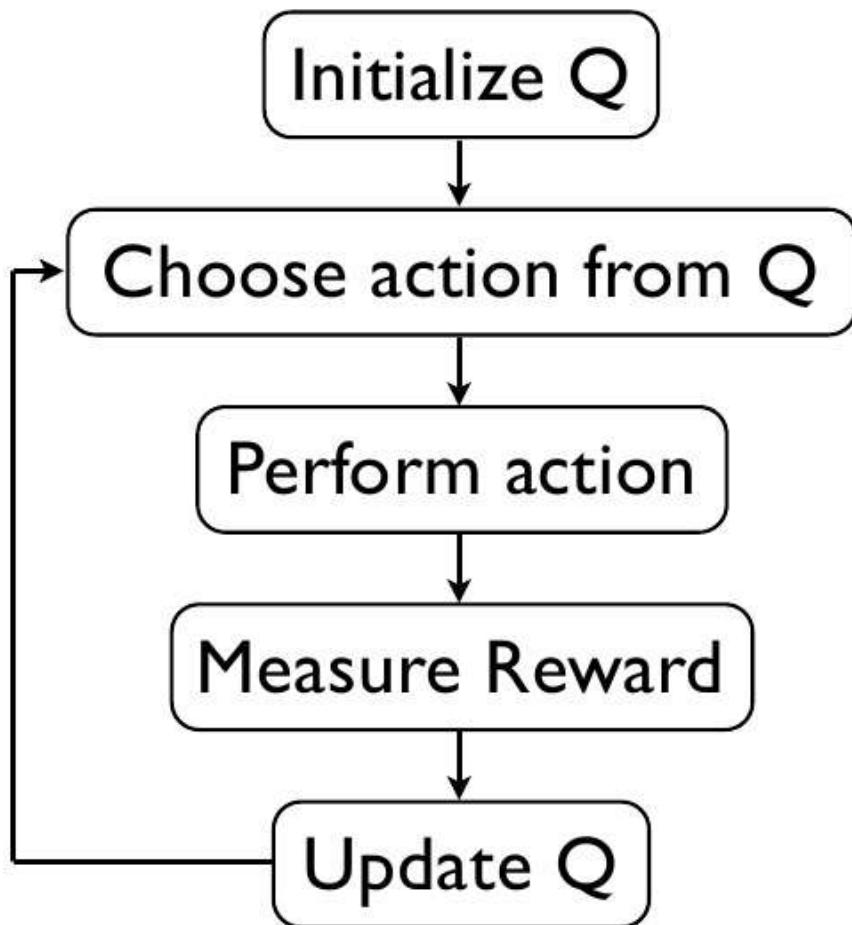
4. An implementation of Reinforcement Learning

We will be using Deep Q-learning algorithm. Q-learning is a policy based learning algorithm with the function approximator as a neural network. This algorithm was used by Google to beat humans at Atari games!

Let's see a pseudocode of Q-learning:

1. Initialize the Values table ' $Q(s, a)$ '.
2. Observe the current state ' s '.
3. Choose an action ' a ' for that state based on one of the action selection policies (eg. epsilon greedy)
4. Take the action, and observe the reward ' r ' as well as the new state ' s' .
5. Update the Value for the state using the observed reward and the maximum reward possible for the next state. The updating is done according to the formula and parameters described above.
6. Set the state to the new state, and repeat the process until a terminal state is reached.

A simple description of Q-learning can be summarized as follows:



We will first see what Cartpole problem is then go on to coding up a solution

When I was a kid, I remember that I would pick a stick and try to balance it on one hand. Me and my friends used to have this competition where whoever balances it for more time would get a "reward", a chocolate!

Here's a short video description of a real cart-pole system

Let's code it up!

To setup our code, we need to first install a few things,

Step 1: Install keras-rl library

From terminal, run the following commands:

```
git clone https://github.com/matthiasplappert/keras-rl.git
cd keras-rl
python setup.py install
```

Step 2: Install dependencies for CartPole environment

Assuming you have pip installed, you need to install the following libraries

```
pip install h5py
pip install gym
```

Step 3: lets get started!

First we have to import modules that are necessary

```
import numpy as np
import gym
from keras.models import Sequential
from keras.layers import Dense, Activation, Flatten
from keras.optimizers import Adam
from rl.agents.dqn import DQNAgent
from rl.policy import EpsGreedyQPolicy
from rl.memory import SequentialMemory
```

Then set the relevant variables

```
ENV_NAME = 'CartPole-v0'
# Get the environment and extract the number of actions available in the Cartpole problem
env = gym.make(ENV_NAME)
np.random.seed(123)
env.seed(123)
nb_actions = env.action_space.n
```

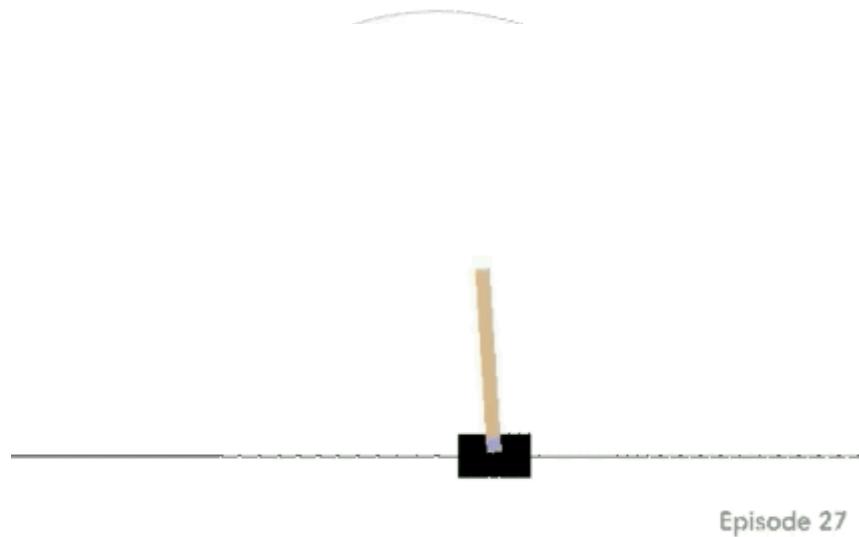
Next, we build a very simple single hidden layer neural network model.

```
model = Sequential()
model.add(Flatten(input_shape=(1,) + env.observation_space.shape))
model.add(Dense(16))
model.add(Activation('relu'))
model.add(Dense(nb_actions))
model.add(Activation('linear'))
print(model.summary())
```

Next, we configure and compile our agent. We set our policy as Epsilon Greedy and we also set our memory as Sequential Memory because we want to store the result of actions we performed and the rewards we get for each action.

```
policy = EpsGreedyQPolicy()
memory = SequentialMemory(limit=50000, window_length=1)
dqn = DQNAgent(model=model, nb_actions=nb_actions, memory=memory,
nb_steps_warmup=10,
target_model_update=1e-2, policy=policy)
dqn.compile(Adam(lr=1e-3), metrics=['mae'])
# Okay, now it's time to learn something! We visualize the training here for show, but this slows
down training quite a lot.
dqn.fit(env, nb_steps=5000, visualize=True, verbose=2)
Now we test our reinforcement learning model

dqn.test(env, nb_episodes=5, visualize=True)
This will be the output of our model:
```



And Voila! You have just built a reinforcement learning bot!

5. Increasing the complexity

Now that you have seen a basic implementation of Re-inforcement learning, let us start moving towards a few more problems, increasing the complexity little bit every time.

Problem – Towers of Hanoi



For those, who don't know the game – it was invented in 1883 and consists of 3 rods along with a number of sequentially-sized disks (3 in the figure above) starting at the leftmost rod. The objective is to move all the disks from the leftmost rod to the rightmost rod **with the least number of moves**. (You can read more on wikipedia)

If we have to map this problem, let us start with states:

- **Starting state** – All 3 disks in leftmost rod (in order 1, 2 and 3 from top to bottom)
- **End State** – All 3 disks in rightmost rod (in order 1, 2 and 3 from top to bottom)

All possible states:

Here are our 27 possible states:

All disks in a rod	One disk in a Rod	(13) disks in a rod	(23) disks in a rod	(12) disks in a rod
(123)**	321	(13)2*	(23)1*	(12)3*
(123)	312	(13)*2	(23)*1	(12)*3
**(123)	231	2(13)*	1(23)*	3(12)*
	132	*(13)2	*(23)1	*(12)3
	213	2*(13)	1*(23)	3*(12)
	123	*2(13)	*1(23)	*3(12)

Where (12)3* represents disks 1 and 2 in leftmost rod (top to bottom) 3 in middle rod and * denotes an empty rightmost rod

Numerical Reward:

Since we want to solve the problem in least number of steps, we can attach a reward of -1 to each step.

Policy:

Now, without going in any technical details, we can map possible transitions between above states. For example (123)** \rightarrow (23)1* with reward -1. It can also go to (23)*1

If you can now see a parallel, each of these 27 states mentioned above can represent a graph similar to that of shortest path algorithm above and we can find the most optimal solutions by experimenting various states and paths.

Problem – 3 x 3 Rubix Cube

While I can solve this for you as well, I would want you to do this by yourself. Follow the same line of thought I used above and you should be good.

Start by defining the Starting state and the end state. Next, define all possible states and their transitions along with reward and policy. Finally, you should be able to create a solution for solving a rubix cube using the same approach.

A Peek into Recent Advancements in Reinforcement Learning

As you would realize that the complexity of this Rubix Cube is many folds higher than the Towers of Hanoi. You can also understand how the possible number of options have increased in number. Now, think of number of states and options in a game of Chess and then in Go! Google DeepMind recently created a deep reinforcement learning algorithm which defeated Lee Sedol!

With the recent success in Deep Learning, now the focus is slowly shifting to applying deep learning to solve reinforcement learning problems. The news recently has been flooded with the defeat of Lee Sedol by a deep reinforcement learning algorithm developed by Google DeepMind. Similar breakthroughs are being seen in video games, where the algorithms developed are achieving human-level accuracy and beyond. Research is still at par, with both industrial and academic masterminds working together to accomplish the goal of building better self-learning robots



Source

Some major domains where RL has been applied are as follows:

- Game Theory and Multi-Agent Interaction
- Robotics
- Computer Networking
- Vehicular Navigation
- Medicine and
- Industrial Logistic.



AI - Data Science Diploma

2019
EDITION

Statistics

QualityThought®

Reach Us:

Quality Thought Infosystems Pvt. Ltd.
#208B, 2nd Floor, Nilgiri Block, Aditya Enclave
Ameerpet, Hyderabad, Telangana - 38

+ 91 - 9515151992

About Institute?

Quality Thought is a professional Training Organization for software developers, IT administrators, and other professionals. It's Located in Hyderabad, India. The training is offered in Four major modes: Classroom Room Trainings, Online instructor Led Trainings, Self-paced e-learning trainings, and Corporate Trainings.

About Course?

This is an Artificial Intelligence Engineer Master Course that is a comprehensive learning approach for mastering the domains of Artificial Intelligence, Data Science, Business Analytics, Business Intelligence, Python coding, and Deep Learning with TensorFlow. Upon completion of the training, you will be able to take on challenging roles in the artificial intelligence domain.

why we take course?

Artificial intelligence is one of the hottest domains being heralded as the one with the ability to disrupt companies cutting across industry sectors. This Quality Thought Artificial Intelligence Engineer Master Course will equip you with all the necessary skills needed to take on challenging and exciting roles in the artificial intelligence, data science, business analytics, Python, R statistical computing domains and grab the best jobs in the industry at top-notch salaries.

Courses Covered in AI Diploma:

Course 1: As a Future ready IT employee I am interested to learn foundations of Data Analytics with statistics and Tableau, R

Course 2: I want to redefine my Analytical knowledge into python programming for Machine Learning Engineer

Course 3: I want to apply statistics and Python on Machine Learning models for Predictions& classifications of Data in various industry segments for intelligent Business

Course 4: So now I am ready to apply machine Learning models to implement Recommendation systems and Creating Machine Learning service in the cloud(IBM WATSON,AWS)

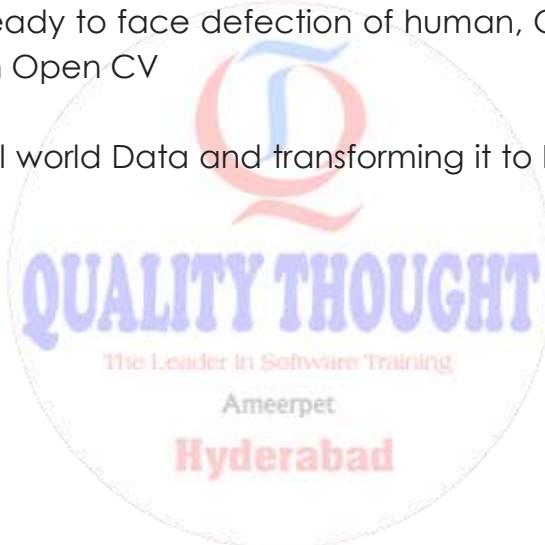
Course 5: Understanding tools of Bigdata and implementation using Machine Learning

Course 6: Applying Machine learning to speech Analytics to process Text using NLP

Course 7: Start Learning Neural Networks using Tensor flows and Keras for image classification and Data Extraction from Image(OCR)

Course 8: Now I am ready to face detection of human, Object using computer vision Technology with Open CV

Course 9: Sensing Real world Data and transforming it to Intelligent actions using IOT



INTRODUCTION

Every day we come across a lot of information in the form of facts, numerical figures, tables, graph, etc. These are provided by newspapers, televisions, magazines, blogs and other means of communication. These may relate to cricket batting or bowling averages, profits of a company, temperatures of cities, expenditures in various sectors of a five year budget plan, polling results, and so on. The numerical facts or figures, collected with a definite purpose are called Data (plural form of Latin word *Datum*).

Our world is becoming more and more information oriented. Every part of our lives utilizes data in one or other forms. So, it becomes essential for us to know how to extract meaningful information from such data which is studied in a branch of mathematics called statistics.

The word statistics appears to have been derived from Latin word *Status* (a political state) meaning collection of data on different aspects of the life of the people, useful to the State.

Later Scientists seek to answer questions using rigorous methods and careful observations. The observations which are collected from field notes, surveys, and experiments form the main pillar or backbone of a statistical investigation and are called data. Statistics is the study of how best to

- collect,
- organize,
- analyze,
- interpret and
- present the data.

However, many of the investigations done by scientists can be addressed with a small number of data collection techniques, analytic tools, and fundamental concepts in statistical inference.

Let's get into details of each stage

POPULATION

Don't get confused when we hear the word population, we typically think of all the people living in a town, state, or country. In statistics, a population is an entire group about which some information is required to be ascertained. A statistical population need not consist only of people. We can have population of heights, weights, BMIs, hemoglobin levels, events, outcomes. In selecting a population for study, the research question or purpose of the study will suggest a suitable definition of the population to

be studied, in terms of location and restriction to a particular age group, sex or occupation. The population must be fully defined so that those to be included and excluded are clearly spelt out (inclusion and exclusion criteria). For example, if we say that our study populations are all lawyers in Delhi, we should state whether those lawyers are included who have retired, are working part-time, or non-practicing, or those who have left the city but still registered at Delhi.

Use of the word population in epidemiological research does not correspond always with its demographic meaning of an entire group of people living within certain geographic or political boundaries. A population for a research study may comprise groups of people defined in many different ways, for example, coal mine workers in singareni, or pilgrims travelling to KumbhMela at Allahabad. The type of information gathering over a whole population by government is called a census.

The limitation of studying population is it is difficult to observe all the individuals in any given group. So, statisticians also study subpopulations and take samples of large population to accurately analyze the full spectrum of behaviors and characteristics of the population at large. So, lets learn about samples.

SAMPLE

A sample is any part of the fully defined population. A syringe full of blood drawn from a patient's vein is a sample of all of his blood in circulation at the moment. Similarly, 100 patients of schizophrenia in a clinical study is a sample of the population of schizophrenics, provided the sample is properly chosen and the inclusion and exclusion criteria are well defined.

To make accurate inferences, the sample has to be representative in which each and every member of the population has an equal and mutually exclusive chance of being selected.

Target population, study population and study sample

A population is a complete set of people with a specialized set of characteristics, and a sample is a subset of the population. The usual criteria we use in defining population are geographic, for example, "the population of Andhra Pradesh". In medical research, the criteria for population may be clinical, demographic and time related.

- Clinical and demographic characteristics define the target population, the large set of people in the world to which the results of the study will be generalized (e.g. all schizophrenics).
- The study population is the subset of the target population available for study (e.g. schizophrenics in the researcher's town).
- The study sample is the sample chosen from the study population.

METHODS OF SAMPLING

Purposive (non-random samples)

- Volunteers who agree to participate
- Convenient sample such as captive medical students or other readily available groups
- Quota sampling, selection of a fixed number from each group
- Referred cases who may be under pressure to participate

Non-random samples do have certain limitations.:

- The larger group is difficult to identify. The results would be valid for the sample only.
- They can never provide clues for further studies based on random samples.
- The statistical inferences such as confidence intervals and tests of significance cannot be estimated from non-random samples

Random sampling methods

Simple random sampling

A sample may be defined as random if every individual in the population being sampled has an equal likelihood of being included. Random sampling is the basis of all good sampling techniques and disallows any method of selection. To undertake a separate exercise of listing the population for the study may consume time and monotonous. Two-stage sampling makes the task easier.

The usual method of selecting a simple random sample from a listing of individuals is to assign a number to each individual and then select certain numbers by reference to random number tables which are published in standard statistical textbooks. Random number can also be generated by statistical software such as EPI INFO developed by WHO and CDC Atlanta.

Systematic sampling

A simple method of random sampling is to select a systematic sample in which every n^{th} person is selected from a list. A systematic sample can be drawn from a queue of people or from patients ordered according to the time of their attendance at a clinic. To fulfill the statistical criteria for a random sample, a systematic sample should be drawn from subjects who are randomly ordered. The starting point for selection should be randomly chosen.

Multistage sampling

Sometimes, a strictly random sample may be difficult to obtain and it may be more feasible to draw the required number of subjects in a series of stages. For example, suppose we wish to estimate the number of CATSCAN examinations made of all patients entering a hospital in a given month in the state of Maharashtra. It would be quite tedious to devise a scheme which would allow the total population of patients to be directly sampled. However, it would be easier

- To list the districts of the state of Maharashtra and randomly draw a sample of these districts.
- Within this sample of districts, all the hospitals would then be listed by name, and a random sample of these can be drawn.
- Within each of these hospitals, a sample of the patients entering in the given month could be chosen randomly for observation and recording.

Thus, by stages, we draw the required sample. If indicated, we can introduce some element of stratification at some stage (urban/rural, gender, age).

It should be cautioned that multistage sampling should only be resorted to when difficulties in simple random sampling are insurmountable.

TYPES

The two main types of statistics are descriptive statistics and inferential statistics.

As we know that the steps to study a survey or an experiment are to collect, organize, analyze, interpret and present the data. Now the steps are divided into two groups where the initial steps like collecting, organizing and presenting belong to Descriptive statistics and the remaining two steps like analyzing and interpreting(drawing the conclusion) the data belong to Inferential statistics.

Since the name Descriptive it involves all the describing about the numbers obtained in the experiment or survey and preparing the data for analysis by finding the measures of central tendency, spread, shape, regressions.etc, depending upon the type of data .Descriptive statistics are limited in so much that they only allow you to make summations about the people or objects that you have actually measured. For example, if we have a new drug which cures a particular virus and it worked on a set of patients, we cannot claim that it would work on other set of patients only based on descriptive statistics. This is where inferential statistics comes in.

As per wiki, Univariate analysis describes the distribution of a single variable, such as

- central tendency like mean, median, and mode
- dispersion such as range and quartiles of the data-set
- measures of spread like the variance and standard deviation.

- shape of the distribution like the skewness and kurtosis.

variable's distribution can also be depicted in graphical or tabular format, including histograms and stem-and-leaf display.

When a sample consists of more than one variable, descriptive statistics can be used to describe the relationship between pairs of variables. In this case, descriptive statistics include:

- Cross-tabulations and contingency tables
- Graphical representation via scatter plots
- Quantitative measures of dependence
- Descriptions of conditional distributions

The main reason for differentiating univariate and bivariate analysis is that bivariate analysis is it describes the relationship between two different variables. Quantitative measures of dependence include correlation and covariance. The un-standardized slope indicates that the unit change in the response variable for one unit change in the predictor variable. The standardized slope indicates this change in standardized (z-score) units. The data having high skewness are often transformed by taking logarithms. Use of logarithms makes graphs more symmetrical and looks more similar to the normal distribution, making them easier to interpret intuitively.

On the other hand, Inferential statistics involves drawing the right conclusions from the descriptive statistics. The methods of inferential statistics are the estimation of parameters and testing of statistical hypotheses. Finally, these inferences make studies important for the future generalizations about a population by studying a smaller sample. While drawing conclusions, one must be very careful to not to draw the wrong or biased conclusions. For example, data dredging is becoming a bigger problem as computers store loads of information which makes it easy(intentionally or unintentionally) to use the wrong inferential methods. The important limitation being the data about a population provided is not fully measured hence we are not completely sure that the values/statistics we calculate are correct. Because, inferential statistics uses the values measured in a sample to infer the values that would be measured in a population; there will always be a degree of uncertainty in doing this. And the second one being chances of biased conclusions.

Inferential Statistics make propositions using data drawn from the population with some form of sampling. Given a hypothesis about a population, for which we wish to draw inferences, statistical inference consists of (first) selecting a model of the process that generates the data and (second) deducing propositions from the model. The conclusion of inferential statistics is known as a statistical proposition. Some common forms of statistical proposition are :

- a point estimate(a particular value that approximates parameter at it's very best)
- an interval estimate, e.g. a confidence interval (or set estimate), i.e. an interval constructed using a dataset drawn from a population contains a true

parameter value with the probability at the stated confidence level under repeated sampling.

- a credible interval (a set of values containing in a particular interval)
- rejection of a hypothesis
- clustering of data points

Both descriptive statistics and inferential statistics go hand in hand and cannot exist without one another.

We can simply further divide Descriptive and Inferential statistics as shown

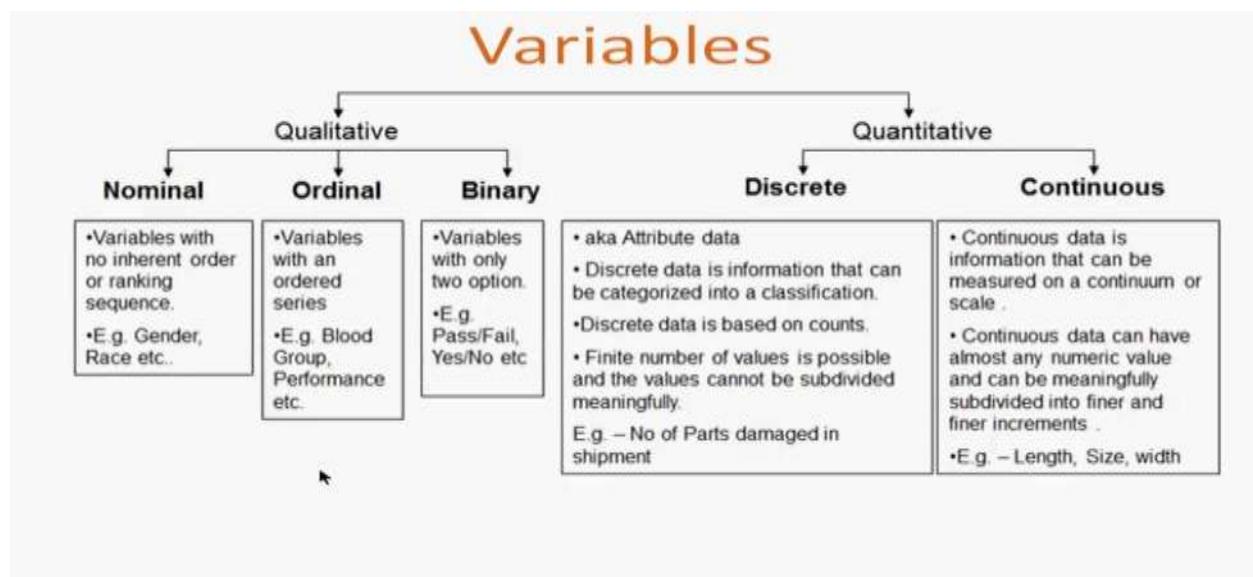
VARIABLES

Suppose we are conducting an experiment to count the marks of students in a class based on a surprise test. We want to know how many students can attempt the test based on memory. We get a list of marks which is varying its value among the students. This item marks is known as variable which is studied in a sample or population.

It can also be defined as below:

- A statistical variable is each of the characteristics or qualities that any individual of a population possess and can be any number, characteristic or quantity which can be counted.
- A variable is a data item, where the value varies between data units in a population and can change value over time.

Variables can be further categorized as below:



Qualitative variables can be values that are names or labels. Let's have a look at this table:

Scale	Properties	Examples	Qualitative / Quantitative
Nominal Operations: =, \neq	There will be a difference however order will not be changed.	Maruthi = 1 Hyundai = 2 Volkswagen = 3 Audi = 4 BMW = 5 Other = 6	Qualitative
Ordinal Operations: =, \neq , $<$, $>$	There will be a difference and direction of the difference will be indicated (less than or more than)	Food taste Agree = 1 Strongly agree = 2 Disagree = 3 Strongly disagree = 4 Don't know = 5	Qualitative
Binary Operations: =, \neq	There will be a difference as it has only 2 values True/False or 0/1	Sun rises in the east? True	Qualitative

Quantitative variables are numerical measurable quantity. For example, when we say population of a city, we are thinking about the number of people in the city which is a measurable attribute (quantity) of the city. Therefore, population is a quantitative variable.

We can further classify Quantitative variables as discrete or continuous. If a variable holds any value between its minimum value and its maximum value, it can be called as continuous variable; otherwise, a discrete variable.

Some of these examples will differentiate discrete and continuous variables:

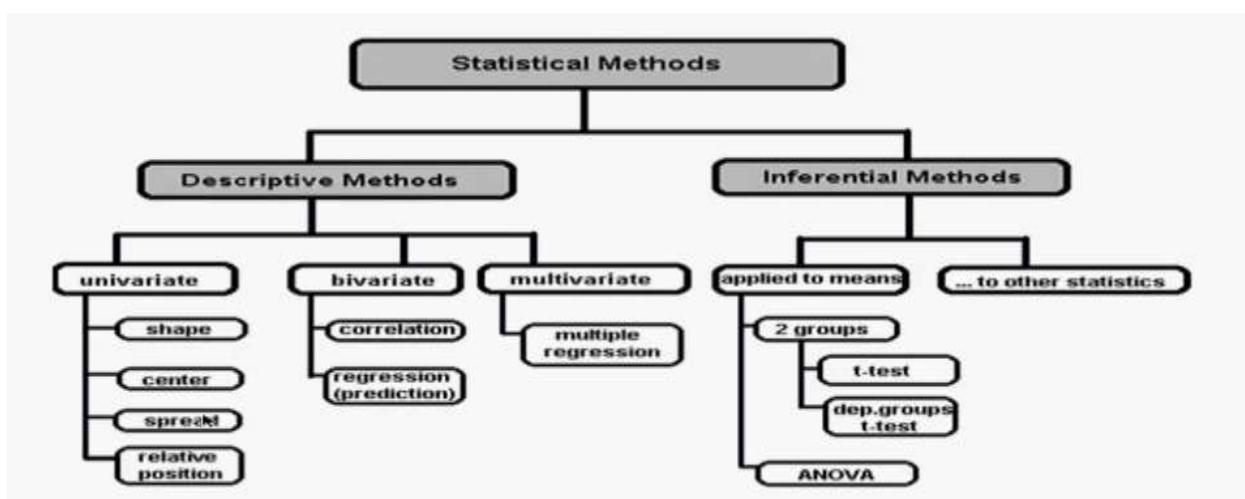
- Suppose the fire department mandates that all fire fighters must weigh between 75 and 90 kilos. The weight of a fire fighter would be an example of a continuous variable; since a fire fighter's weight could take on any value between 75.0 and 90.0 kilos.
- Suppose we flip a coin to count the number of heads. The number of heads can hold any integer value between 0 and $+\infty$. But, We cannot get 2.3 heads. So, the number of heads must be a discrete variable.

In an experiment, a variable can be divided into two types:

- Independent(explanatory/predictor) variable,
- Dependent(response) variable .

A variable which influences changes in the response variable is known as predictor variable and is often known as the independent variable. Independent variable can be random or non random. An independent variable is mostly an input or cause variable. It is tested to see if it is the cause. The explanatory variable can be either categorical data or quantitative data.

In an experimental study, response variable is used as a measure of an outcome . It represents the output or effect and is tested to see if it is the effect. Response variable is dependent on the explanatory variable and is also known as dependent variable, explained variable, outcome variable, experimental variable. It is called as dependent variable which depends on the independent variable. Suppose we held a survey to know how stress rate affects heart rate in human kind. The dependent variable is the heart rate which varies with stress, the independent variable is stress.



DATA

Datum is the singular form of the nounwhere as Data is plural form which has been since 20th century. Data can be categorized as either numeric or nonnumeric. Specific terms are used as follows:

1. Qualitative data or Nonnumeric data

When the data are classified according to some qualitative terms like honesty, beauty, employment, intelligence, occupation, sex, literacy, etc, the classification is termed as qualitative or descriptive or with respect to attributes. Qualitative data are often termed as categorical data.

Univariate data: The data identified on the basis of single characteristic is called as single attribute/univariate data. The number of students in a class room based on the characteristic gender. Here, we consider the characteristic as boys and girls.

Multi variate data: The data identified on the basis of more than one characteristic is called as multivariate data. The number of students on a class room based on the characteristic height and gender. Here, we consider the characteristic boys and girls and also tall and short. If the data is classified into two or more classes with respect to a given attribute, it is said to be a manifold classification. For example, for the attribute intelligence the various classes may be, genius, very intelligent, average intelligent, below average and dull.

2. Quantitative data or numeric data

Quantitative data are collected as countable numbers. They are usually subjected to statistical procedures such as calculating the mean, frequency distribution, standard deviation etc. Moving onto higher statistical analysis such as t-test, factor analysis, Analysis of variance, regression can also be conducted on the data. Quantitative data has four levels of measurement.

Nominal - Nominal refers to categorically discrete data. For example, name of a book, type of car you drive. Nominal sounds like name so it should be easy to remember.

Ordinal - A set of data is said to be ordinal if the observations can be ranked/ordered. It is possible to count and sorted in a particular order but data cannot be measured. Example: T-shirt size (large, medium, small).

Interval - Measurements where the difference between values is measured by a fixed scale and has meaningful intervals but there is no true starting point(zero). Data collected is continuous which has a logical order with standard difference between values. Example: Temperature, Money, Education (In years)

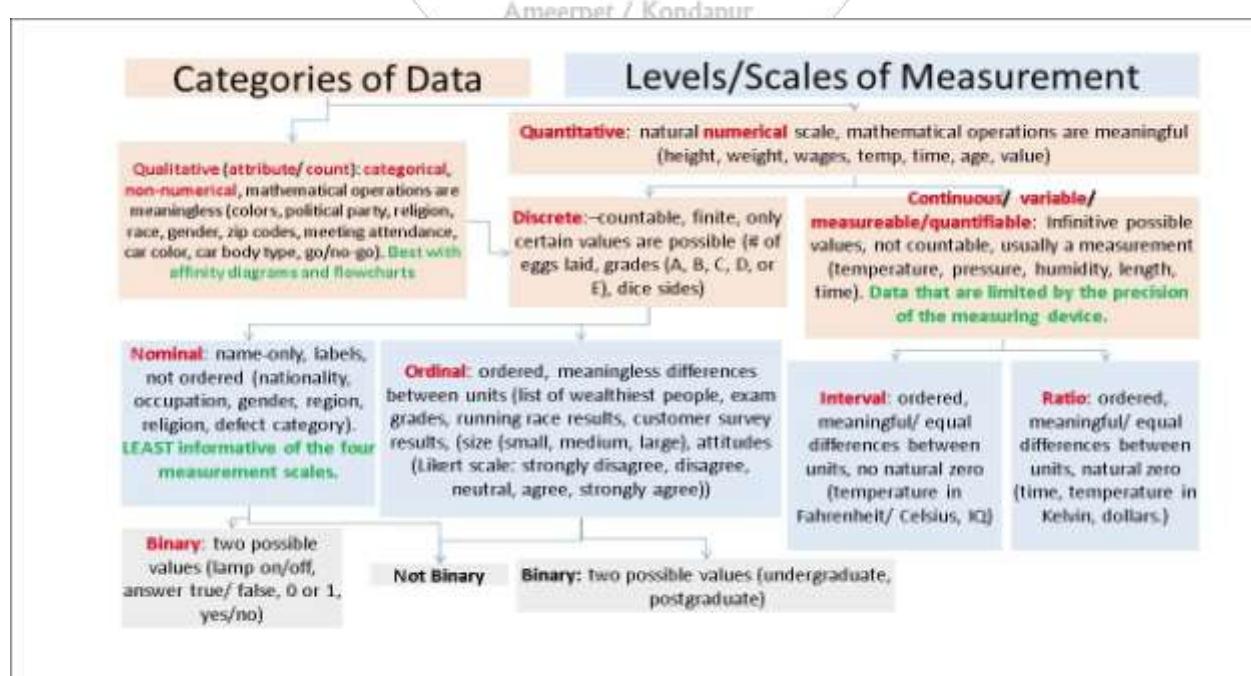
Ratio - Ratio variables are numbers with some base value and there is starting

point(zero). Ratio responses will have order and spacing where multiplication makes sense too. Example: Height, weight.

Once collected data is classified on levels of measurement, then appropriate statistical methods can be used.

Incremental progress	Measure property	Mathematical operators	Advanced operations	Central tendency
Nominal	Classification, membership	=, ≠	Grouping	Mode
Ordinal	Comparison, level	>, <	Sorting	Median
Interval	Difference, affinity	+, -	Yardstick	Mean, Deviation
Ratio	Magnitude, amount	×, /	Ratio	Geometric mean, Coefficient of variation

Depending on the variable types which is actually data can also be included in the categorization as shown



DATA VISUALIZATION

The technique used to convert a set of data into visual insight is known as data visualization. The main aim of data visualization is to give the data a meaningful representation. To create an instant understanding from multi-variable data, it can be displayed as 2d or 3d format images with techniques such as colorization, 3D imaging, animation and spatial annotation.

According to Fisher, Data visualization is "the visual interpretation of complex relationships in multidimensional data." Presenting data in a way that is pleasing to the eye and has the ability to inform and provide value to the user.

A pivotal concept behind the introduction of data visualization is the aim to produce statistical stories. An audience is more likely to learn an idea within a story rather than remember the actual data. By visually displaying data there are many more opportunities for this to occur. Representation of Statistical stories should be able to grab a user's attention, invoking the thought, being informative and ideally entertaining.

The primary objective of any statistical story should be to inform its audience and be newsworthy. It must use the statistics available to provide substance and stimulate interest. It should seek to delve through the large pool of data and only surface those details which will be useful and pertinent to the needs of the user. Once this data has been uncovered the next step must be to ensure that the presentation of the story is in a format that is understandable and easy to use. All statistical stories have a target audience and it is critical that their needs are considered.

There are many different tools for visualizing statistical data. They may be open source or we can purchase .some of the most heard tools mostly open source are :

- Excel

You can actually do some pretty complex things with Excel, from table of cells to scatter plots. As an entry-level tool, it can be a good way of quickly exploring data, or creating visualizations for internal use, but it has limited default set of colors, lines and styles. Excel is part of the Microsoft Office suite, so if you don't have access to it, Google's spreadsheets can do many of the same things.

- R

R is free software environment developed for computing statistics and graphics. A statistical package used to parse large data sets, R is a very complex tool, but has a strong community and package library, with more and more being produced. The learning curve is one of the steepest of any of these tools listed here, but you must be comfortable using it if you want to get to this level.

- Tableau

We can create and share data in real time with Tableau. Tableau public is a popular data visualization tool which is completely free is packed with graphs, charts, maps and more helping users can easily drag and drop data into the system to update in real-time, thereby collaborating with other team members for quick project turnaround.

There is different presentation for qualitative and quantitative data variables.

Qualitative variables:

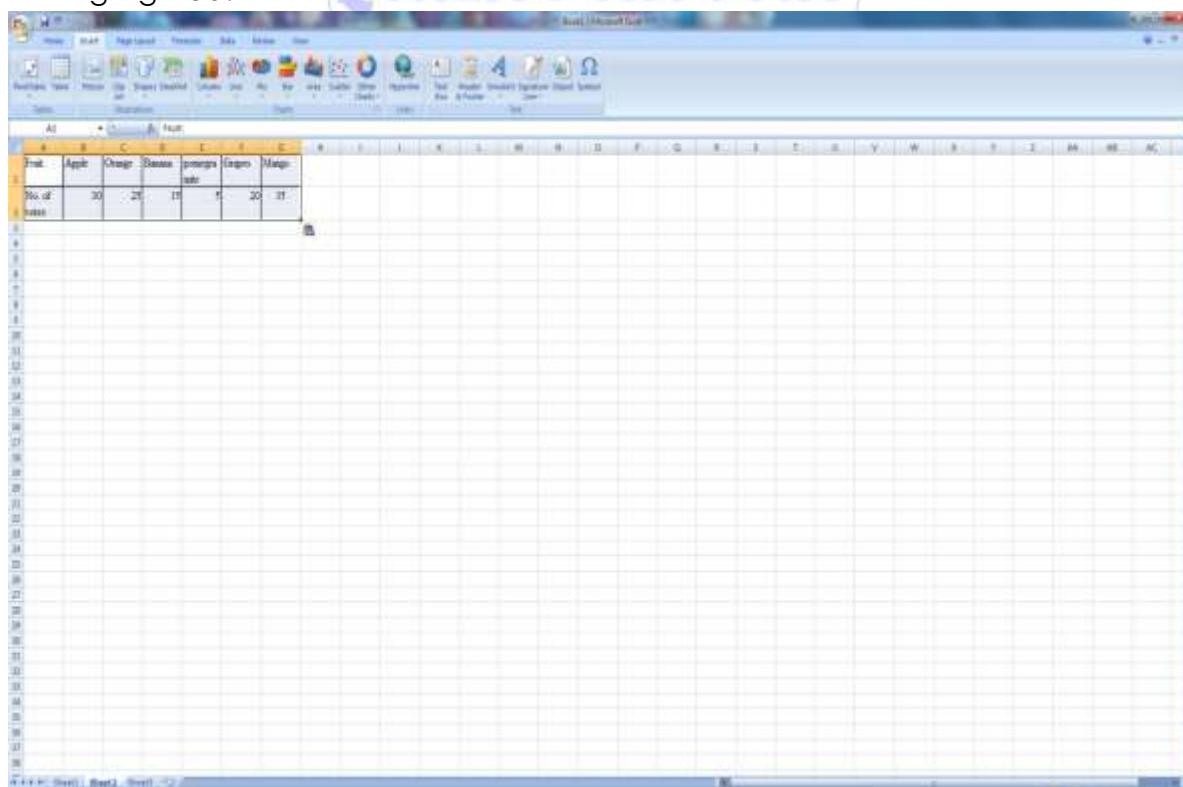
1. The Bar Chart (or Bar Graph) is one of the most common ways of displaying categorical/qualitative data. Bar Graphs mainly holds 2 variables, response (dependent variable) and predictor (independent variable) which can be arranged on the horizontal and vertical axis of a graph. The relationship of the predictor and response variables is shown by a mark of some sort (usually a rectangular box) from one variable's value to the other's.

For example :A survey of 145 people asked them "Which is the nicest fruit?":

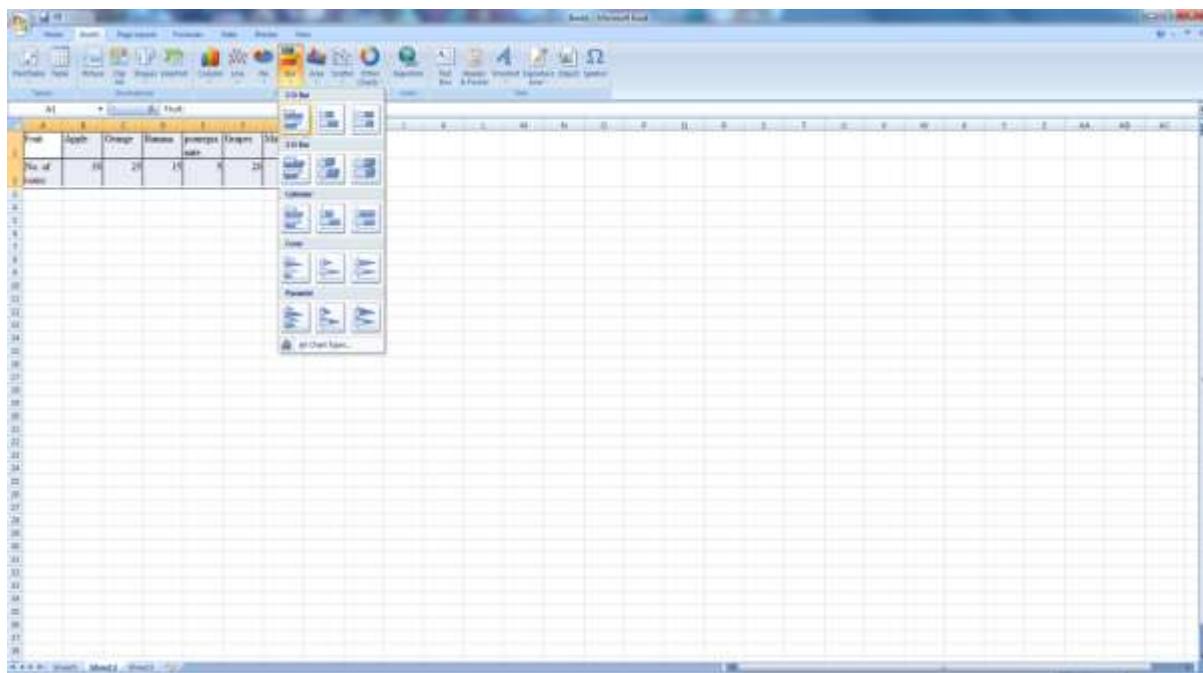
Fruit:	Apple	Orange	Banana	pomegranate	Grapes	Mango
No. of votes:	30	25	15	5	20	35

Lets see how we can use Excel tool to represent this data as bar chart

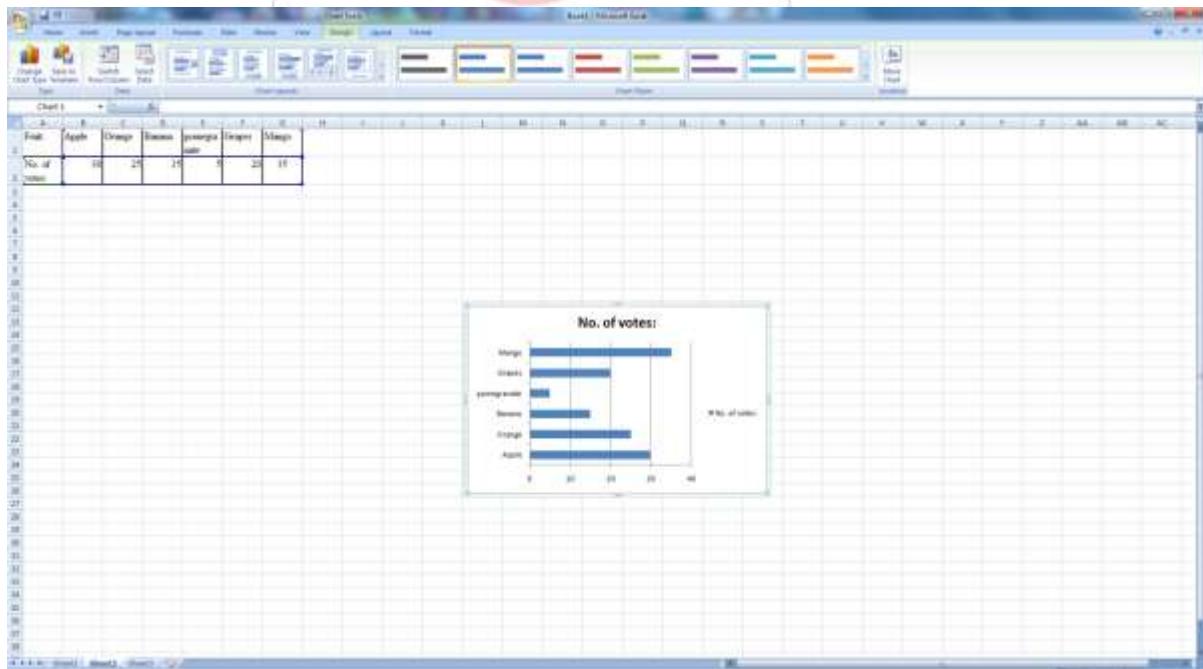
- Open Microsoft Excel and enter the table. Select the table which gets highlighted.



- Insert menu->Bar->2-D Bar



- We get Bar chart displayed horizontally.

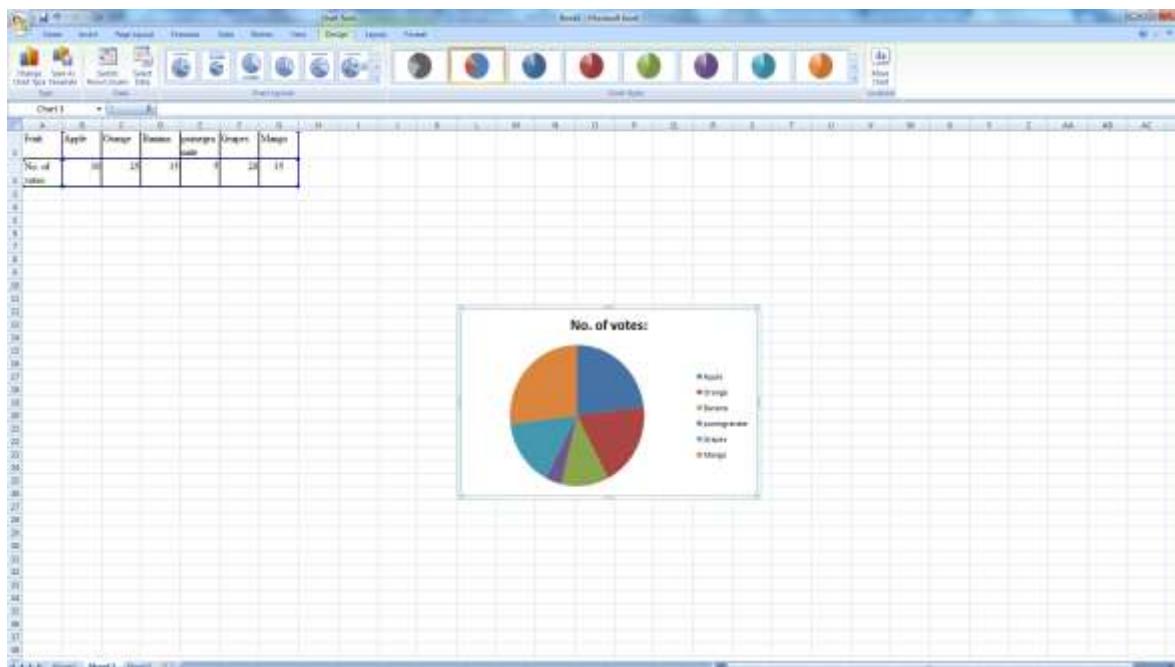
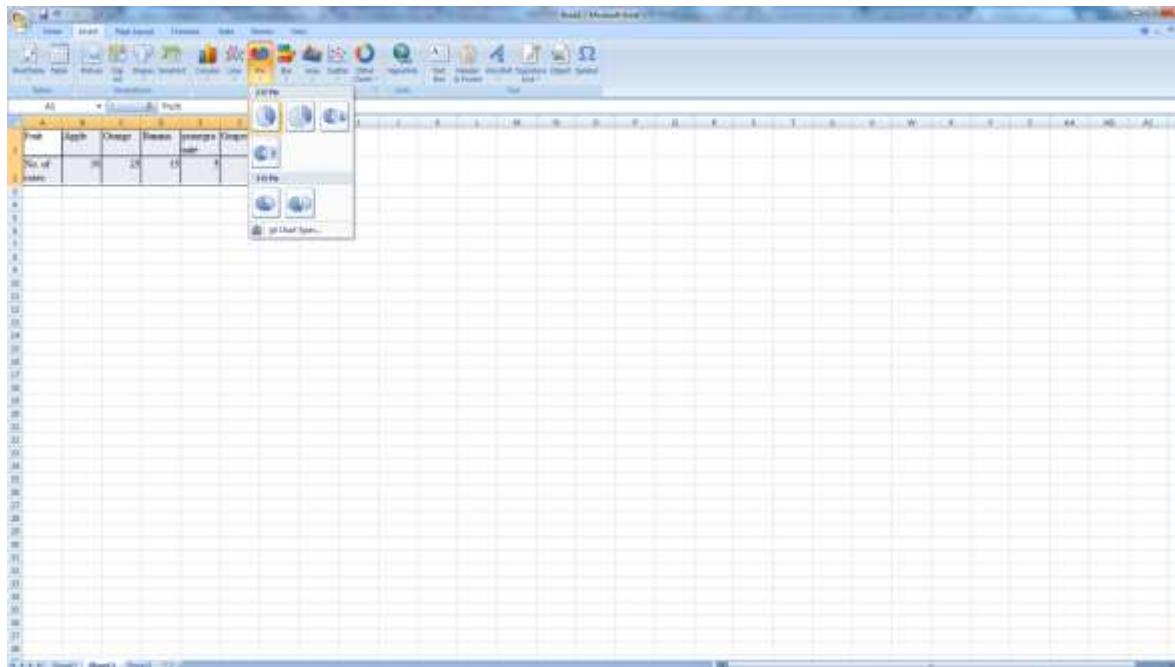


2. A histogram is similar to a bar chart but is used for continuous data. Usually, there is no space between adjacent columns. The columns are positioned over a label that represents a continuous, variable. The column label can either be a single value or a range of values. The size of the group can be equal to the height of column. The area covered by each bar is proportional to the frequency of data.

3. A Pie-Chart/Diagram is a graphical device - a circular shape broken into subdivisions(sectors). These sector areas are proportional to the divided parts. The sectors may be colored differently to show the relationship of parts to the whole.

Lets check out pie chart version for this data

Fruit:	Apple	Orange	Banana	pomegranate	Grapes	Mango
No. of votes:	30	25	15	5	20	35



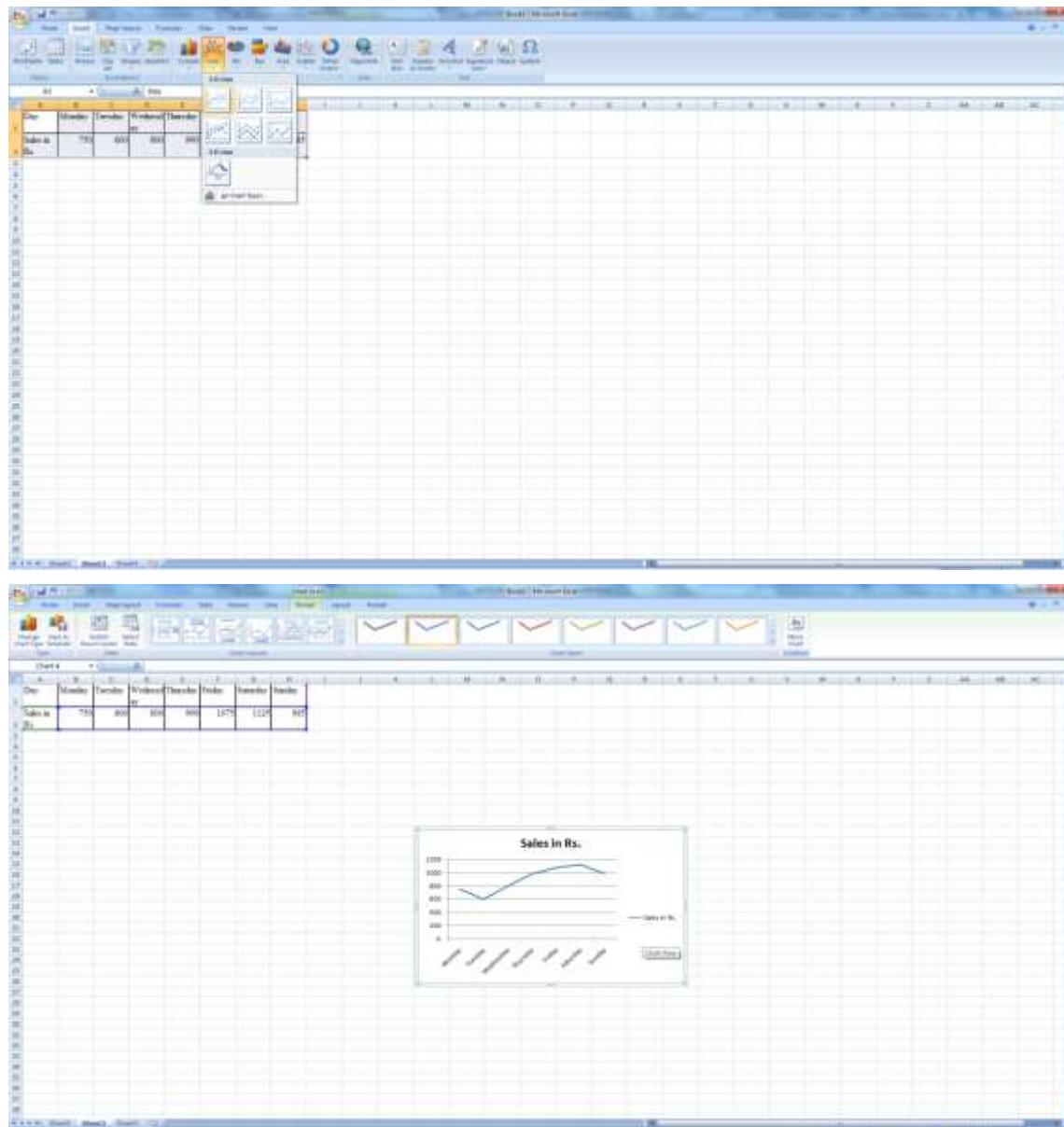
4. A line graph can be, for example, a picture of what happened by/to something (a variable) during a specific time period (also a variable). Usually a line graph is plotted after a table which shows the relationship between the two variables in the form of pairs. Just as in 2D graphs, each of the pairs results to a specific point on the graph, and are connected to one another forming a LINE.

For example:

Sales of ice creams over a week in the month of august are:

Day:	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
Sales in Rs	750	600	800	990	1075	1125	985

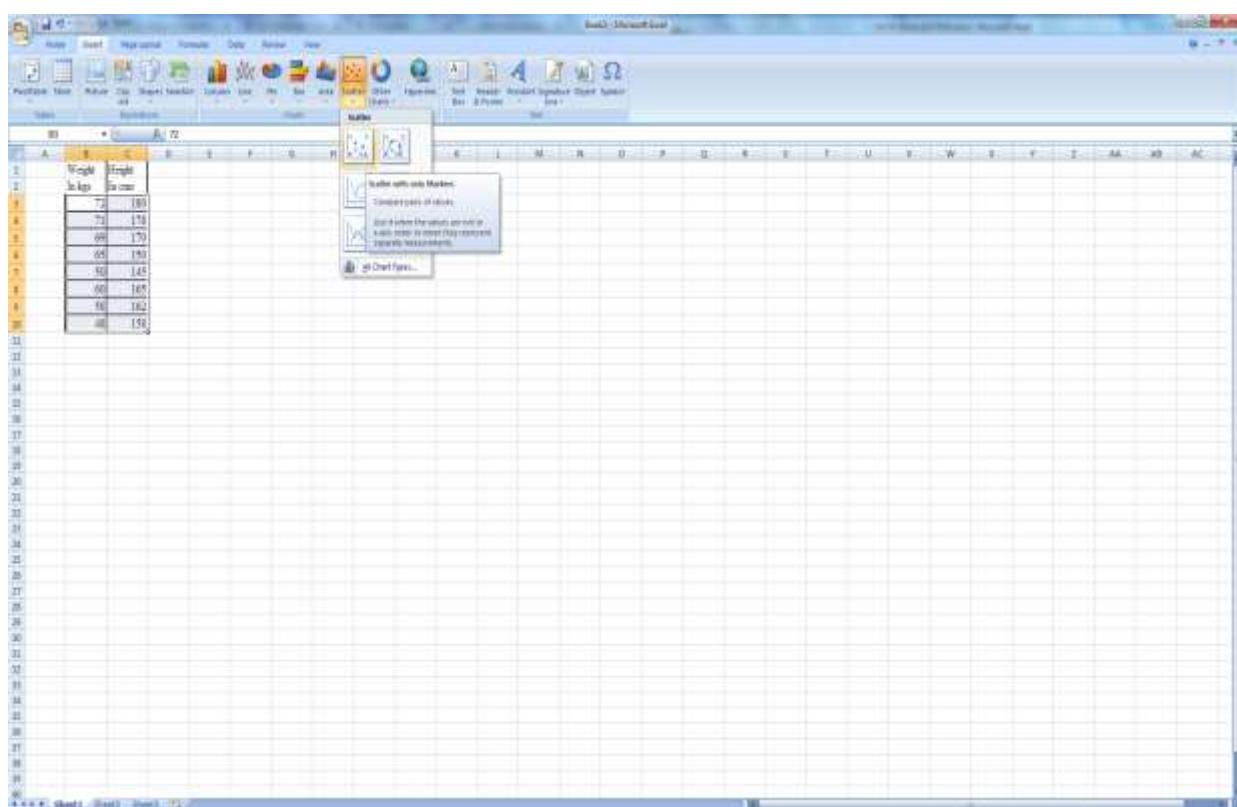
Lets see line graph in excel:

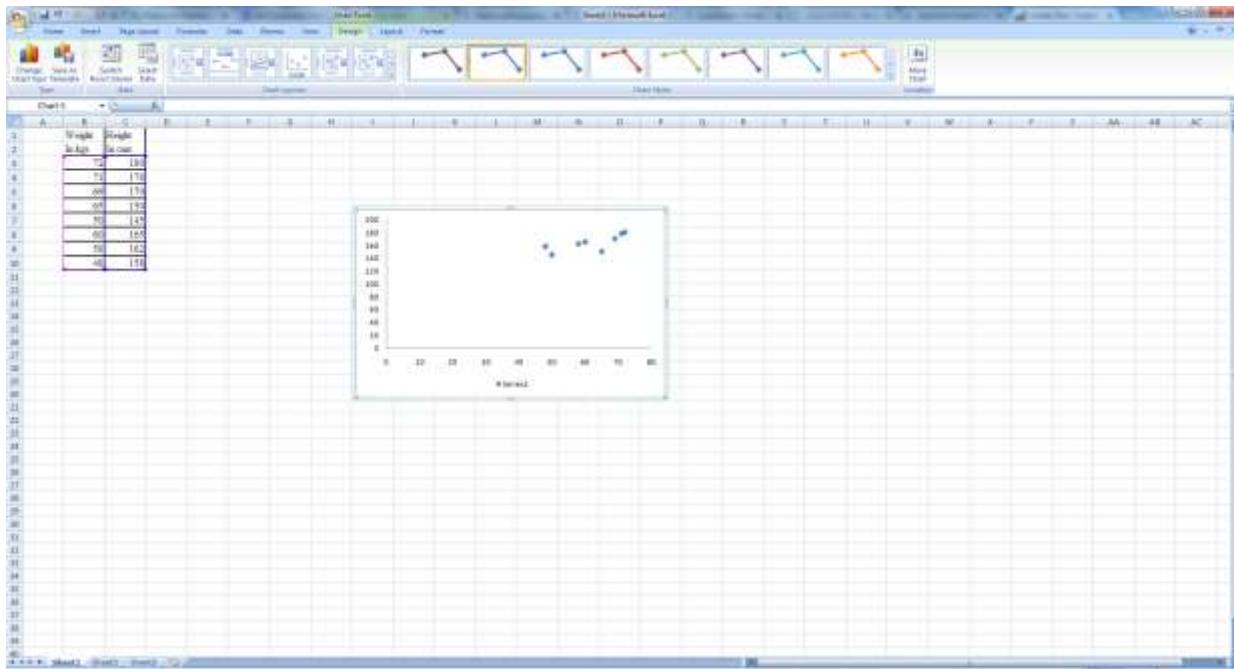


5. Scatter Plot is used to show the relationship between 2 numeric variables. A scatter plot matrix is a collection of pair wise scatter plots of numeric variables.

For example we have height and weight data for students of a class as follows

Height In cms	Weight In kgs
180	72
178	71
170	69
150	65
145	50
165	60
162	58
158	48





We can choose apt presentation for the type of data variables.

MEASURES

In statistics we deal with huge amounts of data related to a particular survey or experiment. We cannot pin locate and analyze the data for future predictions based on each value. The bulkiness of the data can be reduced by organizing it into a frequency table or histogram. Frequency distribution organizes the heap of data into a few meaningful categories. Collected data can also be summarized as a single index/value, which represents the entire data. These measures may also help in the comparison of data. There are three types of measures in statistics based upon the type of data.

1. Measure of central tendency
2. Measure of spread/variability/dispersion
3. Measure of shape

The statistical measure which helps to identify an entire distribution by a single value is known as Central tendency. It is often used as an accurate description of the data. It is the single value that is most typical/representative of the collected data. The three commonly used measures of central tendency are mean, median and mode.

Variability also referred as dispersion/Spread, measures whether the data values are tightly clustered or spread out and how much they differ from the mean value. The spread of the values can be measured for numeric variables (quantitative data) arranged in ascending order. Measures of the spread like variance and standard deviation of the data are present near the mean. If the data set spreads large, the

mean is not as representative of the data as when the spread of data is small. This is because when there are large differences among individual scores it indicates larger spread.

The two numerical measures of shape skewness and kurtosis can be used to test for normality. The data set is not normally distributed when these values are not close to zero.

Let's get into detailed versions.

MEAN

We are aware of the term average during our schooling. In statistics , when dealing with set of quantitative data we term it as mean. It is computed by adding all the values in the data set divided by the number of observations in it.

Mean can be represented using the formula:

$$\text{Mean}(\bar{x}) = \frac{\text{Sum of all values}}{\text{number of values}} = \frac{\Sigma x}{N}$$

Mean of population is denoted by ' μ ' and Mean of sample is denoted by ' \bar{x} '.

Number of items in population is 'N' and in a sample it is 'n'.

$$\mu = \frac{\Sigma x}{N}$$

$$\bar{x} = \frac{\Sigma x}{n}$$

The mean uses each and every value in the dataset and hence is a good representative of the data. Repeated samples drawn from the same population tend to have similar means. Calculating mean eliminates random errors and helps to derive a more accurate result. Therefore the mean is considered as the best measure of central tendency that resists the fluctuation between different samples. The limitation of mean is that it is sensitive to extreme values/outliers, especially when the sample size is small.

Example:

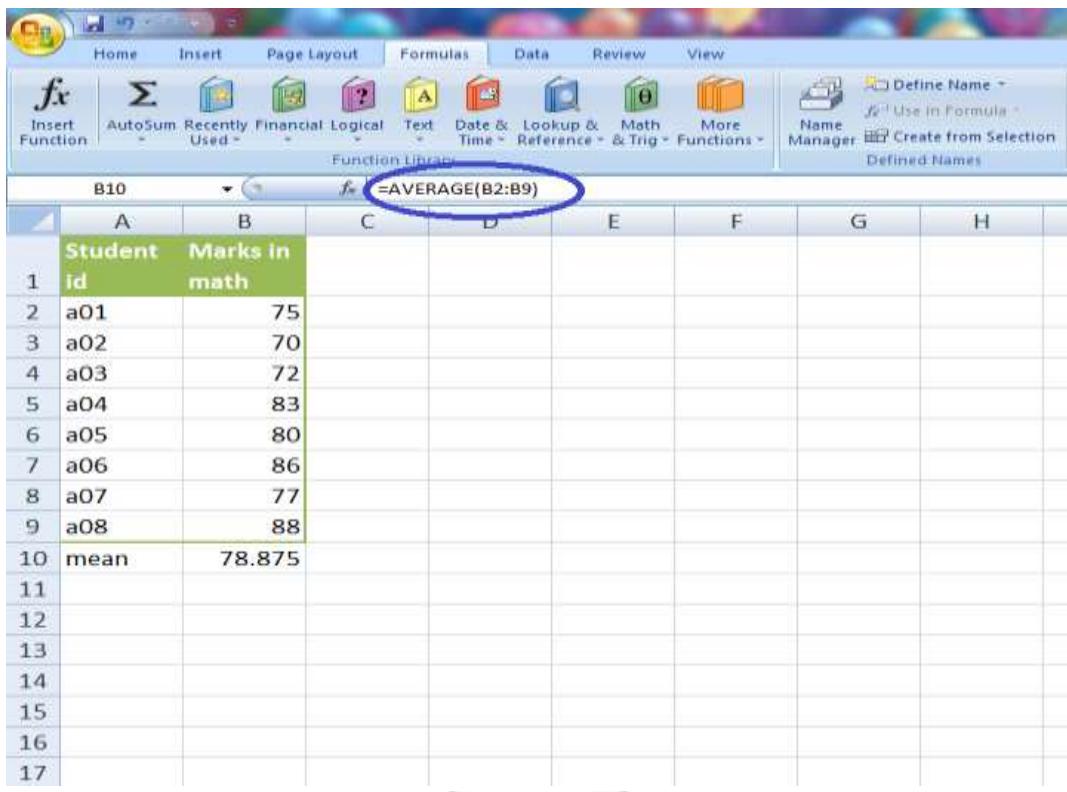
Student id	Marks in math
a01	75
a02	70
a03	72
a04	83
a05	80

a06	86
a07	77
a08	88

Applying the formula we have mean = $\frac{75+70+72+83+80+86+77+88}{8} = \frac{631}{8} = 78.875$

Let's find out mean of a dataset in Excel

We have function AVERAGE(start value : end value)



	A	B	C	D	E	F	G	H
1	Student id	Marks in math						
2	a01	75						
3	a02	70						
4	a03	72						
5	a04	83						
6	a05	80						
7	a06	86						
8	a07	77						
9	a08	88						
10	mean	78.875						
11								
12								
13								
14								
15								
16								
17								

MEDIAN

The median of a numerical data set is the value in the middle most when the data is arranged in ascending or descending order. It is the halfway point in a data set also known as the positional average. We know in triangles median is a line that divides the opposite side into equal lengths giving area being divided exactly 50%.50% of the values of the distribution are less than the median (left of median) and 50% are greater than the median (right of median). Hence median is considered as a measure of central tendency. Median is less affected by Outliers or extreme values. Median can be also used as a measure of position (quartiles and deciles). Median can also be termed as the second quartile or 50th percentile. It is also affected by sampling fluctuations. Used to summarize ordinal or highly skewed interval or ratio scores interval or ratio scores.

To calculate median first we need to

- Arrange the values in ascending or descending order.

- check the number of values n or N is even or odd .
- If N is ODD

median = value in the middle position

Example: {3, 8, 10, 7, 5, 14, 2, 9, 8}

After arranging the order we have set as {2, 3, 5, 7, 8, 8, 9, 10, 14}

Median = middle value =8

- If N is Even

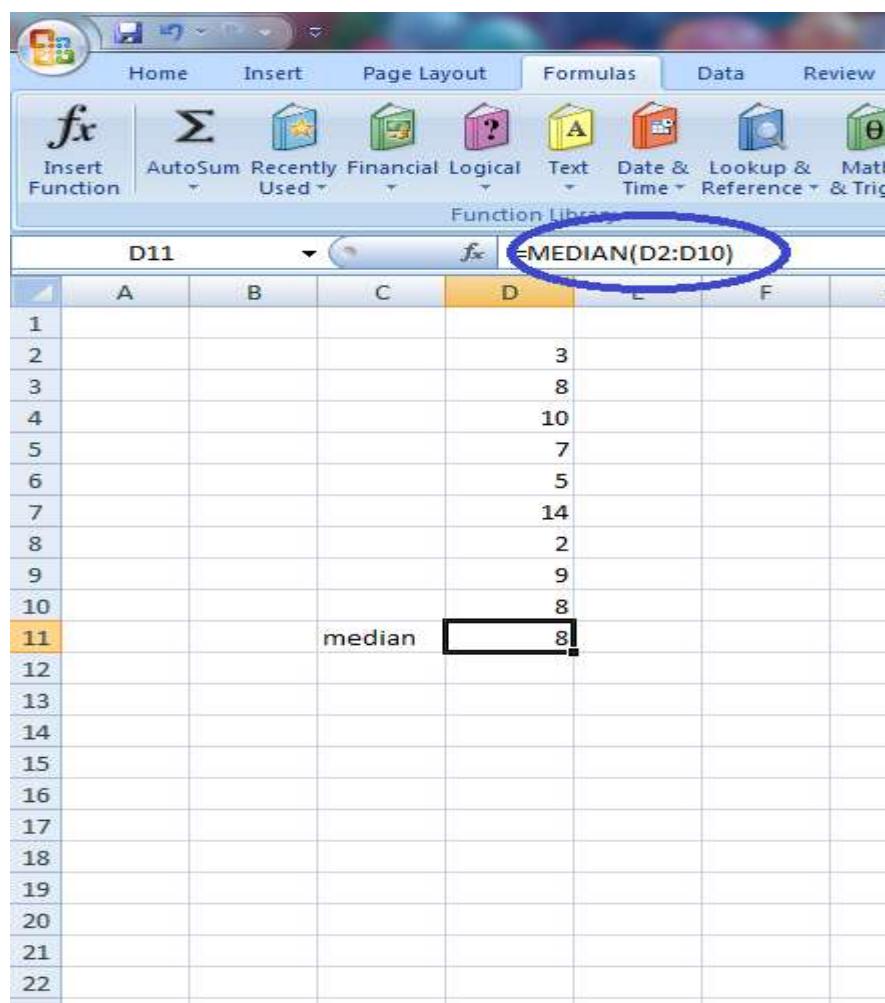
Median = average of values left after dividing set in to two equal groups.

Example:{26, 31, 21, 29, 32, 26, 25, 28}

After arranging the order we have {21, 25, 26, 26, 28, 29, 31, 32 }

$$\text{Median} = \frac{26+28}{2} = 27$$

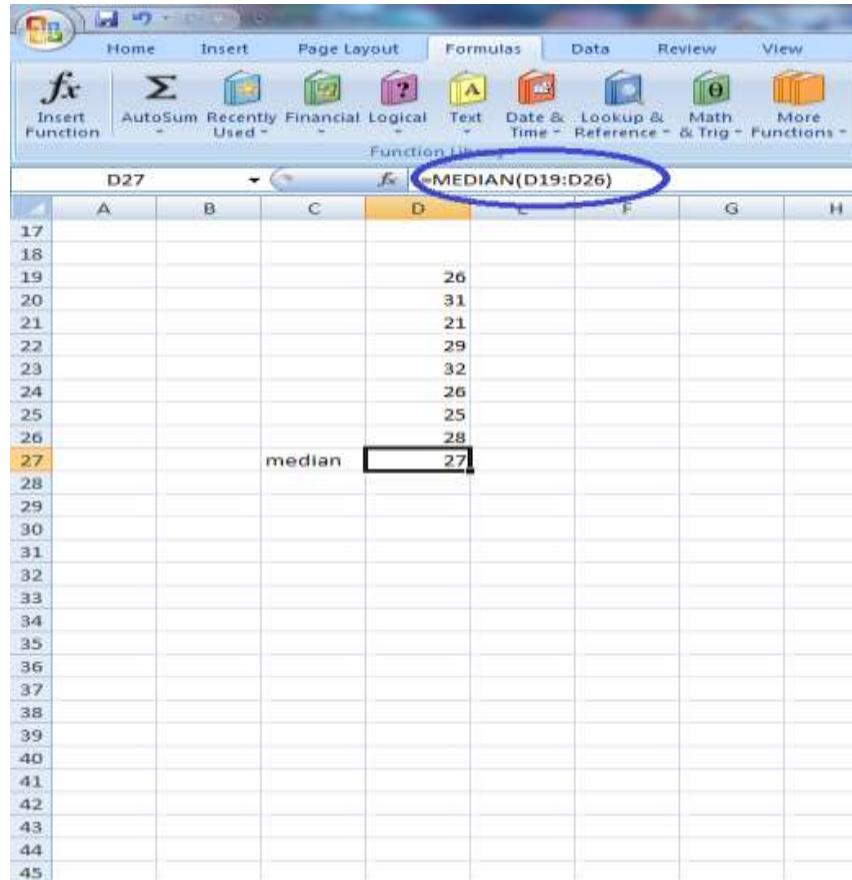
In excel, we have formula MEDIAN(start value : end value)



The screenshot shows an Excel spreadsheet with the following data in column D:

	A	B	C	D	E	F	G
1							
2					3		
3					8		
4					10		
5					7		
6					5		
7					14		
8					2		
9					9		
10					8		
11			median	8			
12							
13							
14							
15							
16							
17							
18							
19							
20							
21							
22							

The formula `=MEDIAN(D2:D10)` is entered in cell D11. The cell D11 is highlighted with a blue oval.

**MODE**

The Leader in Software Training
9963799240 / 7730997544

Along with mean and median, Mode is one of the central tendencies of data distribution. It does not involve much of tedious computations and can be found by easy observation of occurrences of data values. When data is tightly clustered around one or two values, Mode is the most meaningful average. Mode is the value that occurs most often. Because of the highest number of occurrence in the data set, Mode can be considered as the most typical value in the data set and hence a measure of central tendency. Mode can be easily quoted by sheer counting or plotting the frequencies of each item. Mode is not affected by extreme values. The advantage of mode over the other two measures of central tendencies is it can be found for non numerical data sets.

The highest point of the relative frequency histogram corresponds to the mode of the data set. Always exactly one mean and one median exists for any data set. But not the mode because several different values could occur with the highest frequency. It may even happen that every value occurs with the same frequency; in such case the concept of the mode does not make much sense.

A data set with only one greatest frequency value is termed as Unimodal, while data sets which have the same greatest frequency for two values are called Bimodal. When

the data set has more than two values occurring with the same greatest frequency, then the distribution is described as multimodal and each of such values is a mode.

When all the data values have the same frequency (occurring only once), then the data set is said to have no mode.

The class with highest frequency for grouped data with given frequencies is known as the modal class.

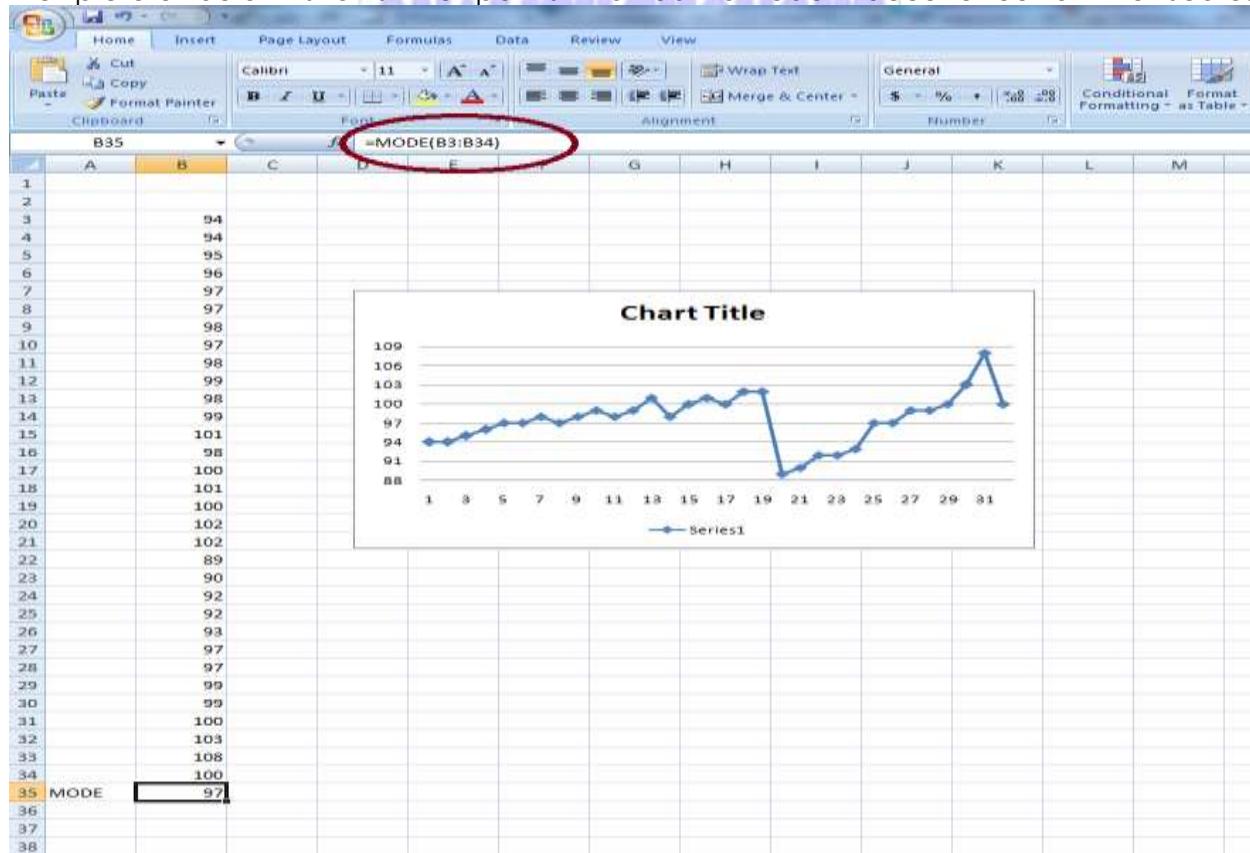
The mode for small data sets can be determined by plotting the occurrences on a number line.

Limitation of mode is it is not well defined, not based on all items and is affected by sampling fluctuations.

Example: The following are points scored by a Basket ball team during the games played in a season.

94, 94, 95, 96, 97, 97, 97, 98, 98, 98, 98, 99, 100, 101, 102, 89, 90, 92, 92, 92, 93, 97, 97, 99, 99, 100, 103, 108.

The picture below shows the points marked for each occurrence of the score.



It can be seen that the point 97 has occurred 5 times in the season which is the highest.

RANGE AND INTER-QUARTILE RANGE

Range is defined as the difference between the maximum value and minimum value in a data set. The minimum and maximum values are useful to know, and helpful in identifying outliers, but the range is extremely sensitive to outliers and not very useful as a general measure of dispersion in the data. We know that measures of central tendencies at one point have the issue with the outliers so, to overcome this we can look at the range of the data after dropping values from each end.

$$\text{Range} = \text{Maximum value} - \text{Minimum value}$$

A common measurement of variability is Inter-Quartile Range which is the difference between the 25th percentile and the 75th percentile. The data set having more variables has the larger IQR.

The first quartile and the third quartile, and these are often labeled Q1 and Q3, respectively.

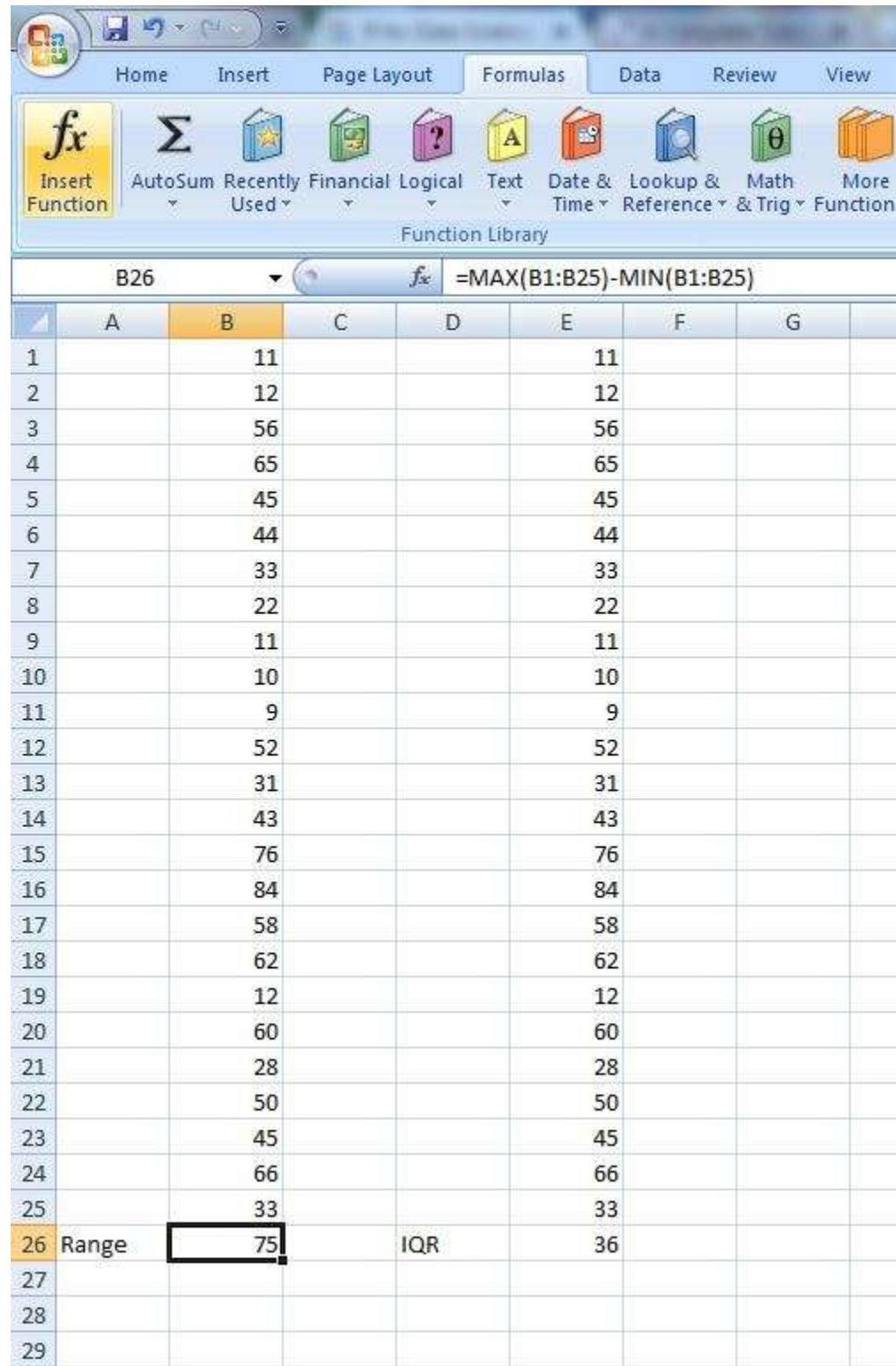
It is calculated as difference between Q3 and Q1.

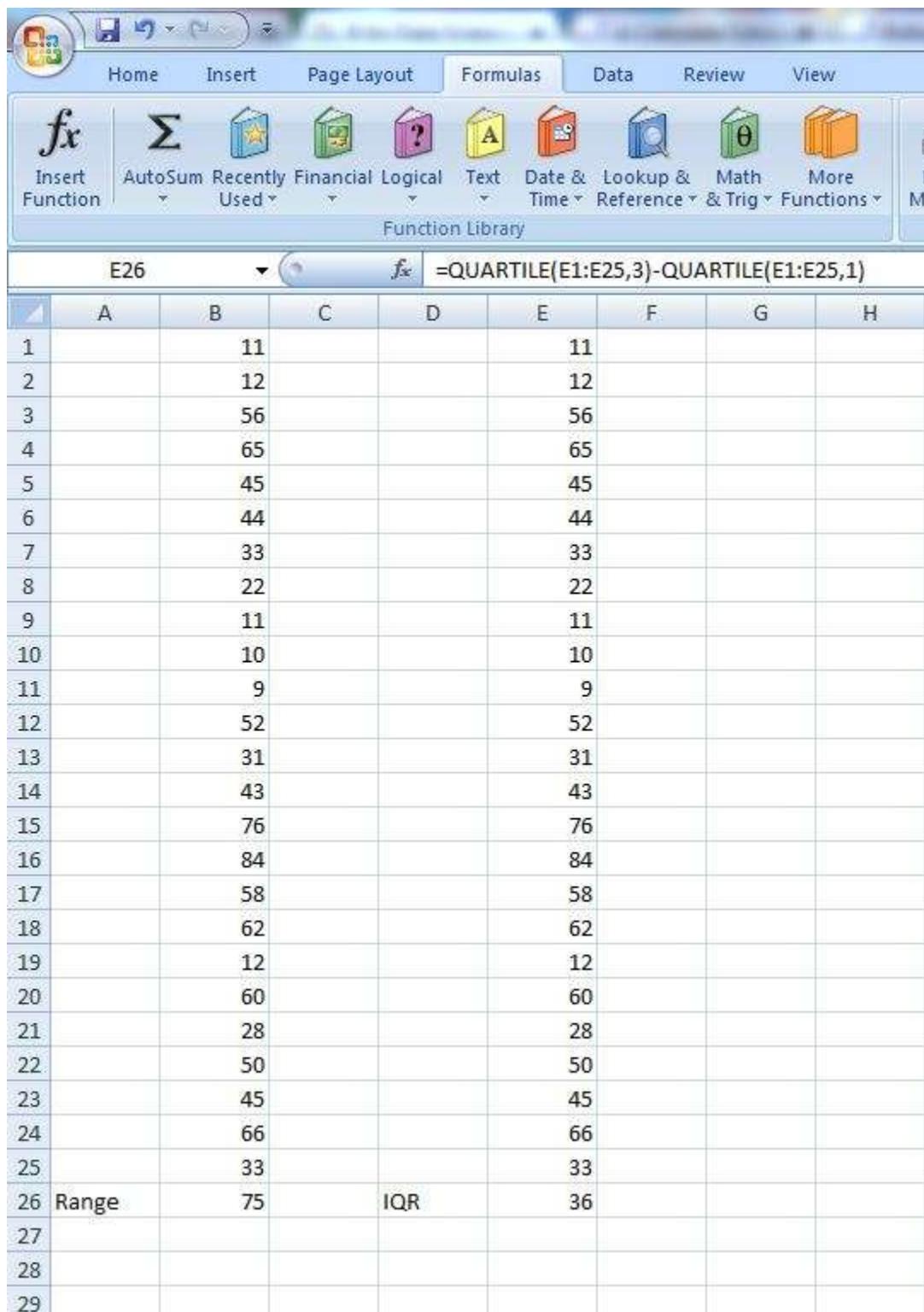
$$\text{IQR} = \text{Q3} - \text{Q1} \text{, where Q1 is 25}^{\text{th}} \text{ percentile and Q3 is 75}^{\text{th}} \text{ percentile.}$$

The IQR is used to plot box plots which are graphical representations of a distribution. The length of the box in a box plot is IQR. For a symmetric distribution, the median equals the midline value is nothing but the average of the first and third quartiles, hence half of the IQR equals the median absolute deviation (MAD).

The half of the IQR is termed as quartile deviation or semi-interquartile range.

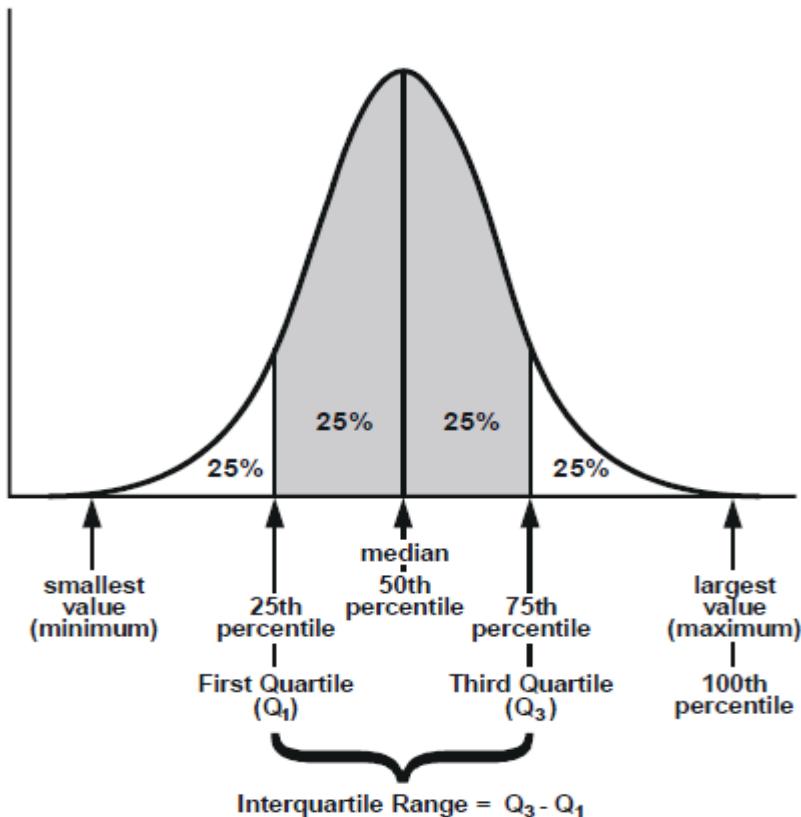
In Excel, the formula "`=QUARTILE(A1:A25, 3)-QUARTILE(A1:A25, 1)`" will calculate the interquartile range and "`=MAX(A1:A25)-MIN(A1:A25)`" will find the range.





The screenshot shows a Microsoft Excel spreadsheet. The formula bar at the top displays the formula $=\text{QUARTILE}(\text{E1:E25},3)-\text{QUARTILE}(\text{E1:E25},1)$. The main table consists of two columns of data. Column A contains values from 1 to 25. Column B contains values from 11 to 36. Row 26 is labeled 'Range' and contains the values 75, IQR, and 36. The Excel ribbon is visible at the top, showing the 'Formulas' tab is selected. The 'Function Library' dropdown is open, showing categories like AutoSum, Recently Used, Financial, Logical, Text, Date & Time, Lookup & Reference, Math & Trig, and More.

	A	B	C	D	E	F	G	H
1		11			11			
2		12			12			
3		56			56			
4		65			65			
5		45			45			
6		44			44			
7		33			33			
8		22			22			
9		11			11			
10		10			10			
11		9			9			
12		52			52			
13		31			31			
14		43			43			
15		76			76			
16		84			84			
17		58			58			
18		62			62			
19		12			12			
20		60			60			
21		28			28			
22		50			50			
23		45			45			
24		66			66			
25		33			33			
26	Range	75	IQR		36			
27								
28								
29								



VARIANCE

the average of the squared differences from the mean is known as the variance. Smaller the variance, closer the data points to the mean and from each other. Higher the variance indicates that the data points are very spread out from the mean and from each other.

The population Variance σ^2 of a discrete set of numbers is :

$$\sigma^2 = \frac{\sum_{i=1}^N (X_i - \mu)^2}{N}$$

where X_i is the i^{th} unit, starting from the first observation to the last

μ - population mean

N - number of units in the population

$$\text{The Variance of a sample } s^2 = \frac{\sum_{i=1}^n (X_i - \bar{x})^2}{n-1}$$

where X_i is the i^{th} unit, starting from the first observation to the last

\bar{x} - sample mean

n - number of units in the sample.

This is known as Bessel's correction.

To calculate manually:

- Find the mean of the set.
- Subtract each value from the mean to find its distance from the mean.
- Square all distances.
- Add all the squares of the distances.
- Divide by the number of pieces of data.

The sum of actual deviations having both positive and negative values from the mean is zero. We use sum of squared deviations instead of the actual differences. Since square of the deviations is always positive, the variance is a positive for all data distributions.

Let's check out variance in excel:

We have function VARP for population variance

	A	B	C	D	E	F	G
1							
2							
3							
4							
5							
6							
7				3			
8				8			
9				6			
10				10			
11				12			
12				9			
13				11			
14				10			
15				12			
16				7			
17		mean		8.8			
18							
19		variance		7.36			
20							
21							

STANDARD DEVIATION

One of the measures of spread is Standard deviation. If in a normal distribution, the mean and standard deviation are known, it is easy to calculate percentile rank of any given score. The standard deviation is a statistic that tells you how tightly all the values in dataset are clustered around the mean. We can remember like the square root of the variance is standard deviation or mean of the mean.

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2}$$

The "Population Standard Deviation":

$$s = \sqrt{\frac{1}{N-1} \sum_{i=1}^N (x_i - \bar{x})^2}$$

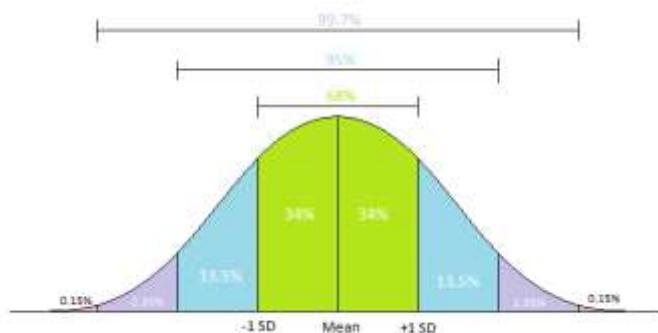
The "Sample Standard Deviation":

Looks complicated, but the important change is to divide by **N-1** (instead of **N**) when calculating a Sample Variance.

- For each value x , subtract the mean from x ,
- Multiply that result by itself.
- Sum up all those squared values.
- Then divide that result by n (population) or $(n-1)$ (sample).

When the values are pretty tightly bunched together and the bell-shaped curve is steep, the standard deviation is small.

When the values are spread apart and the bell curve is relatively flat, that tells you have a relatively large standard deviation.

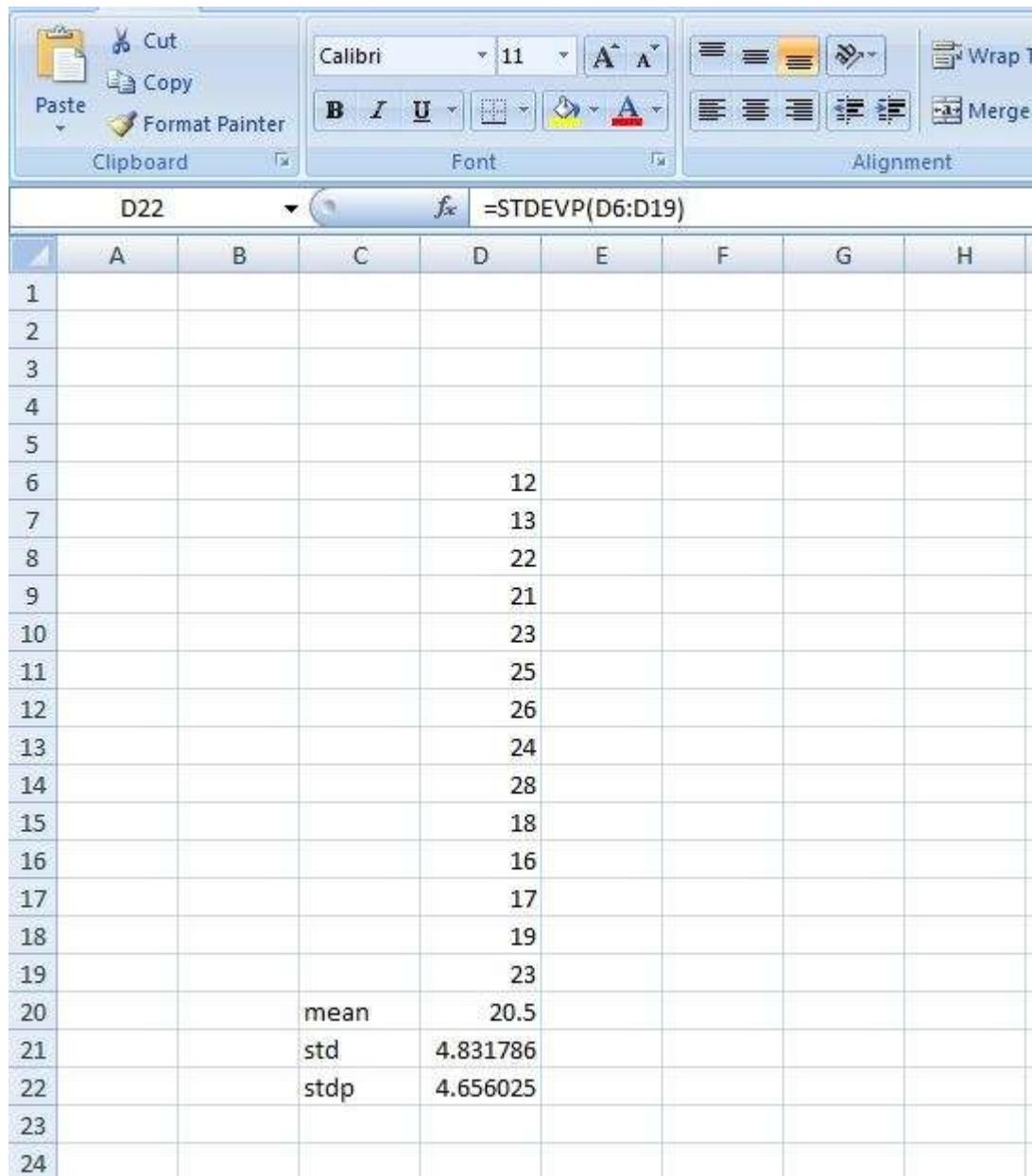


One standard deviation away from the mean in either direction on the horizontal axis (-1SD to +1SD) accounts to about 68% of the people in the group. Two standard deviations away from the mean(-2SD to +2SD) can account to roughly 95% of the

people. Three standard deviations (all the shaded areas) accounts to 99.7% of the people.

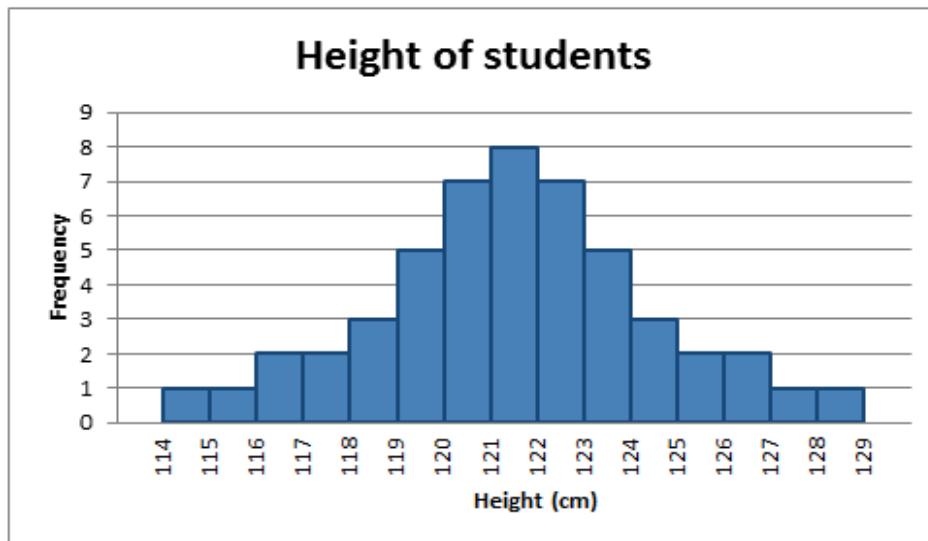
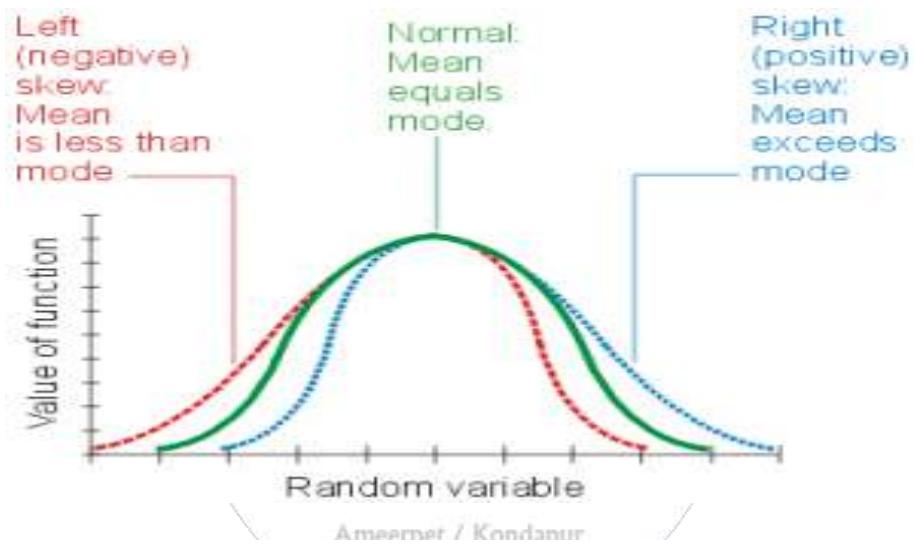
In highly-skewed distributions, standard deviation is not considered as a good measure of spread. In such cases, along with Standard deviation, semi-interquartile range must be taken into consideration.

In Microsoft Excel, `STDEV(A1:Z99)` for sample and `STDEVP(A1:Z99)` if you want to use the "biased" or "n" method for population.



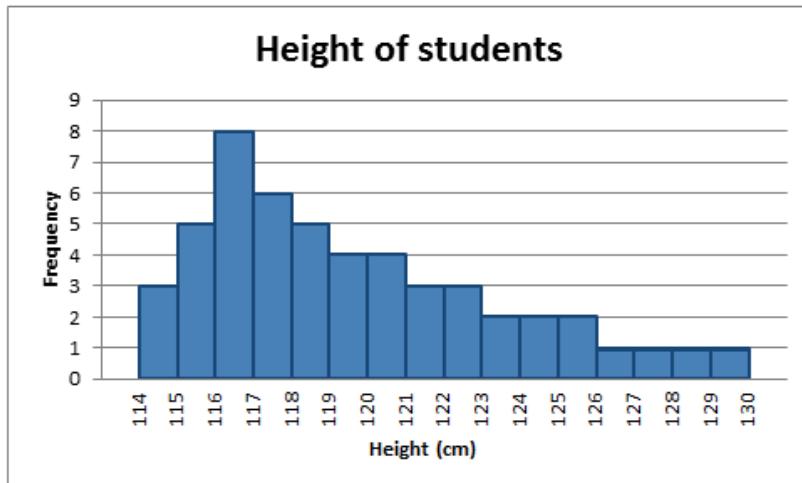
SKEWNESS

A fundamental task in any statistical analyses is to characterize the location and variability of a data set. A distribution of data item values can be symmetrical or asymmetrical. Skewness is asymmetry in a statistical distribution, where the curve appears distorted or skewed either to the left or to the right. Skewness can be quantified to what extent a distribution differs from a normal distribution. In a normal distribution, the graph appears as a symmetrical "bell-shaped curve (the tails on either side of the curve are exact mirror images of each other)." At the maximum point on the curve the mean, median, mode are equal (in Normal distribution). Therefore are all said to be appropriate measure of central tendency.

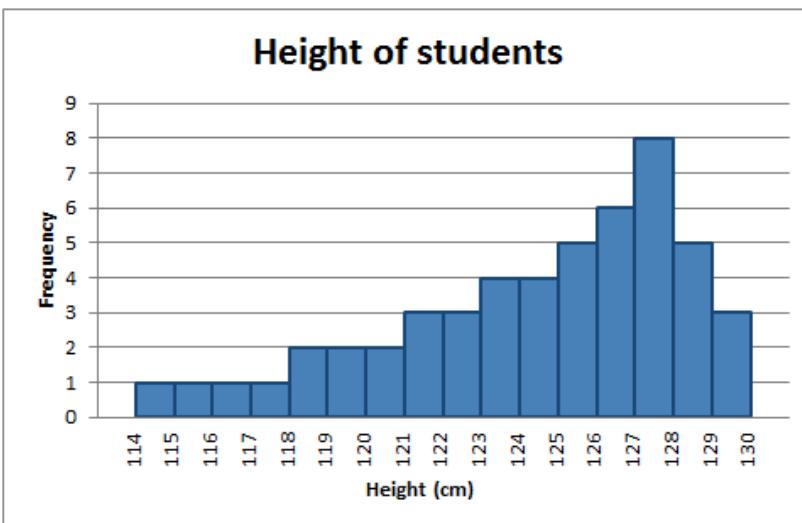


When a histogram is constructed for skewed data it is easy to identify skewness by the shape of the distribution.

A distribution is positively skewed when the tail on the right side of the histogram is longer than the left side. Most of the values tend to cluster toward the left side of the x-axis with increasingly fewer values at the right side of the x-axis



A distribution is said to be negatively skewed when the tail on the left side is longer than the right side of the histogram



According to Karl Pearson, coefficient of skewness can be calculated as

$$\text{Mode skewness coefficient (first skewness coefficient)} = \frac{\text{mean} - \text{mode}}{\text{standard deviation}}$$

$$\text{Median skewness coefficient (second skewness coefficient)} = \frac{3(\text{mean} - \text{mode})}{\text{standard deviation}}$$

To calculate "Skewness" (the amount of skew), we use the SKEW() function in Excel.

	A	B	C	D	E
1					
2					
3					
4					
5					
6			12		
7			12		
8			13		
9			15		
10			16		
11			13		
12			14		
13			18		
14			17		
15			19		
16			20		
17			21		
18			12		
19			15		
20			14		
21			13		
22	mean		15.25		
23	mode		12		
24	std		2.955221		
25	skewness coefficient		0.695219		
26					
27					
28					
29					
30					

KURTOSIS

The Leader in Software Training
9963799240 / 7730997544

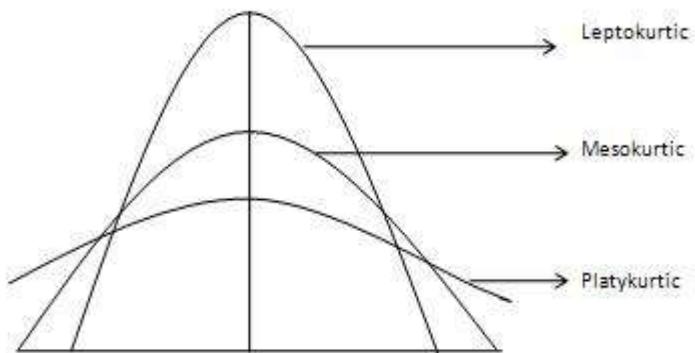
Kurtosis is a measure of thickness of a variable distribution found in the tails. The outliers in the given data have more effect on this measure. Moreover, it does not have any unit. The kurtosis of a distribution can be classified as leptokurtic, mesokurtic and platykurtic.

Leptokurtic distributions are variable distributions with wide and heavier tails and have positive kurtosis(kurtosis >0). As the name tells us lepto means slender. Examples of leptokurtic distributions are Student's t-distribution, Rayleigh distribution, Laplace distribution, exponential distribution, Poisson distribution and the logistic distribution.

Platykurtic distributions have narrow and lighter tails and thus have negative kurtosis(kurtosis <0). As the name tells us platy means broad. Examples of platykurtic distributions are continuous or discrete uniform distributions, the raised cosine distribution, the Bernoulli distribution.

Mesokurtic distributions (such as the normal distribution) have a kurtosis of zero. Most often, kurtosis is measured against the normal distribution. For example, the binomial distribution is mesokurtic.

Diagrammatically, shows the shape of three different types of curves.



The normal curve is called Mesokurtic curve. If the curve of a distribution is more peaked than a normal or mesokurtic curve then it is referred to as a Leptokurtic curve. If a curve is less peaked than a normal curve, it is called as a platykurtic curve.

Formula :

$$\text{Kurtosis} = \frac{\sum_{i=1}^N (X_i - \bar{X})^4}{N s^4}$$

where,

\bar{X} is the mean,

s is the standard deviation

and N is the sample size

9963799240 / 7730997544

The sample kurtosis is a useful measure of whether there is a problem with outliers in a data set. Larger kurtosis indicates a more serious outlier problem, which helps researcher to choose alternative statistical methods.

CORRELATION

Correlation is used to find relationships between quantitative variables or categorical variables. A positive correlation indicates the extent to which those variables increase or decrease in parallel; a negative correlation (inverse correlation) indicates the extent to which one variable increases as the other decreases.

The degree of association is measured by a correlation coefficient, denoted by 'r' also called as Pearson's correlation coefficient, which is a measure of linear association.

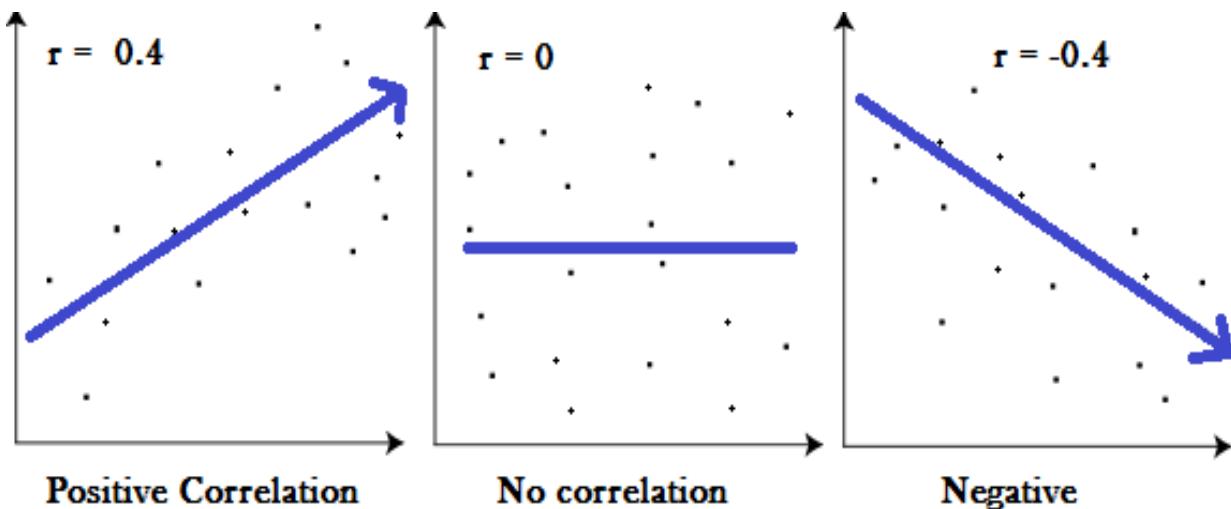
The correlation coefficient varies from + 1 through 0 to - 1. Complete absence of correlation is represented by 0.

When an investigator has collected two series of observations and wishes to see whether there is a relationship between them, first one should construct a scatter diagram. If one set of observations consists of experimental results and the other consists

of observed (independent variable) classification, is measured along the horizontal axis and the experimental results (dependent variable) on the vertical axis.

For absolute values of r ,

- 0-0.19 is regarded as very weak,
- 0.2-0.39 as weak,
- 0.40-0.59 as moderate,
- 0.6-0.79 as strong,
- 0.8-1 as very strong correlation.



Pearson correlation coefficient:

$$r = \frac{\sum XY - \frac{(\sum X)(\sum Y)}{n}}{\sqrt{\left[\left(\sum X^2 - \frac{(\sum X)^2}{n} \right) \left(\sum Y^2 - \frac{(\sum Y)^2}{n} \right) \right]}}$$

- $\sum X$ This simply tells you to add up all the X scores
- $\sum Y$ This tells you to add up all the Y scores
- $\sum X^2$ This tells you to square each X score and then add them up
- $\sum Y^2$ This tells you to square each Y score and then add them up
- $\sum XY$ This tells you to multiply each X score by its associated Y score and then add the resulting products together (this is called a "cross-products")
- n This refers to the number of "pairs" of data you have.

A correlation report also shows a second result of each test which is statistical significance.

In Excel we have the function CORREL(array1, array 2)
For example, =CORREL(A2:A6,B2:B6)

COVARIANCE

The covariance of two variables x and y in a data set is a measure of the directional relationship between them. A positive covariance indicates a positive linear relationship between the variables which move together. A negative covariance indicates that the variables move inversely. Covariance is similar to variance except that we have two variables x and y .

$$\text{Sample covariance, } S_{xy} = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{(n-1)}$$

$$\text{Population covariance, } \sigma_{xy} = \frac{\sum_{i=1}^N (x_i - \mu_x)(y_i - \mu_y)}{N}$$

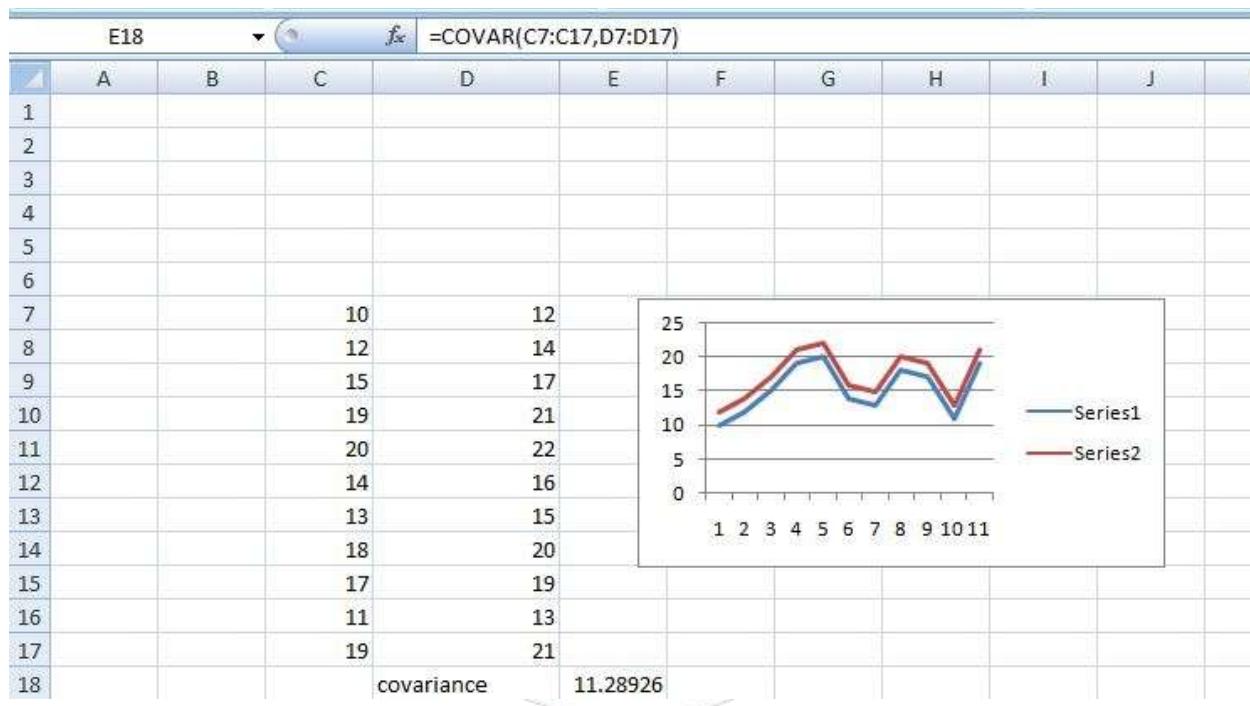
For a scatter plot of two variables the covariance measures how close the scatter is to a line. Positive covariance corresponds to upward-sloping scatter plots. Negative covariance corresponds to downward-sloping scatter plots. Covariance is scale dependent and has units. Nonlinear dependencies have zero covariance. Independence implies zero covariance. The value for a perfect linear relationship depends on the data because covariance values are not standardized.

There is confusion in understanding terms covariance and correlation. Here are some of the differences:

- The correlation coefficient is a function that uses covariance. The correlation coefficient is the covariance divided by the product of the respective standard deviations of the variables.
- Covariance is a measure of a correlation while correlation is a scaled version of covariance.

- Covariance can involve the relationship of two variables or data sets whereas correlation can involve the relationship of several variables.
- Correlation values range from +1 to -1. But, covariance values can exceed this scale.
- The Spearman correlation coefficient tells how close or far two variables are independent from each other. The covariance calculation tells you how much two variables tend to change together.

In Excel, we have the function COVAR (array 1, array 2) where arrays are the 2 different data sets.



PROBABILITY

The probability theory is very helpful for making predictions. In research investigation, estimates and predictions form an important part. Using statistical methods, we estimate for the further analysis. The role of probability in modern science is simply a substitute for certainty. Probability can be defined in terms of a random process giving rise to an outcome. Rolling a die or flipping a coin is a random process which gives rise to an outcome.

Probability of occurrence of an event A is

$$P(A) = \frac{\text{Number of favourable outcomes}}{\text{Total number of equally likely outcomes}}$$

Choose a random number from 1 to 5. What is the probability of each outcome?

The possible outcomes are 1, 2, 3, 4 and 5.

$$P(1) = \frac{\text{no of ways to choose a 1}}{\text{total numbers}} = \frac{1}{5}$$

Similarly each number can occur $p(2)=p(3)=p(4)=p(5) = \frac{1}{5}$

- An event with probability 0 has no chance of occurring whereas, an event of probability 1 is certain to occur.
- Probability always takes values between 0 and 1 (inclusively).
- It may also be represented as a percentage between 0% and 100%.
- As more observations are collected, the proportion \hat{p}_n of occurrences with a particular outcome converges to the probability p of that outcome.

Here are some of the rules in finding probabilities in different situations:

- Two outcomes are said to be disjoint or mutually exclusive if they both cannot happen.
- If two events, A and B, are mutually exclusive, the probability that A or B will occur is:

$$P(A \text{ or } B) = P(A) + P(B)$$

- If two events, A and B, are non-mutually exclusive, the probability that A or B will occur is:

$$P(A \text{ or } B) = P(A) + P(B) - P(A \text{ and } B)$$

A single random card is chosen from a deck of 52 playing cards. What is the probability of choosing a Queen or a club?

Probabilities:

$$P(\text{queen or club}) = P(\text{queen}) + P(\text{club}) - P(\text{queen of clubs})$$

$$= \frac{4}{52} + \frac{13}{52} - \frac{1}{52}$$

$$= \frac{16}{52}$$

$$= \frac{4}{13}$$

- If two events A and B are independent, the probability of both occurring is:

$$P(A \text{ and } B) = P(A) \cdot P(B)$$

If a coin is tossed and a single 6-sided die is rolled. Find the probability of landing on the head side of the coin and rolling a 5 on the die.

Probabilities:

$$P(\text{head}) = \frac{1}{2}$$

$$P(5) = \frac{1}{6}$$

$$P(\text{head and } 5) = P(\text{head}) \cdot P(5)$$

$$= \frac{1}{2} \cdot \frac{1}{6} = \frac{1}{12}$$



- Conditional probability is $P(B | A)$ read as probability of event B given that event A has already occurred.

$$P(B | A) = \frac{P(\text{A and B})}{P(\text{A})}$$

A math teacher gave her class two tests. 23% of the class passed both tests and 46% of the class passed the first test. What is the percentage of those who passed the first test also passed the second test?

$$P(\text{Second} | \text{First}) = \frac{P(\text{First and Second})}{P(\text{First})} = \frac{0.23}{0.46} = 0.50 = 50\%$$

- If two events, A and B, are dependent, the probability of both occurring is:

$$P(A \text{ and } B) = P(A) \cdot P(B | A)$$

Mr. Parth needs two students to help him with a science demonstration for his class of 16 girls and 12 boys. He randomly chooses one student who comes to the front of the

room. He then chooses a second student from the remaining class. What is the probability that both chosen students are girls?

$$P(G1 \text{ and } G2) = P(G1) \text{ and } P(G2 | G1)$$

$$= \frac{16}{28} \cdot \frac{15}{27}$$

$$= \frac{240}{756}$$

$$= \frac{60}{189}$$

RANDOM VARIABLES

A random variable is the value of the variable which represents the outcome of a statistical experiment within sample space (range of values). It is usually represented by X. The two types of random variables are discrete random variable and continuous random variable.

Discrete random variable is a variable which can take countable number of distinct values.

For example, while tossing two coins, let us consider the random variable(X) to be number of heads observed.

Here, the possible outcomes are {HH, HT, TH, TT} which means the number of heads possible in a single outcome may be 2 heads or 1 head or no heads at all. Hence, the possible values taken by the random variable are 0, 1, and 2 (discrete).

Continuous random variable is a variable which can take infinite number of values in an interval. It is usually represented by the area covered under the curve. Examples are height and weight of the subjects, maximum and minimum temperatures of a particular place.

Let Y be the random variable for the average height of a random group consisting of 25 people, the resulting outcome is a continuous figure since height may be 5 ft or 5.15 ft or 5.002 ft. Clearly, there is an infinite number of continuous possible values for height.

Properties of Random variable

- Let X and Y be two random variables and the constant C. Then CX, X+Y, X-Y are also random variables. Any arithmetic operation with random number results in another random variable.

- If X is a random variable, then $\frac{1}{X}$ and $|X|$ are also said to be random variables.

A random variable's behavior is defined by a probability distribution which is the likelihood that any of the possible values would occur.

Let Z be the random variable which is the number on the top face of a die when it is rolled once. The possible numbers for Z are 1, 2, 3, 4, 5, and 6. $P(1)=P(2)=P(3)=P(4)=P(5)=P(6) = 1/6$ as they are all equally likely to be the value of Z . Note that the sum of all probabilities is 1.

Suppose in a probability distribution where the outcomes of a random event are not equally likely to happen we can still find the probabilities of random variable. Let Y be random variable which is the number of heads we get from tossing two coins, then Y could be 0, 1, or 2. The two coins will flip in 4 possible different ways – TT(no heads), HT(one head), TH(one head), HH(2 heads).

So, $P(Y=0) = 1/4$ as we have only one chance of getting no heads (TT).

Similarly, $P(Y=2) = 1/4(HH)$.

But, $P(Y=1) = 2/4 = 1/2.(HT, TH)$

PROBABILITY DISTRIBUTION

Probability distribution is a measure of random variable's behavior/dispersion which indicates the likelihood of an event or outcome. To represent a probability distribution, we use equations and tables of variable values and probabilities.

To describe probabilities, statisticians use this notation $p(x)$ = the likelihood that random variable which takes a specific value of x .

Based on the type of Random variable, we have two types of distributions:

- For discrete variables- Discrete probability distributions
- For continuous variables- Probability density functions

Discrete Probability Distributions:

Discrete probability functions / probability mass functions (PMF) can assume a discrete number of values. For example, number of coin tosses and counts of events are discrete functions because there are no in-between values. For example, only heads or tails occur in a single coin toss.

Each possible value has non-zero likelihood. Since total probability must be 1, one of the values must occur for each opportunity.

To display discrete distribution having a finite number of values, we can arrange them in a tabular manner with their corresponding probabilities.

Types of Discrete Probability Distribution:

- Binomial distribution is used for experiments whose outcome is in form of binary data, such as coin tosses.
- Poisson distribution is used for quantitative random variables (countable data), such as the count of check-ins per hour at an airport.
- Uniform distribution is used for multiple events having the same probability, such as rolling a die.

The mathematical representation of a discrete probability function, $p(x)$, is a function that satisfies the following properties:

- The probability that x can have a specific value is $p(x)$
 $P[X=x]=p(x)=p_x$, where X is random variable.
- $p(x)$ is non-negative for all Real x i.e., where $0 \leq p(x) \leq 1$
- The sum of $p(x)$ over all possible values of x is 1
$$\sum_j p_j = 1$$

Where, j represents all possible values that x can have and p_j is the probability at x_j .

Continuous Probability Distributions:

Continuous probability functions / probability density functions (PDF) can assume an infinite number of values between any two values such as height, weight, and temperature.

In discrete probability distributions, each particular value has non-zero likelihood (probability), but in continuous distributions, specific values may have a zero probability. For example, the likelihood of measuring a temperature that is exactly 32 degrees is zero. Statisticians say that any individual value has an infinitesimally small probability that is equivalent to zero.

Probabilities for continuous distributions are measured over ranges of values rather than single points. Here we calculate probability whether the likelihood of a value falls within an interval or not.

In discrete distributions, the sum of all probabilities must equal one. Similarly, in continuous distributions, the entire area in a probability plot under the distribution curve must be equal to 1. The proportion of the area under the curve that falls within a range of values along the X-axis represents the probability.

Each probability distribution has parameters that define its shape. Most distributions have between 1-3 parameters. Based on these parameters, the shape of the distribution and all of its probabilities can be established, such as the central tendency and the variability.

The mathematical representation of a continuous probability function, $f(x)$

- The probability that x is between two points a and b is
$$p[a \leq x \leq b] = \int_a^b f(x) dx$$
- It is non-negative for all real x .
- $\int_{-\infty}^{\infty} f(x) dx = 1$

BERNOULLI DISTRIBUTION

A Bernoulli trial is one of the simplest experiments with exactly two possible outcomes, success and failure. For example

- Coin tosses: Number of heads up/number of tails up.
- Births: number of boys/girls born each day.
- Rolling Dice: the probability of two die roll resulting in a double six.

A Bernoulli distribution is a discrete probability distribution in which the random variable (X) takes only two possible values (Bernoulli trial). One possible value is 1 (success) with probability p and another value is 0 (failure) with probability $(1-p)$. Here, p denotes the probability of success.

The Bernoulli probability distribution of the random variable X is given by,

$$P(x) = p^x (1-p)^{1-x}, x = 0, 1$$

It can also be rewritten as,

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

The expected value (MEAN) of a Bernoulli distribution is

$$E(X) = 0 \times (1-p) + 1 \times p = p.$$

The variance of Bernoulli distribution is

$$\text{Var}(X) = E(X^2) - E(X)^2 = [1^2 \times p + 0^2 \times (1-p) - p^2] = p - p^2 = p(1-p).$$

The mode of a Bernoulli distribution (the value with the highest probability of occurring) is

$$\text{Mode} = \begin{cases} 0 \text{ if } 1-p > p \\ 0,1 \text{ if } 1-p = p \\ 1 \text{ if } 1-p < p \end{cases}$$

The Bernoulli distribution can also be defined as the Binomial distribution with $n = 1$.

The Bernoulli distribution is sometimes used to model a single individual experiencing an event like death, a disease, or disease exposure in clinical trials. The occurrence of a disease can be modeled in logistic regression with help of Bernoulli distributions.

BINOMIAL DISTRIBUTION

A Binomial distribution is a discrete probability distribution in which the random variable (X) follows:

- When there are only two possible outcomes of each trial, success and failure.
- Here, probability (success) is p and the probability (failure) is q or $(1-p)$ where either of them remains constant throughout experiment.

- Experiment consisting of 'n' finite number of trials.
- Each trial is independent of the last.
- Outcomes are mutually exclusive and the sum of their probabilities is complementary ($p+q = 1$).

Binomial probability formula is given by

$$P(X) = {}_n^X C \cdot p^x \cdot (1-p)^{n-x}, \text{ where } {}_n^X C = \frac{n!}{X!(n-X)!}$$

The mean of binomial distribution = $E[X] = E[X_1+X_2+X_3+\dots+X_n] = p + \underbrace{p + p + \dots + p}_{n \text{ times}} = np$

For example, we can calculate the probability that two of the next three babies born are male using binomial distribution.

- The variance of binomial distribution is $\text{Var}[X] = np(1-p)$

- Mode =
$$\begin{cases} [(n+1)p] & \text{if } (n+1)p = 0 \text{ or noninteger,} \\ (n+1)p \text{ and } (n+1)p - 1 & \text{if } (n+1)p \in \{1, 2, \dots, n\}, \\ n & \text{if } (n+1)p = n+1 \end{cases}$$

The Binomial distribution with a single trial ($n = 1$), is Bernoulli distribution.

Syntax for calculating binomial distribution using Excel is

`BINOMDIST(number_s,trials,probability_s,cumulative)`

where ,Number_s is the number of successes

Trials is the number of independent trials

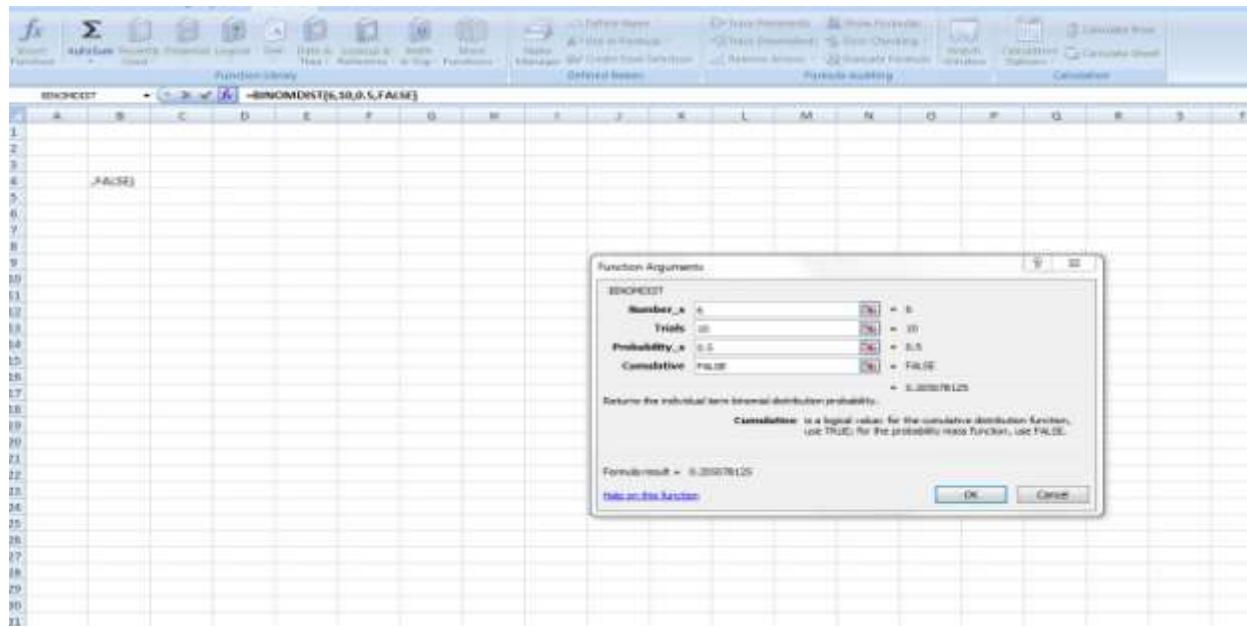
Probability_s is the probability of success in a single trial

Cumulative is a logical value which determines the form of the function. If TRUE, then BINOMDIST returns the cumulative distribution function (at most n successes), if FALSE, it returns the probability mass function (exactly n successes).

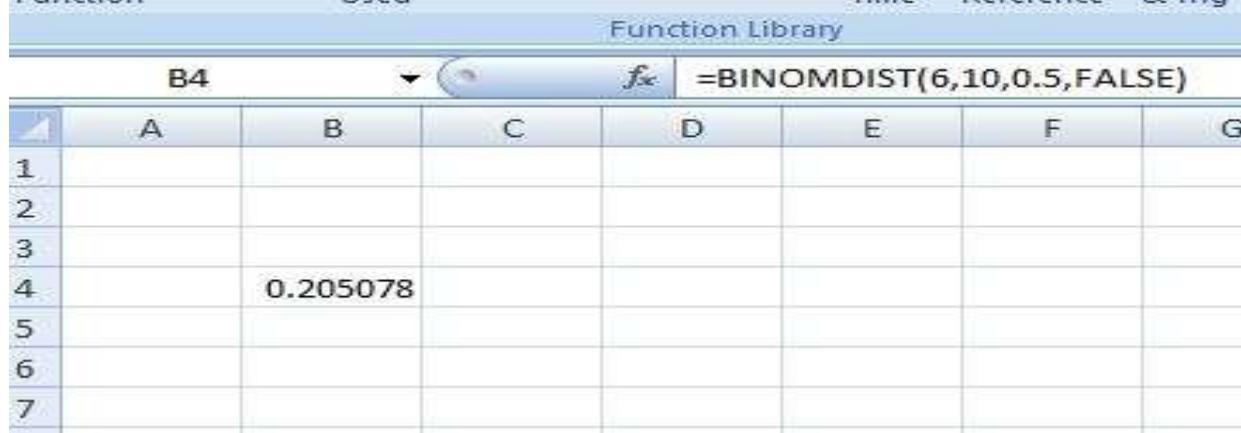
For example:

If a coin is tossed 10 times. The probability of getting exactly 6 heads is ?

Here, number_s = 6 , Trails = 10, Probability_s = $\frac{1}{2} = 0.5$ (probability of head/ tail in a single toss of a coin is $\frac{1}{2}$), Cumulative = FALSE



The probability of getting exactly 6 heads = 0.20507



GEOMETRIC DISTRIBUTION

Consider a sequence of Bernoulli trials (failure and success), the geometric distribution is used to find the number of failures before the first success. For a geometric distribution with probability of success, the probability that exactly x **failures** occur before the first success is

$$P(X=x) = (1 - p)^x p \text{ for } x = 0, 1, 2, 3, \dots$$

The geometric distribution is the only discrete distribution with the memory less property. The successive probabilities in this distribution form a geometric series, hence the name to the distribution.

- In baseball, a geometric distribution is useful in analyzing the probability of a batter earning a hit before three strikes, a success within 3 trials.
- In cost-benefit analyses, whether to fund research trials that, if successful, will earn the company some estimated profit, getting a success before the cost outweighs the potential gain.

The probability mass function, the probability that the x^{th} trial (out of x trials) is the first success is

$$f(x) = P(X=x) = (1-p)^{x-1} p \text{ where } 0 < p < 1, x = 1, 2, 3..$$

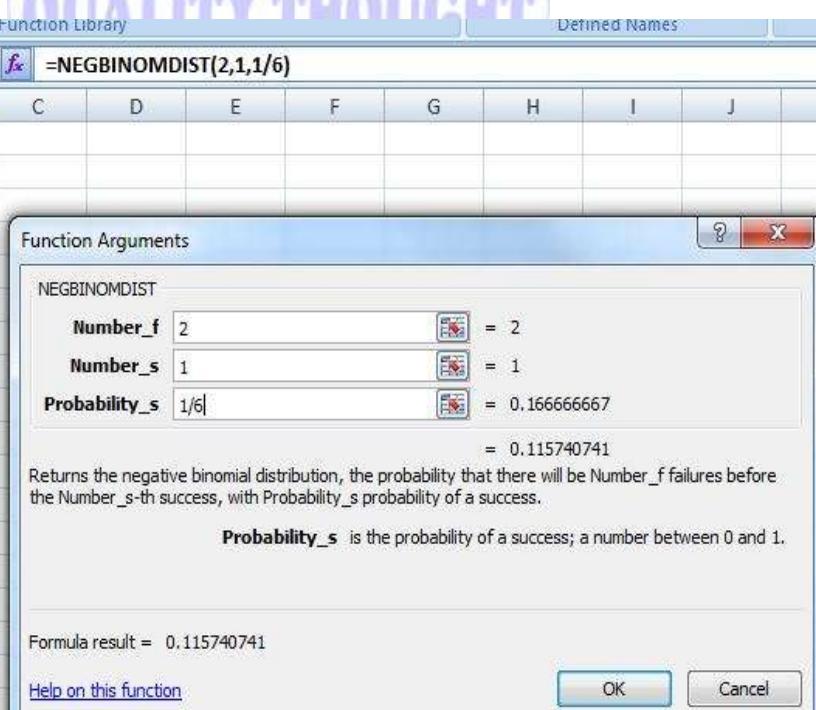
$$\text{MEAN, } \mu = E(X) = \frac{1}{p}$$

$$\text{Variance, } \sigma^2 = \text{Var}(X) = \frac{1-p}{p^2}$$

In Excel, we use the function `NEGBINOMDIST(number_f, number_s, probability_s)` where

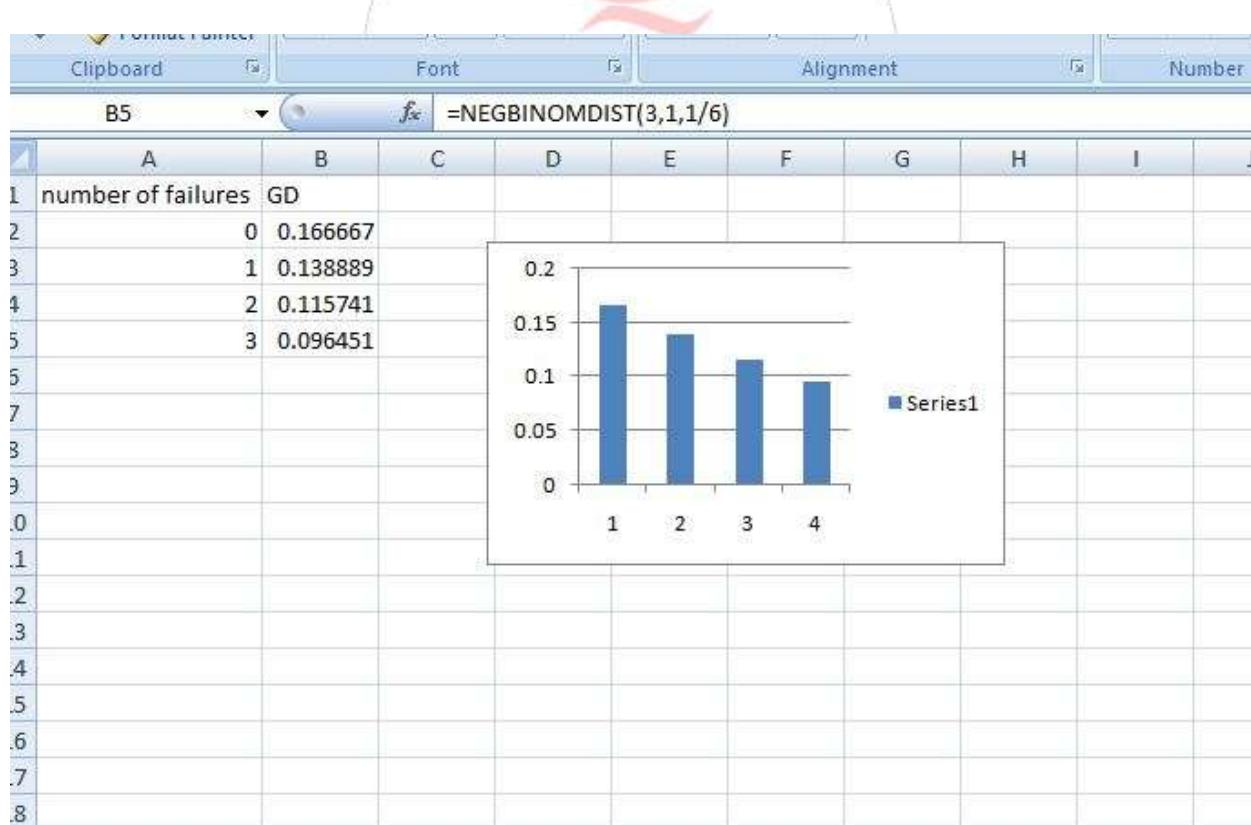
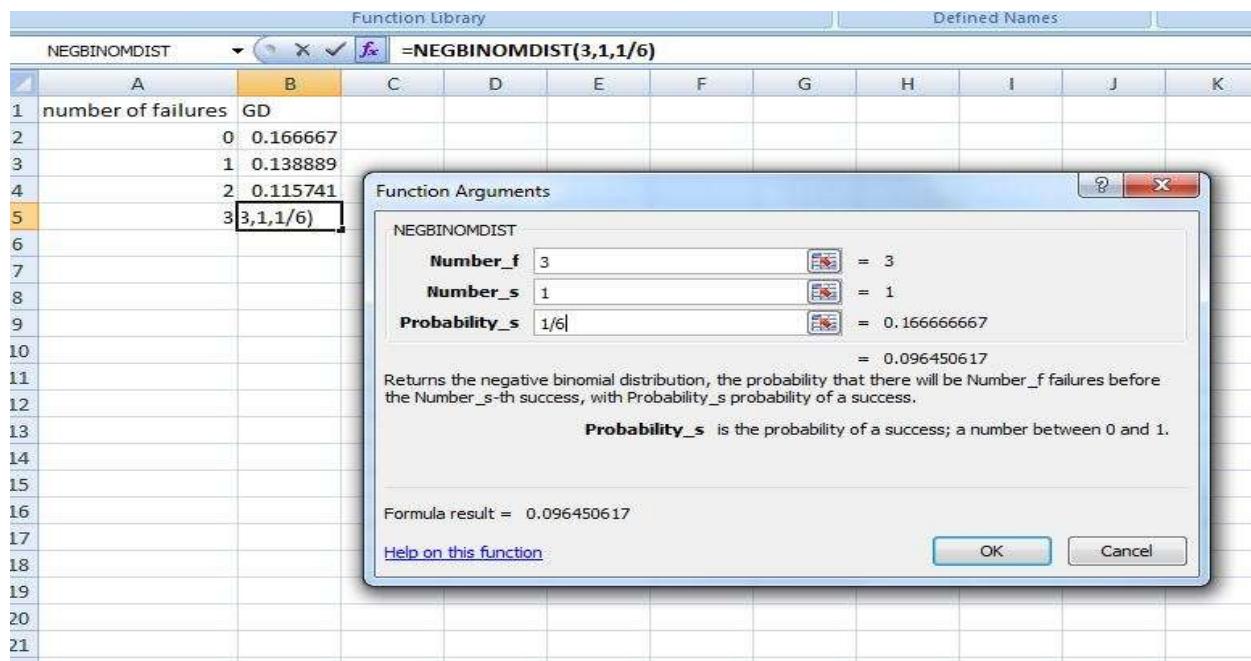
Number_f - number of failures , Number_s - the threshold number of successes(for geometric distribution,1) , Probability_s - the probability of a success on each trial.

For example, A die is rolled until a 1 shows up. Using the function, resulting geometric distribution is shown graphically



	A	B	C	D	E	F	G	H	I	J	K
1	number of failures	GD									
2		0	0.166667								
3		1	0.138889								
4		2	2,1,1/6)								
5		3									
6		4									
7		5									
8											
9											
10											
11											
12											
13											
14											
15											
16											
17											
18											
19											
20											
21											

Similarly, for 3 trials



HYPER GEOMETRIC DISTRIBUTION

In statistics, a distribution where selections are made from two groups without replacing members of the groups is known as **Hyper geometric distribution**. Hyper geometric distribution is the probability distribution of a hyper geometric random variable.

For example,

- balls in an urn - either red or green
- a batch of components - either good or defective
- a population of people - either male or female
- a population of animals in zoo - either tagged or untagged
- voters - either democrats or republicans

A hyper geometric random variable X has the following probability distribution:

$$P(X=k) = \frac{\binom{K}{k} \binom{N-K}{n-k}}{\binom{N}{n}} = \frac{\frac{K!}{k!(K-k)!} \frac{(N-K)!}{(n-k)!(N-K-(n-k))!}}{\frac{N!}{n!(N-n)!}},$$

Where N - population size,

K - Number of successful states in the population,

n - Number of draws or events,

k - Number of successes.

Properties of a hypergeometric experiment are:

- A sample size of n is collected from a population of N without replacement.
- If k is defined as successes in the population N , then $(N-k)$ items will be defined as failures.

$$\text{Mean} = \frac{(n \times k)}{N}$$

$$\text{Variance} = \frac{n \times k \times (N-k) \times (N-n)}{N^2 \times (N-1)}$$

There are 52 cards in a deck. Find the probability of getting randomly 1 red card out of two cards chosen without replacement.

total population, $N = 52$ (total cards in a deck)

sample population, $n = 2$ (number of cards chosen)

number of successes in total population, $K = 26$ (number of red cards)

number of successes in draws, $k = 1$ (number of red cards to be chosen)

$$P(X=1) = \frac{\binom{K}{k} \binom{N-K}{n-k}}{\binom{N}{n}} = \frac{\binom{26}{1} \binom{26}{1}}{\binom{52}{2}} = \frac{26 \times 26}{26 \times 51} = \frac{26}{51} = 0.5098$$

Let us solve this example using Excel function,

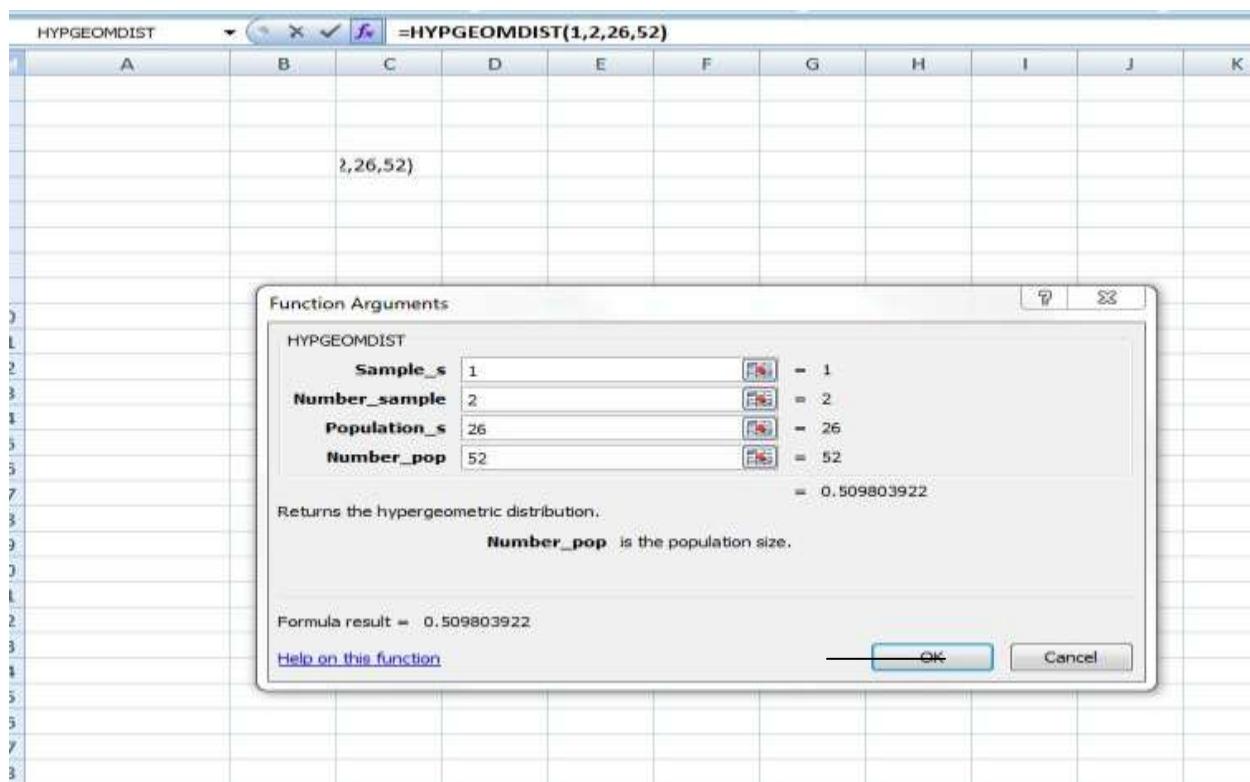
HYPGEOMDIST (sample_s, number_sample, population_s, number_population)

Sample_s is the number of successes in the sample(k).

Number_sample is the size of the sample(n).

Population_s is the number of successes in the population(K).

Number_population is the population size(N).



POISSON DISTRIBUTION

A discrete probability distribution that gives the probability of a given number of events k occurring in a fixed interval of time is known as Poisson distribution. The Poisson distribution is used to calculate the probabilities of number of successes based on the mean number of successes. In a Poisson distribution, events are assumed to occur with a known constant rate, μ , independent of the time since the last event.

$$P(X=x) = \frac{\mu^x e^{-\mu}}{x!}$$

Where e is the base of natural logarithm as in $\log_e n$ (2.7183)
 μ is the mean number of "successes"
 x is the number of "successes"

- o The events occur in disjoint intervals (non-overlapping).
- o Two or more events cannot occur simultaneously.
- o Each event occurs at a constant rate.
- o Mean = μ .
- o Variance = μ .

For example, the mean number of calls to a fire station on a weekday is 8. What is the probability that there would be 11 calls on a given weekday?

$$P = \frac{e^{-8} 8^{11}}{11!} = 0.072$$

Let us solve this example in Excel using the function, $\text{POISSON}(x, \text{mean}, \text{cumulative})$

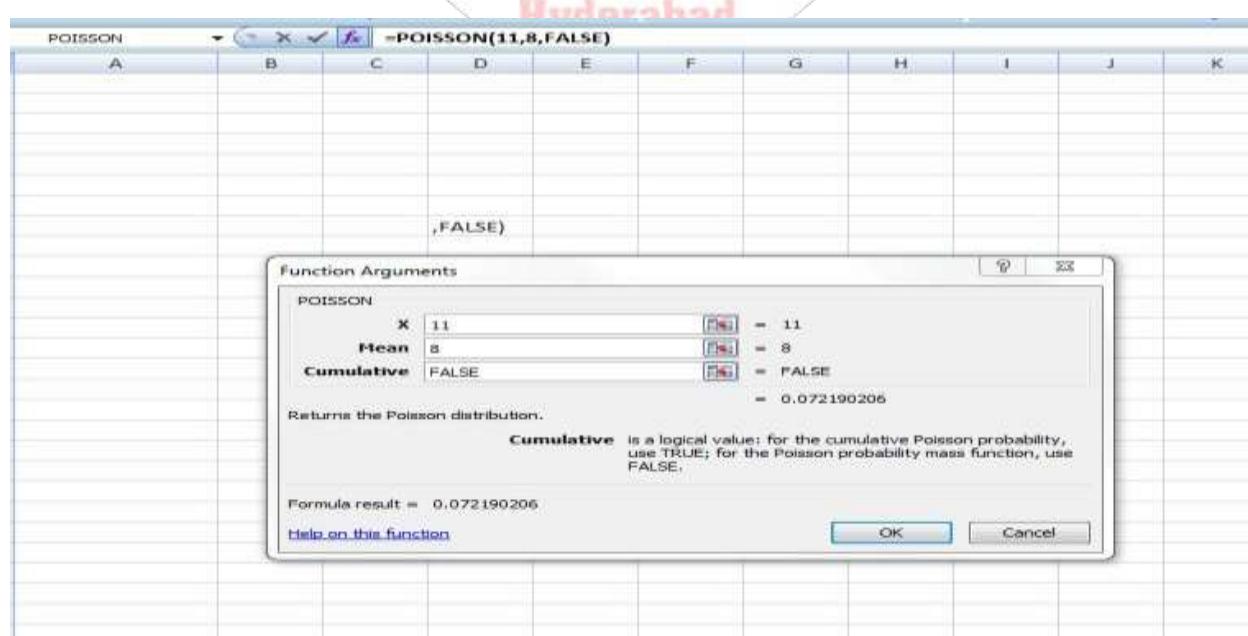
X - number of events (11).

Mean - expected numeric value(8).

cumulative is TRUE, if the number of random events occurring will be between 0 and x inclusive; FALSE, if the number of events occurring will be exactly x.

9963799240 / 7730997544

Here for this example we need cumulative to be FALSE.



POISSON

=POISSON(11,8,FALSE)

Function Arguments

POISSON

X	11	= 11
Mean	8	= 8
Cumulative	FALSE	= FALSE

Formula result = 0.072190206

Help on this function

OK Cancel

EXPONENTIAL DISTRIBUTION

The exponential distribution is a continuous memory less distribution that describes the time between events in a Poisson process. The continuous analogue of the geometric distribution gives exponential distribution. The exponential distribution models time between successive events over a continuous time interval, whereas the Poisson distribution deals with events that happen over a fixed period of time. The exponential distribution is mostly used for testing product reliability which deals with the amount of time a product lasts. For example, the amount of time (beginning now) until an earthquake occurs, the length (in minutes) of long distance business telephone calls, and the amount of time (in months) a car battery lasts.

- The events occur in disjoint intervals (non-overlapping)
- Two or more events cannot occur simultaneously
- Each event occurs at a constant rate

A continuous random variable X is said to have an exponential distribution with parameter $\lambda > 0$, if its PDF is given by

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & x > 0 \\ 0 & x \leq 0 \end{cases}$$

Where, e = the natural number e , λ = mean time between events, x = a random variable. The formula for the cumulative distribution function of the exponential distribution is: $F(x) = 1 - e^{-\lambda x}$ where $x \geq 0; \lambda > 0$

- Mean = $\frac{1}{\lambda}$
- Variance = $\frac{1}{\lambda^2}$

Suppose you are testing new software, and a bug causes errors randomly at a constant rate of three times per hour. The probability that the first bug will occur within the first ten minutes is?

Let constant rate or intensity be $\lambda = 3 / \text{hour}$ and $t = 1/6 \text{ hours}$ (10 minutes)

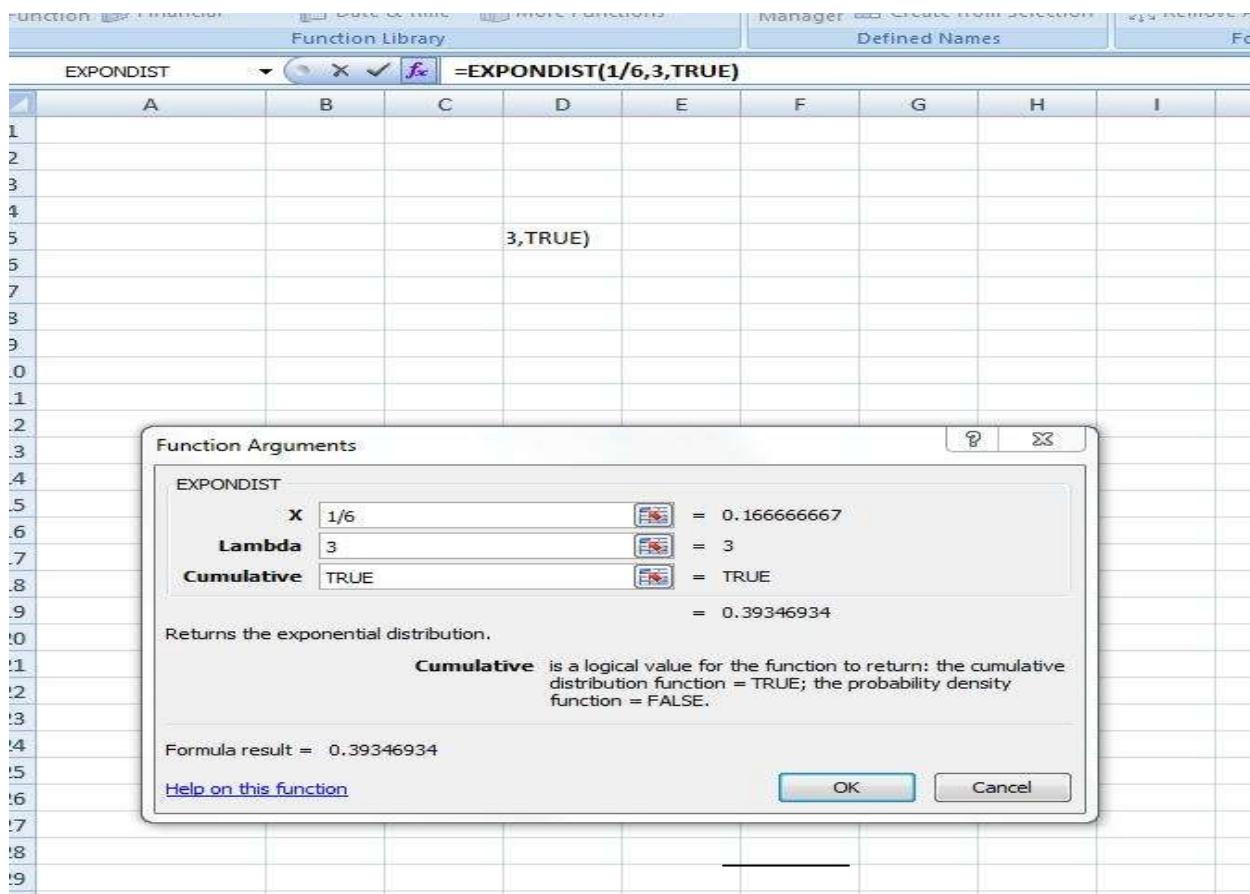
$$P(X < 1/6) = \int_0^{1/6} \lambda e^{-\lambda t} dt = 0.393$$

The probability that the first bug will occur in the next 10 minutes is 0.393.

Let us see this example in Excel using the function, EXPONDIST(x,lambda,cumulative)

X - Value of the function in question, Lambda - constant value,

Cumulative = $\begin{cases} \text{TRUE for cumulative distribution function} \\ \text{FALSE for probability density function} \end{cases}$



Function Arguments

EXPONDIST

X	1/6	= 0.166666667
Lambda	3	= 3
Cumulative	TRUE	= TRUE

= 0.39346934

Returns the exponential distribution.

Cumulative is a logical value for the function to return: the cumulative distribution function = TRUE; the probability density function = FALSE.

Formula result = 0.39346934

Help on this function

SAMPLING

Data sampling is a statistical technique used to select, manipulate and analyze a subset of data points to identify patterns and trends in the larger data set being examined. Data scientists, predictive modelers and data analysts often work with a small, manageable amount of data about a statistical population to build and run analytical models more quickly and accurately.

Sampling is particularly useful with data sets that are too large to efficiently analyze. For example, in big data analytics applications or surveys, identifying and analyzing a representative sample is more efficient and cost-effective than surveying the entire population.

The size of the required data sample and the possibility of introducing a sampling error are important. A small sample may sometimes reveal the most important information

about a data set. Using a larger sample increases the likelihood of accurately representing the data as a whole, even though the increased size of the sample may impede ease of manipulation and interpretation.

Data sampling can be accomplished using probability or nonprobability methods.

- **Probability Sampling** uses randomization to select sample members in the data set to ensure that there is no correlation between points.
- **Non-probability sampling** uses non-random techniques (i.e. the judgment of the researcher). It can be difficult to calculate the odds of any particular item, person or thing being included in your sample.

Probability data sampling methods include:

- Simple random sampling is used to randomly select data from the whole population using a software.
- Stratified sampling: Subsets of the data sets or population are grouped based on a common factor, and samples are randomly collected from each subgroup.
- Cluster sampling: The larger data set is divided into subsets (clusters) based on a specific factor, then a random sampling of clusters is analyzed.
- Multistage sampling: A complicated form of cluster sampling, this method also involves dividing the larger population into a number of clusters. Second-stage clusters are further broken out based on a secondary factor, and those clusters are then sampled and analyzed and the process continues.
- Systematic sampling: A sample is created by setting an interval at which point to extract data from the larger population. For example, selecting every 10th row in a spreadsheet of 20 items to create a sample size of 2 rows to analyze.

Non probability data sampling methods include:

- Convenience sampling: Data is collected from a convenient group (easily accessible, available).
- Consecutive sampling: Until the predetermined sample size is met, data is collected from every subject that meets the criteria.

- Purposive or judgmental sampling: Selecting the data to sample based on predefined criteria.
- Quota sampling: A selection ensuring equal representation within the sample for all subgroups in the data set or population.

Errors happen when you take a sample from the population rather than using the *entire* population. Sample error is the difference between the statistic you measure and the parameter you would find for the entire population.

If you were to survey the entire population, there would be no error. Sample error can only be reduced, since it is considered to be an acceptable tradeoff to avoid measuring the entire population. When the sample size gets larger, the margin of error becomes smaller. But, there is a notable exception: if you use cluster sampling, this may increase the error because of the similarities between cluster members.

RANDOM SAMPLING

Random sampling is a technique where each item in the population has an even chance and likelihood of being selected in the sample. Here the selection of items completely depends on chance or by probability and therefore the name 'method of chances'. The main attribute being every sample has the same probability of being chosen.

The steps involved in simple random sampling:

1. A list of all the members of the population is prepared initially and then each member is numbered starting from 1.
2. From this population, random samples are chosen in two ways:
 - Random number tables
 - Random number generator software (preferred more as the sample numbers can be generated randomly without human interference)

To minimize any biases in the process of simple random sampling, there are two approaches:

- Method of lottery

One of the oldest methods which is a mechanical example of random sampling. Here, each member of the population is numbered systematically. By writing each number on a separate piece of paper, they are mixed in a box and then numbers are drawn out of the box in a random manner.

- Use of random numbers

The use of random numbers (table) is an alternative method that also involves numbering the population.

Advantages of random sampling

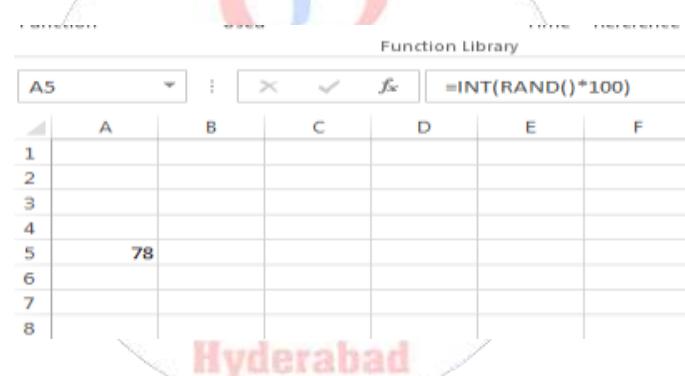
- No restriction on the sample size even when the population size is large.
- The quality of the data collected through this sampling method depends on the size of sample i.e., more the samples better the quality of the data.

Disadvantages of random sampling

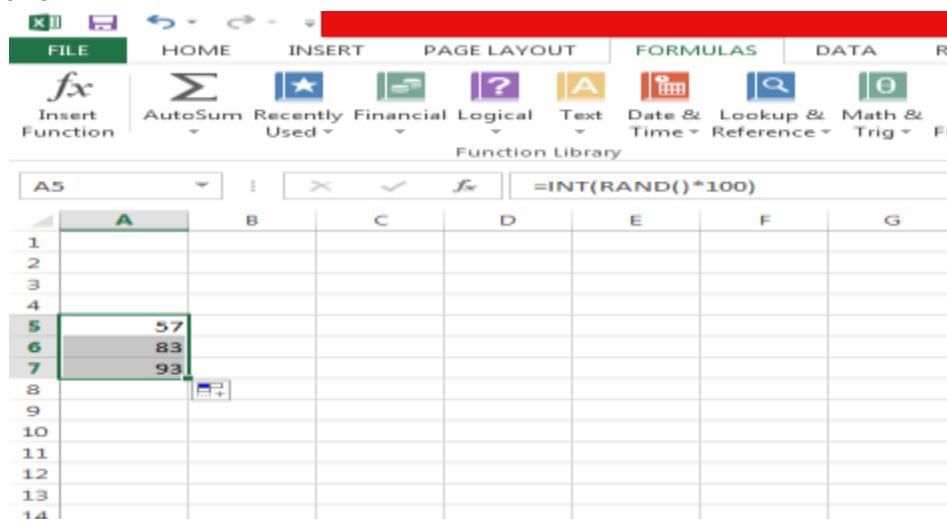
- Costly method of sampling as it needs the complete list of all potential data to be available beforehand.
- Not suitable for face-to-face interviews in larger geographical areas due to cost and time constraints.
- The larger population means a larger sample frame which is difficult to manage.

In Excel we have the function

- `RAND()` which generates a random number X where $0 \leq X < 1$
- `RAND()*100` which generates a random number X where $0 \leq X < 100$
- `INT(RAND()*100)` which generates a integer random number X where $0 \leq X < 100$



When we click on drag, we can have the function for the selected cells as shown



SYSTEMATIC SAMPLING

Systematic sampling is a probability sampling method in which the sample is chosen from a target population by selecting a random starting point and selecting other members after a fixed 'sampling interval'. This sampling interval is calculated by dividing the population size by the desired sample size.

For example, a local NGO is seeking to form a systematic sample of 500 volunteers from a population of 5000, they can select every 10th person ($5000/500 = 10$) in the population to systematically form a sampling interval.

Systematic sampling consumes the least time as it requires selection of sample size and identification of starting point for this sample that needs to be continued at regular intervals to form a sample.

Steps to form a systematic sample:

- A defined structural audience (population) to start working on the sampling aspect.
- Figuring out the ideal size of the sample (prefer bigger size to achieve accurate results).
- To each and every member of the sample, a number must be assigned.
- Calculating the sampling interval (population size/sample size).
- A number will be randomly chosen as the starting member (r) of the sample between 1 and (N/n) and this interval will be added to the random number to keep adding members in the sample. $r, r+i, r+2i$ etc. will be the elements of the sample.

Ameerpet / Kondapur

Types of Systematic Sampling:

- Linear systematic sampling:

A systematic sampling method where samples are not repeated at the end and ' n ' units are selected to be a part of a sample having ' N ' population units. It stops at the end of a population.

- Circular systematic sampling:

In circular systematic sampling, a sample starts again from the same point once again after ending, which is similar to linear systematic sampling but with a repetition. For example, if $N = 7$ and $n = 2$, $k=3.5$ named as a, b, c, d, e. There are two probable ways to form sample:

1. If we consider $k=3$, the samples will be – ad, be, ca, db and ec.
2. If we consider $k=4$, the samples will be – ae, ba, cb, dc and ed.

Linear Systematic Sampling	Circular Systematic Sampling
samples = k (sampling interval)	samples = N (total population)
The starting and ending points of this sample are distinct.	Repeats from the start point once the entire population is considered.
Arranged in a linear manner prior to selection.	Arranged in a circular manner.

Advantages:

- Simple and convenient to create, conduct, and analyze samples by the researchers.
- Beneficial in case of diverse population because of the even distribution of sample.

Disadvantages:

- Difficult when the population size cannot be estimated.
- Data becomes skewed if sample is taken from a group which already has a pattern.
- Sometimes even a standard arrangement (order/pattern) may not be obvious or visible, resulting in sampling bias.

STRATIFIED SAMPLING

A probability sampling technique in which the researcher divides the entire population into different subgroups (strata), then randomly selects the final items proportionally from the different strata (must be non-overlapping). Stratified sampling is also known as proportional sampling or quota sampling. We use strata like age, gender, socioeconomic status, religion, nationality and educational attainment.

Steps for Stratified sampling:

- Divide the population into smaller strata, based on shared attributes and characteristics of the members.
- Select a random sample from each stratum in a number that is proportional to the size of the stratum.
- To form a random sample, pool the subsets of the strata together.

Types of Stratified Sampling:

- Proportionate Stratified Random Sampling

The sample size of each stratum is proportionate to the respective population size when viewed against the entire population i.e., each stratum has the same sampling fraction.

For example, we have 3 strata with 100, 200 and 300 population sizes respectively. Let sampling fraction be $\frac{1}{2}$. Then, we should randomly sample 50, 100 and 150 subjects from each stratum respectively.

Stratum	A	B	C
Population Size	100	200	300
Sampling Fraction	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$
Final Sample Size	50	100	150

- Disproportionate Stratified Random Sampling

With disproportionate sampling, the different strata have different sampling fractions.

Advantages:

- Highlighting a specific subgroup within the population.
- Observing existing relationships between two or more strata.
- This allows the researcher to sample the rare extremes (even the smallest and most inaccessible) subgroups of the given population.
- When there is homogeneity within strata and heterogeneity between strata, the estimates can be of higher statistical precision.
- Due to precision, it requires a small sample size which saves a lot of time, money and effort of the researchers.

Disadvantages:

- This sampling requires the knowledge of distinguishing between strata in the sample frame
- Research process may take longer and more expensive due to the extra stage in the sampling procedure.
- If mistakes happen in allotting sampling fractions, a stratum may either be overrepresented or underrepresented which will result in skewed results adding further complexity.

Suppose for a survey, we need 50 students who are either juniors or seniors in a high school.

Based on gender,

year	boys	girls
junior	126	94
senior	77	85
total	203	179

To calculate the number of senior girls to be included in the 50 person sample issimply

$$(85/382) * 50 = 11.2 = 11$$

CLUSTERED SAMPLING

A sampling technique which divides the main population into various clusters which consist of multiple sample parameters like demographics, habits, background or any other attribute. Cluster sampling allows the researchers to collect data by bifurcating the data into small, more effective groups instead of selecting the entire population of data.

There are two ways to classify cluster sampling. The first way is based on the number of stages followed to obtain the cluster sample and the second way is the representation of the groups in the entire cluster.

Cluster sampling can be classified as single-stage, two-stage, and multiple stages (in most of the cases).

- Single Stage Cluster Sampling: Here, sampling will be done just once. For example, An NGO wants to create a sample of girls across 5 neighboring towns to provide education. To form a sample, The NGO can randomly select towns (clusters). Then they can extend help to the girls deprived of education in those towns directly.
- Two-Stage Cluster Sampling: A sample created using two-stages is always better than using a single stage because more filtered elements can be selected which can lead to improved results from the sample. In two-stage cluster sampling, by implementing systematic or simple random sampling only a handful of members are selected from each cluster instead of selecting all the elements of a cluster. For example, a business owner wants to explore the statistical performance of her plants which are spread across various parts of the state. Based on the number of plants, number of employees per plant and work done from each plant, single-stage sampling would be time and cost consuming. The owner thus creates samples of employees belonging to different plants to form clusters and then divides it into the size or operation status of the plant.

- **Multiple Stage Cluster Sampling:** For A research to be conducted across large multiple geographies, one needs to form complicated clusters that can be achieved only using multiple-stage cluster sampling technique. For example, if an organization intends to conduct a survey to analyze the performance of smart phones across India. They can divide the entire country's population into states (clusters) and further select cities with the highest population and also filter those using mobile devices.

Steps to form a Clustered sample:

- Form a target audience and required size of the sample.
- Create a sampling frame by using either an existing frame or by creating a new one for the target audience.
- Evaluate frames on the basis of coverage and clustering considering the population which can be exclusive and comprehensive.
- Determine the number of distinct groups by including the same average members in each group.
- Randomly choose clusters for sampling (mostly clusters are created by geography).

Advantages:

- Sampling of groups divided geographically require less work, time and cost.
- Ability to choose larger samples which will increase accessibility to various clusters.
- Due to large samples in each cluster, loss of accuracy in information per individual can be compensated.
- Since cluster sampling facilitates information from various areas and groups, it can be easily implemented in practical situations in comparison to other sampling methods.

Disadvantages:

- Requires group-level information to be known prior.
- Commonly has higher sampling error than othersampling techniques.

Even though both strata and clusters are non-overlapping subsets of the population, they do differ in several ways.

- All strata are represented in the sample; but only a subset of clusters are in the sample.
- With stratified sampling, the best survey results occur when elements within strata are internally homogeneous. Whereas, with cluster sampling, the best survey results occur when elements within clusters are internally heterogeneous.

QUOTA SAMPLING

A non-probability sampling technique where the assembled sample has the same proportions of individuals as the entire population with respect to known characteristics. The main reason in choosing quota samples is that it allows the researchers to sample a subgroup that is of great interest to the study. In a study that considers gender, socioeconomic status and religion as the basis of the subgroups, the final sample may have a skewed representation of age, race, educational attainment, marital status and a lot more.

Quota sampling can be classified as: controlled and uncontrolled.

Controlled quota sampling involves certain restrictions in order to limit sample choice of researcher.

Uncontrolled quota sampling, on the other hand, does not have any restrictions.

Step-by-step Quota Sampling

- The first most step is to bifurcate the entire population into mutually exhaustive subgroups, i.e., the elements of each of the subgroups should be a part of only one of those subgroups. For example, if a researcher wishes to understand the target market for an upcoming Bluetooth headphones variant, the most precise quota characteristic will be according to age groups.
- The researcher then evaluates the proportion in which the subgroups exist. This proportion has to be maintained within the sample selected using this sampling method. If 58% of the people who are interested in purchasing Bluetooth headphones are between the age group of 25-35 years, your subgroups also should have the same percentages of people.
- Selecting the sample size while maintaining the proportion evaluated in the previous step. If the population size is 500, the researcher can select a sample size of 50 elements.

Advantages of Quota Sampling

- Time effective since the primary data collection can be done in shorter time.
- Cost-effective.
- Independent on the presence of the sampling frames.

Disadvantages of Quota Sampling

- Calculating the sampling error is not possible in non-probability sampling methods.
- The risky projection of the research findings to the total population.

- Other characteristics may be disproportionately present in the final sample group.
- Great scope for researcher bias and the quality of work depends on researcher's experience.

CONVENIENCE SAMPLING

A non-probability sampling technique in which subjects are selected because of their convenience. The convenience samples are also called as the Accidental Samples because the subjects happen to be accidentally selected under study. This method is also referred as the chunk (fraction) taken from the population on the basis of its convenient availability and accessibility to the investigator. Face book polls is a popular example for convenience sampling. Some of the convenience samples which are available readily are the telephone directory, census survey report, automobile registration list and so on.

In pilot studies, the researcher prefers convenience sample because it allows to obtain basic data and trends regarding his study and also to avoid the complications of a randomized sample. This sampling technique is useful in documentation that a particular quality or phenomenon occurs within a given sample.

When using convenience sampling, it is necessary to describe how the sample differs from an ideal random sample. Description of the individuals who might be left out during the selection process or the individuals who are overrepresented in the sample might also be necessary.

In business studies, this method is used to collect initial primary data regarding specific issues such as perception of image of a particular brand or collecting opinions of perspective customers to a new design of a product.

Advantages:

- Ease of research and simplicity of sampling
- Helps in pilot studies and for hypothesis generation
- Data collection can be facilitated within short duration of time due to its simplicity.

Disadvantages:

- Risky due to selection bias and influences beyond the control of the researcher. The best way of reducing bias is using it along with probability sampling. Since probability sampling gets the measurement parameter with it to keep the bias under check.

- High level of sampling error since the samples are selected conveniently, it is not necessary that these reflect the true attributes or characteristics of the target population. Also, there are several choices of bias of the investigator's selection which can tamper the results that leads to higher level of sampling error.

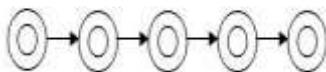
SNOWBALL SAMPLING

A non-probability sampling technique in which researcher begins with a small population of known individuals and expands the sample by asking those initial participants to identify others that should participate in the study. In other words, the sample starts small (like a snowball running down the hill) but into a larger sample through the course of the research. Snowball sampling is also known as *chain sampling*, *cold-calling*, *chain-referral sampling* or *referral sampling*.

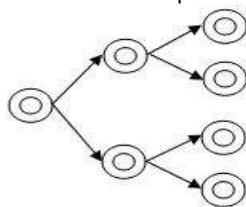
This often occurs when the population is somehow marginalized, like homeless or formerly incarcerated individuals or those who are involved in illegal activities. It is also common to use this sampling technique with people whose membership in a particular group is difficult to find not widely known, such as closeted gay people or bisexual or transgender individuals, homeless people, drug addicts, members of an elite golf club etc. Some people may not want to be found. For example, if a study requires investigating cheating on exams, shoplifting, drug use, or any other "unacceptable" social behavior, the participants would worry to come forward due to possible ramifications. However, other study participants would likely know other people in the same situation as themselves and could inform others about the benefits of the study and reassure them of confidentiality.

Types of Snowball Sampling [9963799240](tel:9963799240) / [7730997544](tel:7730997544)

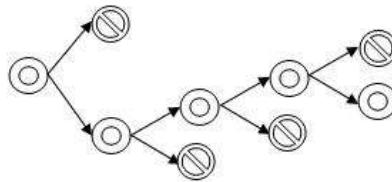
1. *Linear snowball sampling*: Forming a sample group which starts with only one subject and the subject refers only one referral and further continues until the fully sampled group is formed.



2. *Exponential non-discriminative snowball sampling*: The first subject recruited to the sample group refers one or more referrals. Each new referral is explored until primary data forms sufficient amount of samples.



3. *Exponential discriminative snowball sampling*: Subjects give multiple referrals but only one subject is recruited among them to form effective samples.



Snowball sampling consists of two steps:

1. Identify potential subjects from sampling frame in the population. Mostly one or two subjects can be found at this stage.
 2. Ask those subjects to recruit other people and then ask those people to recruit till there are no more subjects left/sample size becomes unmanageable.

Advantages:

- This process allows the researcher to reach populations that are difficult to sample compared to other sampling methods.
 - The process is cheap, simple and cost-efficient.
 - This technique needs lesser workforce as the subjects were involved directly compared to other sampling techniques.

Disadvantages:

- Oversampling a particular network of peers may lead to biasing.
 - There is no guarantee about the representation of samples. It is not possible to determine the actual pattern of distribution of population.
 - Determination of the sampling error and make statistical inferences from the sample to the population is not possible due to the absence of random selection of samples.

BIAS IN SAMPLING

A sampling method is called biased if the survey sample does not accurately represent the population. Sampling bias is sometimes called ascertainment bias or systematic bias. Sampling bias refers to sample and also the method of sampling. Bias can be either intentional or not. Some time seven poor measurement process can lead to bias. When measuring a nonlinear functional of the probabilities from a limited number of experimental samples a bias may occur, even when these samples are picked randomly from the underlying population and there is thus no sampling bias. This bias is called “limited sampling bias”.

Example:

Telephone sampling is common in marketing surveys. In a survey, a random sample is chosen from the sampling frame having a list of telephone numbers of people in the particular area. This method does involve taking a simple random sample, but it will miss

1. People who do not have a phone or
2. People who only have a cell phone that has an area code not in the region being surveyed
3. People who do not wish to be surveyed, including those who monitor calls on an answering machine
4. People who don't answer those from telephone surveyors. Thus systematically excluding certain types of consumers in the area.

Here are few sources of sampling bias:

Convenience samples:

David A. Freedman, statistics professor stated "Statistical inference with convenience samples is a risky business." In cases where it may not be possible or not be practical to choose a random sample, a convenience sample might be used. Sometimes convenience sample is considered as a random sample, but often it gets biased. Under coverage problem arises with convenience samples. Under coverage occurs when some members of the population are not adequately represented in the sample. In the above example people who do not have a phone under covered.

Voluntary response samples:

If the researcher appeals to people to voluntarily participate in a survey, then the resulting sample is called a voluntary response sample. These samples are always biased because they only include people who choose volunteer, whereas a random sample would need to include people whether or not they choose to volunteer. Often, in a survey a voluntary response samples oversample (people having strong opinions) or under sample (people who don't care). In the above example people who do not want to be surveyed come under these samples.

Extrapolation:

Drawing of a conclusion about something beyond the data range is called extrapolation. Extrapolation of a biased sample systematically excludes certain parts of the population under consideration, the inferences only apply to the subpopulation which has actually been sampled. Extrapolation also occurs if, for example, an inference based on a sample of senior citizens is applied to older adults or to adults without citizenship.

Self-Selection Bias

A self-selection bias results when the non-random component occurs after the potential subject has enlisted in the experiment. Considering the hypothetical experiment in which subjects were asked about the details of their sex lives, assume that the subjects did not know what the experiment was about until they showed up. Many of the subjects would definitely leave the experiment resulting in a biased sample. Many of television or web site polls taken are prone to self-selection bias.

Correction and reduction of sampling bias

If the statistic is unbiased, the average of all the statistics from all possible samples will equal the true population parameter; even though any individual statistic may differ from the population parameter. The variability among statistics from different samples is sampling error. Increasing the sample size tends to reduce the sampling error but does not affect survey bias (under coverage, non response bias, etc.).

To reduce sampling bias, the two steps when designing an experiment are (i) to avoid convenience sampling (ii) to ensure that the target population is properly defined and the sample frame matches it as much as possible. Random sampling generates representative samples by eliminating voluntary response bias and guarding against under coverage bias. All probability sampling methods rely on random sampling.

Solutions to the bias due to non-response samples can be divided into ex-ante and ex-post solutions. Ex-ante solutions help to prevent and minimize non-response in various ways (for instance specific training of enumerators, several attempts to interview the respondent, etc.) whereas ex-post solutions try to gather auxiliary information about non-respondents which is then used to calculate a probability of response for different population sub-groups and re-weight response data for the inverse of such probability.

SAMPLING DISTRIBUTION

Sampling distribution is the probability distribution of a sample of a population instead of the entire population using various statistics (mean, mode, median, standard deviation and range) based on randomly selected samples. This distribution helps in hypothesis testing (likeness of an outcome).

The properties of sampling distribution vary depending on the sample size as compared to the population. Properties of Sampling Distributions:

1. The shape of the distribution is symmetric and approximately normal.
2. There are no outliers or other deviations from the overall pattern.
3. The center of the distribution must be very close to the true population mean.

The population is assumed to be normally distributed. If the sample size is large enough, then sampling distribution will also be normal which is determined by the mean and the standard deviation values.

For Example, a random sample of 20 people from the population of women in Hyderabad between the ages of 22 and 35 years is selected and computed the mean height of sample. It might be lesser or greater, but not equal the population mean exactly. The most common measure of how much sample means differ from each

other is the standard deviation (standard error of the mean) of the sampling distribution of the mean. The standard error of the mean would be small, if all the sample means were very close to the population mean. The standard error of the mean would be large, if the sample means varied considerably.

CENTRAL LIMIT THEOREM

The central limit theorem states that:

In a population with a finite mean, μ and a finite non-zero variance, σ^2 the sampling distribution of the mean approaches a normal distribution with a mean of μ and a variance of σ^2/N as the sample size, N increases. As the sample size increases, the closer the sampling distribution of the mean to become a normal distribution.

The sampling distribution of the difference between means can be computed by following these steps repeatedly:

1. sample n_1 scores from Population 1 and n_2 scores from Population 2,
2. compute the means of the two samples (μ_1 and μ_2), and
3. Compute the difference between means, $\mu_1 - \mu_2$.

The distribution of the differences between means is the sampling distribution of the difference between means, the mean of the sampling distribution is:

$$\mu_{M_1 - M_2} = \mu_1 - \mu_2$$

From the variance sum law, we know that:

$$\sigma_{M_1 - M_2}^2 = \sigma_{M_1}^2 + \sigma_{M_2}^2$$

$$\sigma_{M_1 - M_2}^2 = \frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}$$

The standard error of the difference between means,

$$\sigma_{M_1 - M_2} = \sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}$$

The sampling distributions were often derived from the normal distribution implied by the central limit theorem. This holds for

- normal distribution for sample means and proportions

- t distribution for sample means in a t-test
- beta coefficients in regression analysis;
- chi-square distribution for variances;
- F distribution for variance ratios in ANOVA.

Sampling distribution of the Mean

The mean of the sampling distribution is in fact the mean of the population after computing sample means and population means. However, the standard deviation differs for the sampling distribution as compared to the population.

If the population is large enough, this is given by $\sigma_x = \frac{\sigma}{\sqrt{n}}$

Where σ - the mean of the population,

σ_x - the population mean.

Sampling distribution of the Mean

The mean of the sampling distribution, $\mu_x = \mu$ (mean of the population)

The standard error of the sampling distribution, $\sigma_x = \frac{\sigma}{\sqrt{n}} * \sqrt{\frac{N-n}{N-1}}$

Where σ - standard deviation of the population,

N - the population size, and

n - the sample size.

In the standard error formula, the factor $\sqrt{\frac{N-n}{N-1}}$ is called the finite population correction or fpc. When the population size is very large relative to the sample size, the $fpc \approx 1$; and the standard error formula can be approximated to:

$$\sigma_x = \frac{\sigma}{\sqrt{n}}$$

Safer to use this formula when the sample size is no bigger than 1/20 of the population size.

Sampling distribution of the Proportion

In a population of size N , suppose that the probability of the occurrence of an event is P for success; and the probability of the event's non-occurrence is Q for failure. From this population, we then draw all possible samples of size n . And within each sample, we determine the proportion of successes p and failures q , thus creating a sampling distribution of the proportion.

We find that p is equal to the probability of success in the population (P). And p is determined by the standard deviation of the population (σ), the population size, and the sample size.

The mean of the sampling distribution of the proportion, $\mu_p = P$

The standard error of the sampling distribution, $\sigma_p = \frac{\sigma}{\sqrt{n}} * \sqrt{\frac{N-n}{N-1}} = \sqrt{\frac{PQ}{n}} * \sqrt{\frac{N-n}{N-1}}$

When the population size is very large relative to the sample size, the $fpc \approx 1$;

So, the standard error formula can be approximated to $\sigma_p = \sqrt{\frac{PQ}{n}}$

Safer to use this formula when the sample size is no bigger than 1/20 of the population size.

To make it easier:

- Use the normal distribution, if the population standard deviation is known/ if the sample size is large.
- Use the t-distribution, if the population standard deviation is unknown/ if the sample size is small.

CENTRAL LIMIT THEOREM

Statement: Given a sufficiently large sample size selected from a population with a finite variance, the mean of all samples from the same population will be approximately equal to the mean of the population thereby forming an approximate normal distribution pattern. When we draw repeated samples from a given population, the shape of the distribution of means will be converging to the normal distribution irrespective of the shape of the population distribution. As the sample size increases, the sampling distribution of the mean, can be approximated by a normal distribution with mean μ and standard deviation σ/\sqrt{n} .

A certain random variable of interest is a sum of a large number of independent random variables where we use the CLT to justify using the normal distribution. Examples of such random variables are found in almost every discipline like:

- Usually errors in laboratory measurement are modeled by normal random variables.
- In communication and signal processing, the most frequently used model for noise is Gaussian noise.

When we select random samples from a population to obtain statistics (mean, variance...) about the population, we often assume the resultant as a normal random variable. If you have a problem in which you are interested in a sum of one thousand random variables, it might be extremely difficult to find the distribution of the sum by direct calculation. Using the CLT we can immediately write the distribution, if we know the mean and variance of the random variables.

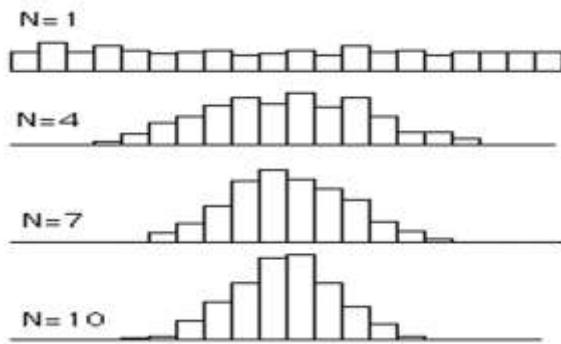
We can use normal approximation, if $n \geq 30$.

Three different components of the central limit theorem

- (1) Successive sampling from a population
- (2) Increasing sample size
- (3) Population distribution.

Example

Here the resulting frequency distributions each based on 500 means is shown. For $n = 4$, 4 scores were sampled from a uniform distribution 500 times and then computed the mean each time. Similarly, with means of 7 scores for $n = 7$ and 10 scores for $n = 10$. As n is increasing, the spread of the distributions is decreasing and the distributions are becoming limited to center (clustering around the mean)



Limitations of central limit theorem:

- The values must be drawn independently from the same distribution having finite mean and variance and should not be correlated.
- The rate of convergence depends on the skewness of the distribution.
- Sums from an exponential distribution converge for smaller sample sizes. Sums from a lognormal distribution require larger sizes.

STANDARD NORMAL DISTRIBUTION

A standard normal distribution is a normal distribution with mean 0, standard deviation 1, Area under curve 1 and of infinite extent. When the unit of measure is changed to measure standard deviations from the mean, then all normal distributions will become equivalent to the standard normal distribution. Transformation of Normal distributions into standard normal distribution is given by the formula:

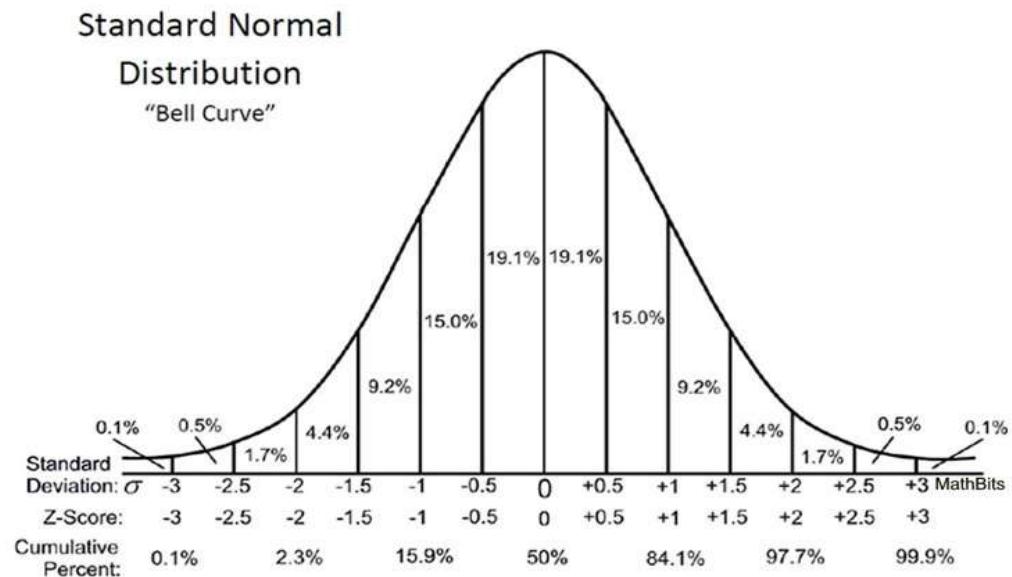
$$z = \frac{X - \mu}{\sigma}$$

where X - a score, μ - the mean, σ - the standard deviation of original normal distribution. The standard normal distribution is often known as the z distribution. A z score tells us how far the score is from the mean in terms of number of standard deviations. Percentile rank is the area (probability) to the left of the value i.e., by adding the percentages from the chart from left of the curve. The mean is the 50th percentile (50%).

From the 68-95-99.7 rule/Empirical rule:

For a variable with the standard normal distribution,

- 68% of the observations fall between -1σ and 1σ (within 1 standard deviation of the mean of 0),
- 95% fall between -2σ and 2σ (within 2 standard deviations of the mean) and
- 99.7% fall between -3σ and 3σ (within 3 standard deviations of the mean).



Z-SCORES



A **z-score/ standard score** indicates how many standard deviations an element is far from the mean. A z-score can be calculated using the formula:

$$z = (X - \mu) / \sigma$$

Where z - z-score, X - value of the element, μ - population mean, and σ - standard deviation.

Interpretation of z-scores:



- If z -score $< 0 \Rightarrow$ an element $<$ mean.
- If z -score $> 0 \Rightarrow$ an element $>$ mean.
- If z -score $= 0 \Rightarrow$ an element $=$ mean.
- If z -score $= 1 \Rightarrow$ an element which is $1\sigma >$ mean.
- If z -score $= 2 \Rightarrow$ an element which is $2\sigma >$ mean.
- If z -score $= -1 \Rightarrow$ an element which is $1\sigma <$ mean.
- If z -score $= -2 \Rightarrow$ an element which is $2\sigma <$ mean and so on.
- If the number of elements in the set is larger, about
 - 68% have a z-score between -1 and 1;
 - 95% have a z-score between -2 and 2;
 - 99% have a z-score between -3 and 3.

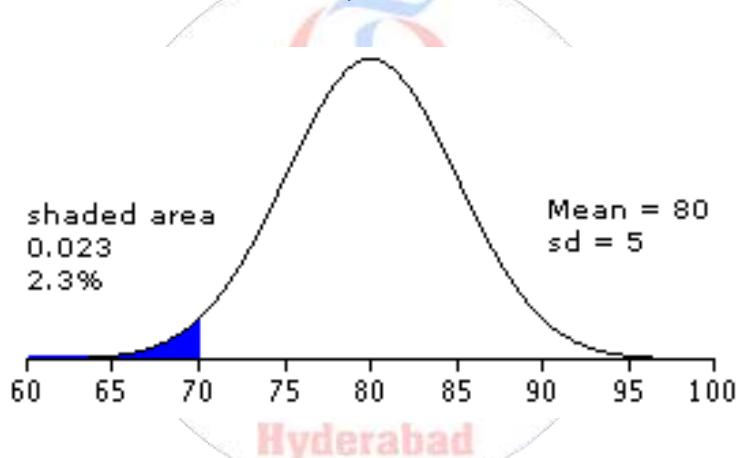
We can find the z score table online and save it for later calculations.

Example1: Find the probability for IQ values between 75 and 130, assuming a normal distribution, mean = 100 and standard deviation = 15. An IQ of 75 corresponds with a z score of $-1.67(75-100/15)$ and an IQ of 130 corresponds with a z score of 2.00. The value for -1.67 is .4525 from the table. For 2.00 we find .4772. The probability of an IQ between 75 and 130 is the same as $.4525+.4772=.9297$.

If the mean and standard deviation of a normal distribution are known, the percentile rank of a person obtaining a specific score can be calculated.

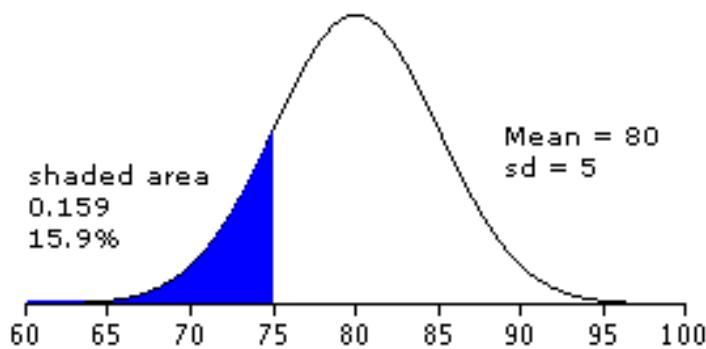
Example2: Assume a test in Introductory Psychology is normally distributed with a mean of 80 and a standard deviation of 5. What is the percentile rank of a person who receives a score of 70?

Using the formula, we know that $z = 70-80/5=-2$. Using the table only 2.3% of the population will be less than or equal to a score 2σ below the mean.



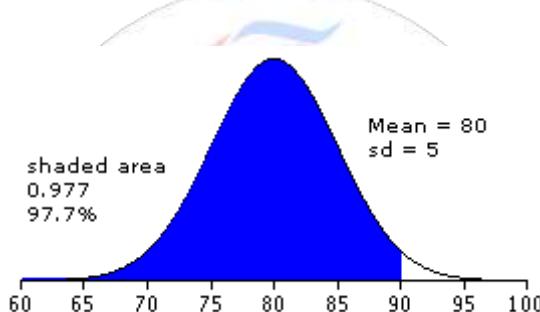
The shaded area occupies 2.3% of the total area. The proportion of the area below 70 = the proportion of the scores below 70.

Similarly, a person scoring 75 on the test is 15.9% of the population.



Area from -∞ to z	
z	Area from -∞ to z
-3.0	.0013
-2.5	.0062
-2.0	.0227
-1.5	.0668
-1.0	.1587
-0.5	.3085
0.0	.5000
0.5	.6915
1.0	.8413
1.5	.9332
2.0	.9772
2.5	.9938
3.0	.9987

Same way, the percentile rank of a person receiving a score of 90 on the test is 97.7%



Since $z = (90 - 80)/5 = 2$ it can be determined from the table that a z score of 2 is equivalent to the 97.7th percentile: The proportion of people scoring below 90 is thus 97.7%.

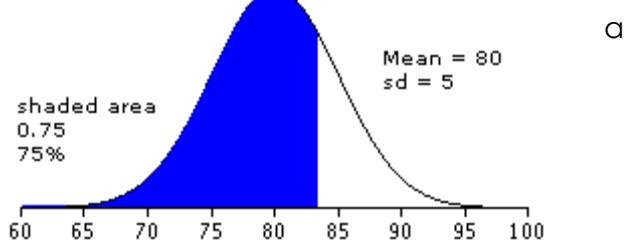
Example 3: What score on the Introductory Psychology test would it have taken to be in the 75th percentile? (Mean is 80 and a standard deviation is 5)

We now have to find out z score which can be done by reversing the steps followed in example 2

First, determine z score from table with value associating to 0.75 by using a z table. The value of z is 0.674.

Second, the standard deviation is 5, since little algebra demonstrates that $X = \mu + z\sigma$

$$X = 80 + (.674)(5) = 83.37.$$



What is a Confidence Interval?

Statisticians use a confidence interval to describe the amount of uncertainty associated with a sample estimate of a population parameter.

How to Interpret Confidence Intervals

Suppose that a 90% confidence interval states that the population mean is greater than 100 and less than 200. How would you interpret this statement?

Some people think this means there is a 90% chance that the population mean falls between 100 and 200. This is incorrect. Like any population parameter, the population mean is a constant, not a random variable. It does not change. The probability that a constant falls within any given range is always 0.00 or 1.00.

The confidence level describes the uncertainty associated with a *sampling method*. Suppose we used the same sampling method to select different samples and to compute a different interval estimate for each sample. Some interval estimates would include the true population parameter and some would not. A 90% confidence level means that we would expect 90% of the interval estimates to include the population parameter; a 95% confidence level means that 95% of the intervals would include the parameter; and so on.

Confidence Interval Data Requirements

To express a confidence interval, you need three pieces of information.

- Confidence level
- Statistic
- Margin of error

Given these inputs, the range of the confidence interval is defined by the *sample statistic + margin of error*. And the uncertainty associated with the confidence interval is specified by the confidence level.

Often, the margin of error is not given; you must calculate it. Previously, we described how to compute the margin of error.

How to Construct a Confidence Interval

There are four steps to constructing a confidence interval.

- Identify a sample statistic. Choose the statistic (e.g, sample mean, sample proportion) that you will use to estimate a population parameter.
- Select a confidence level. As we noted in the previous section, the confidence level describes the uncertainty of a sampling method. Often, researchers choose 90%, 95%, or 99% confidence levels; but any percentage can be used.
- Find the margin of error. If you are working on a homework problem or a test question, the margin of error may be given. Often, however, you will need to compute the margin of error, based on one of the following equations.

Margin of error = Critical value * Standard deviation of statistic

Margin of error = Critical value * Standard error of statistic

For guidance, see how to compute the margin of error.

- Specify the confidence interval. The uncertainty is denoted by the confidence level. And the range of the confidence interval is defined by the following equation.

Confidence interval = sample statistic + Margin of error

The sample problem in the next section applies the above four steps to construct a 95% confidence interval for a mean score. The next few lessons discuss this topic in greater detail.

Test Your Understanding

Problem 1

Suppose we want to estimate the average weight of an adult male in Dekalb County, Georgia. We draw a random sample of 1,000 men from a population of 1,000,000 men and weigh them. We find that the average man in our sample weighs 180 pounds, and the standard deviation of the sample is 30 pounds. What is the 95% confidence interval.

- (A) $180 + 1.86$
- (B) $180 + 3.0$
- (C) $180 + 5.88$
- (D) $180 + 30$
- (E) None of the above

Solution

The correct answer is (A). To specify the confidence interval, we work through the four steps below.

- Identify a sample statistic. Since we are trying to estimate the mean weight in the population, we choose the mean weight in our sample (180) as the sample statistic.
- Select a confidence level. In this case, the confidence level is defined for us in the problem. We are working with a 95% confidence level.
- Find the margin of error. Previously, we described how to compute the margin of error. The key steps are shown below.
 - Find standard error. The standard error (SE) of the mean is:

$$SE = s / \sqrt{n}$$

$$SE = 30 / \sqrt{1000} = 30/31.62 = 0.95$$

- Find critical value. The critical value is a factor used to compute the margin of error. To express the critical value as a t score (t^*), follow these steps.

- Compute alpha (α):

$$\alpha = 1 - (\text{confidence level} / 100) = 0.05$$

- Find the critical probability (p^*):

$$p^* = 1 - \alpha/2 = 1 - 0.05/2 = 0.975$$

- Find the degrees of freedom (df):

$$df = n - 1 = 1000 - 1 = 999$$

- The critical value is the t statistic having 999 degrees of freedom and a cumulative probability equal to 0.975. From the t Distribution Calculator, we find that the critical value is 1.96.

Note: We might also have expressed the critical value as a z-score. Because the sample size is large, a z-score analysis produces the same result - a critical value equal to 1.96.

- Compute margin of error (ME):

$$ME = \text{critical value} * \text{standard error}$$

$$ME = 1.96 * 0.95 = 1.86$$

- Specify the confidence interval. The range of the confidence interval is defined by the sample statistic + margin of error. And the uncertainty is denoted by the confidence level. Therefore, this 95% confidence interval is $180 + 1.86$.

Confidence Interval: Proportion (Large Sample)

This lesson describes how to construct a confidence interval for a sample proportion, p , when the sample size is large.

Estimation Requirements

The approach described in this lesson is valid whenever the following conditions are met:

- The sampling method is simple random sampling.
- The sample is sufficiently large. As a rule of thumb, a sample is considered "sufficiently large" if it includes at least 10 successes and 10 failures.

Note the implications of the second condition. If the population proportion were close to 0.5, the sample size required to produce at least 10 successes and at least 10 failures would probably be close to 20. But if the population proportion were extreme (i.e., close to 0 or 1), a much larger sample would probably be needed to produce at least 10 successes and 10 failures.

For example, imagine that the probability of success were 0.1, and the sample were selected using simple random sampling. In this situation, a sample size close to 100 might be needed to get 10 successes.

The Variability of the Sample Proportion

To construct a confidence interval for a sample proportion, we need to know the variability of the sample proportion. This means we need to know how to compute the standard deviation or the standard error of the sampling distribution.

- Suppose k possible samples of size n can be selected from the population. The standard deviation of the sampling distribution is the "average" deviation between the k sample proportions and the true population proportion, P . The standard deviation of the sample proportion σ_p is:

$$\sigma_p = \sqrt{P * (1 - P) / n} * \sqrt{(N - n) / (N - 1)}$$

where P is the population proportion, n is the sample size, and N is the population size. When the population size is much larger (at least 20 times larger) than the sample size, the standard deviation can be approximated by:

$$\sigma_p = \sqrt{P * (1 - P) / n}$$

- When the true population proportion P is not known, the standard deviation of the sampling distribution cannot be calculated. Under these circumstances, use the standard error. The standard error (SE) can be calculated from the equation below.

$$SE_p = \sqrt{p * (1 - p) / n} * \sqrt{(N - n) / (N - 1)}$$

where p is the sample proportion, n is the sample size, and N is the population size. When the population size at least 20 times larger than the sample size, the standard error can be approximated by:

$$SE_p = \sqrt{p * (1 - p) / n}$$

Alert

The Advanced Placement Statistics Examination only covers the "approximate" formulas for the standard deviation and standard error.

$$\sigma_p = \sqrt{P * (1 - P) / n}$$

$$SE_p = \sqrt{p * (1 - p) / n}$$

However, students are expected to be aware of the limitations of these formulas; namely, the approximate formulas should only be used when the population size is at least 20 times larger than the sample size.

How to Find the Confidence Interval for a Proportion

Previously, we described how to construct confidence intervals. For convenience, we repeat the key steps below.

- Identify a sample statistic. In this case, the sample statistic is the sample proportion. We use the sample proportion to estimate the population proportion.
- Select a confidence level. The confidence level describes the uncertainty of a sampling method. Often, researchers choose 90%, 95%, or 99% confidence levels; but any percentage can be used.
- Find the margin of error. Previously, we showed how to compute the margin of error.
- Specify the confidence interval. The range of the confidence interval is defined by the sample statistic + margin of error. And the uncertainty is denoted by the confidence level.

In the next section, we work through a problem that shows how to use this approach to construct a confidence interval for a proportion.

Test Your Understanding**Problem 1**

A major metropolitan newspaper selected a simple random sample of 1,600 readers from their list of 100,000 subscribers. They asked whether the paper should increase its

coverage of local news. Forty percent of the sample wanted more local news. What is the 99% confidence interval for the proportion of readers who would like more coverage of local news?

- (A) 0.30 to 0.50
- (B) 0.32 to 0.48
- (C) 0.35 to 0.45
- (D) 0.37 to 0.43
- (E) 0.39 to 0.41

Solution

The answer is (D). The approach that we used to solve this problem is valid when the following conditions are met.

- The sampling method must be simple random sampling. This condition is satisfied; the problem statement says that we used simple random sampling.
- The sample should include at least 10 successes and 10 failures. Suppose we classify a "more local news" response as a success, and any other response as a failure. Then, we have $0.40 * 1600 = 640$ successes, and $0.60 * 1600 = 960$ failures - plenty of successes and failures.
- If the population size is much larger than the sample size, we can use an "approximate" formula for the standard deviation or the standard error. This condition is satisfied, so we will use one of the simpler "approximate" formulas.

Since the above requirements are satisfied, we can use the following four-step approach to construct a confidence interval.

- Identify a sample statistic. Since we are trying to estimate a population proportion, we choose the sample proportion (0.40) as the sample statistic.
- Select a confidence level. In this analysis, the confidence level is defined for us in the problem. We are working with a 99% confidence level.
- Find the margin of error. Elsewhere on this site, we show how to compute the margin of error when the sampling distribution is approximately normal. The key steps are shown below.
 - Find standard deviation or standard error. Since we do not know the population proportion, we cannot compute the standard deviation;

instead, we compute the standard error. And since the population is more than 20 times larger than the sample, we can use the following formula to compute the standard error (SE) of the proportion:

$$SE = \sqrt{[p(1 - p) / n]}$$

$$SE = \sqrt{[(0.4)*(0.6) / 1600]} = 0.012$$

- Find critical value. The critical value is a factor used to compute the margin of error. Because the sampling distribution is approximately normal and the sample size is large, we can express the critical value as a z-score by following these steps.
 - Compute alpha (α):
$$\alpha = 1 - (\text{confidence level} / 100)$$
$$\alpha = 1 - (99/100) = 0.01$$
 - Find the critical probability (p^*):
$$p^* = 1 - \alpha/2 = 1 - 0.01/2 = 0.995$$
 - Find the degrees of freedom (df):
$$df = n - 1 = 1600 - 1 = 1599$$
 - Find the critical value. Since we don't know the population standard deviation, we'll express the critical value as a t statistic. For this problem, it will be the t statistic having 1599 degrees of freedom and a cumulative probability equal to 0.995. Using the t Distribution Calculator, we find that the critical value is 2.58.
- Compute margin of error (ME):

$$ME = \text{critical value} * \text{standard error}$$

$$ME = 2.58 * 0.012 = 0.03$$

- Specify the confidence interval. The range of the confidence interval is defined by the *sample statistic + margin of error*. And the uncertainty is denoted by the confidence level.

Therefore, the 99% confidence interval is 0.37 to 0.43. That is, the 99% confidence interval is the range defined by $0.4 + 0.03$.

Chi-Square Test for Independence

This lesson explains how to conduct a **chi-square test for independence**. The test is applied when you have two categorical variables from a single population. It is used to determine whether there is a significant association between the two variables.

For example, in an election survey, voters might be classified by gender (male or female) and voting preference (Democrat, Republican, or Independent). We could use a chi-square test for independence to determine whether gender is related to voting preference. The sample problem at the end of the lesson considers this example.

When to Use Chi-Square Test for Independence

The test procedure described in this lesson is appropriate when the following conditions are met:

- The sampling method is simple random sampling.
- The variables under study are each categorical.
- If sample data are displayed in a contingency table, the expected frequency count for each cell of the table is at least 5.

This approach consists of four steps: (1) state the hypotheses, (2) formulate an analysis plan, (3) analyze sample data, and (4) interpret results.

State the Hypotheses

Suppose that Variable A has r levels, and Variable B has c levels. The null hypothesis states that knowing the level of Variable A does not help you predict the level of Variable B. That is, the variables are independent.

H_0 : Variable A and Variable B are independent.

H_a : Variable A and Variable B are not independent.

The alternative hypothesis is that knowing the level of Variable A can help you predict the level of Variable B.

Note: Support for the alternative hypothesis suggests that the variables are related; but the relationship is not necessarily causal, in the sense that one variable "causes" the other.

Formulate an Analysis Plan

The analysis plan describes how to use sample data to accept or reject the null hypothesis. The plan should specify the following elements.

- Significance level. Often, researchers choose significance levels equal to 0.01, 0.05, or 0.10; but any value between 0 and 1 can be used.
- Test method. Use the chi-square test for independence to determine whether there is a significant relationship between two categorical variables.

Analyze Sample Data

Using sample data, find the degrees of freedom, expected frequencies, test statistic, and the P-value associated with the test statistic. The approach described in this section is illustrated in the sample problem at the end of this lesson.

- **Degrees of freedom.** The degrees of freedom (DF) is equal to:

$$DF = (r - 1) * (c - 1)$$

where r is the number of levels for one categorical variable, and c is the number of levels for the other categorical variable.

- **Expected frequencies.** The expected frequency counts are computed separately for each level of one categorical variable at each level of the other categorical variable. Compute $r * c$ expected frequencies, according to the following formula.

$$E_{r,c} = (n_r * n_c) / n$$

where $E_{r,c}$ is the expected frequency count for level r of Variable A and level c of Variable B, n_r is the total number of sample observations at level r of Variable A, n_c is the total number of sample observations at level c of Variable B, and n is the total sample size.

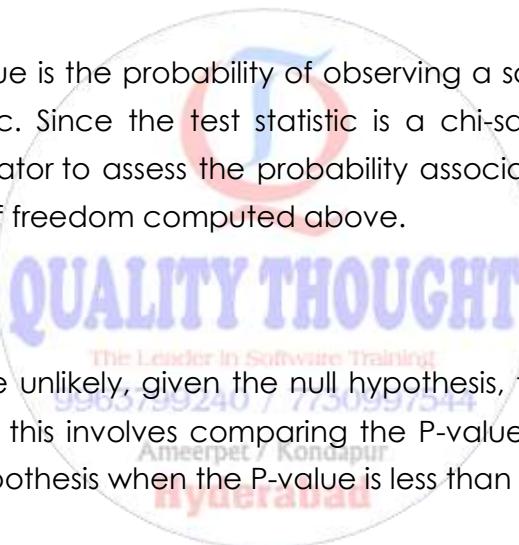
- **Test statistic.** The test statistic is a chi-square random variable (X^2) defined by the following equation.

$$X^2 = \sum [(O_{r,c} - E_{r,c})^2 / E_{r,c}]$$

where $O_{r,c}$ is the observed frequency count at level r of Variable A and level c of Variable B, and $E_{r,c}$ is the expected frequency count at level r of Variable A and level c of Variable B.

- **P-value.** The P-value is the probability of observing a sample statistic as extreme as the test statistic. Since the test statistic is a chi-square, use the Chi-Square Distribution Calculator to assess the probability associated with the test statistic. Use the degrees of freedom computed above.

Interpret Results



QUALITY THOUGHT

If the sample findings are unlikely, given the null hypothesis, the researcher rejects the null hypothesis. Typically, this involves comparing the P-value to the significance level, and rejecting the null hypothesis when the P-value is less than the significance level.

Test Your Understanding

Problem

A public opinion poll surveyed a simple random sample of 1000 voters. Respondents were classified by gender (male or female) and by voting preference (Republican, Democrat, or Independent). Results are shown in the contingency table below.

	Voting Preferences			Row total
	Rep	Dem	Ind	
Male	200	150	50	400
Female	250	300	50	600
Column total	450	450	100	1000

Is there a gender gap? Do the men's voting preferences differ significantly from the women's preferences? Use a 0.05 level of significance.

Solution

The solution to this problem takes four steps: (1) state the hypotheses, (2) formulate an analysis plan, (3) analyze sample data, and (4) interpret results. We work through those steps below:

- **State the hypotheses.** The first step is to state the null hypothesis and an alternative hypothesis.

H_0 : Gender and voting preferences are independent.

H_a : Gender and voting preferences are not independent.

- **Formulate an analysis plan.** For this analysis, the significance level is 0.05. Using sample data, we will conduct a chi-square test for independence.
- **Analyze sample data.** Applying the chi-square test for independence to sample data, we compute the degrees of freedom, the expected frequency counts, and the chi-square test statistic. Based on the chi-square statistic and the degrees of freedom, we determine the P-value.

$$DF = (r - 1) * (c - 1) = (2 - 1) * (3 - 1) = 2$$

$$E_{r,c} = (n_r * n_c) / n$$

$$E_{1,1} = (400 * 450) / 1000 = 180000/1000 = 180$$

$$E_{1,2} = (400 * 450) / 1000 = 180000/1000 = 180$$

$$E_{1,3} = (400 * 100) / 1000 = 40000/1000 = 40$$

$$E_{2,1} = (600 * 450) / 1000 = 270000/1000 = 270$$

$$E_{2,2} = (600 * 450) / 1000 = 270000/1000 = 270$$

$$E_{2,3} = (600 * 100) / 1000 = 60000/1000 = 60$$

$$X^2 = \sum [(O_{r,c} - E_{r,c})^2 / E_{r,c}]$$

$$X^2 = (200 - 180)^2/180 + (150 - 180)^2/180 + (50 - 40)^2/40 \\ + (250 - 270)^2/270 + (300 - 270)^2/270 + (50 - 60)^2/60$$

$$X^2 = 400/180 + 900/180 + 100/40 + 400/270 + 900/270 + 100/60$$

$$X^2 = 2.22 + 5.00 + 2.50 + 1.48 + 3.33 + 1.67 = 16.2$$

where DF is the degrees of freedom, r is the number of levels of gender, c is the number of levels of the voting preference, n_r is the number of observations from level r of gender, n_c is the number of observations from level c of voting preference, n is the number of observations in the sample, $E_{r,c}$ is the expected frequency count when gender is level r and voting preference is level c, and $O_{r,c}$ is the observed frequency count when gender is level r voting preference is level c.

The P-value is the probability that a chi-square statistic having 2 degrees of freedom is more extreme than 16.2.

We use the Chi-Square Distribution Calculator to find $P(X^2 > 16.2) = 0.0003$.

- **Interpret results.** Since the P-value (0.0003) is less than the significance level (0.05), we cannot accept the null hypothesis. Thus, we conclude that there is a relationship between gender and voting preference.

Note: If you use this approach on an exam, you may also want to mention why this approach is appropriate. Specifically, the approach is appropriate because the sampling method was simple random sampling, the variables under study were categorical, and the expected frequency count was at least 5 in each cell of the contingency table.

What is Hypothesis Testing?

A **statistical hypothesis** is an assumption about a population parameter. This assumption may or may not be true. **Hypothesis testing** refers to the formal procedures used by statisticians to accept or reject statistical hypotheses.

Statistical Hypotheses

The best way to determine whether a statistical hypothesis is true would be to examine the entire population. Since that is often impractical, researchers typically examine a random sample from the population. If sample data are not consistent with the statistical hypothesis, the hypothesis is rejected.

There are two types of statistical hypotheses.

- **Null hypothesis.** The null hypothesis, denoted by H_0 , is usually the hypothesis that sample observations result purely from chance.
- **Alternative hypothesis.** The alternative hypothesis, denoted by H_1 or H_a , is the hypothesis that sample observations are influenced by some non-random cause.

For example, suppose we wanted to determine whether a coin was fair and balanced. A null hypothesis might be that half the flips would result in Heads and half, in Tails. The alternative hypothesis might be that the number of Heads and Tails would be very different. Symbolically, these hypotheses would be expressed as

$$H_0: P = 0.5$$

$$H_a: P \neq 0.5$$

Suppose we flipped the coin 50 times, resulting in 40 Heads and 10 Tails. Given this result, we would be inclined to reject the null hypothesis. We would conclude, based on the evidence, that the coin was probably not fair and balanced.

Can We Accept the Null Hypothesis?

Some researchers say that a hypothesis test can have one of two outcomes: you accept the null hypothesis or you reject the null hypothesis. Many statisticians, however, take issue with the notion of "accepting the null hypothesis." Instead, they say: you reject the null hypothesis or you fail to reject the null hypothesis.

Why the distinction between "acceptance" and "failure to reject?" Acceptance implies that the null hypothesis is true. Failure to reject implies that the data are not sufficiently persuasive for us to prefer the alternative hypothesis over the null hypothesis.

Hypothesis Tests

Statisticians follow a formal process to determine whether to reject a null hypothesis, based on sample data. This process, called **hypothesis testing**, consists of four steps.

- State the hypotheses. This involves stating the null and alternative hypotheses. The hypotheses are stated in such a way that they are mutually exclusive. That is, if one is true, the other must be false.

- Formulate an analysis plan. The analysis plan describes how to use sample data to evaluate the null hypothesis. The evaluation often focuses around a single test statistic.
- Analyze sample data. Find the value of the test statistic (mean score, proportion, t statistic, z-score, etc.) described in the analysis plan.
- Interpret results. Apply the decision rule described in the analysis plan. If the value of the test statistic is unlikely, based on the null hypothesis, reject the null hypothesis.

Decision Errors

Two types of errors can result from a hypothesis test.

- **Type I error.** A Type I error occurs when the researcher rejects a null hypothesis when it is true. The probability of committing a Type I error is called the **significance level**. This probability is also called **alpha**, and is often denoted by α .
- **Type II error.** A Type II error occurs when the researcher fails to reject a null hypothesis that is false. The probability of committing a Type II error is called **Beta**, and is often denoted by β . The probability of not committing a Type II error is called the **Power** of the test.

Decision Rules

The analysis plan includes decision rules for rejecting the null hypothesis. In practice, statisticians describe these decision rules in two ways - with reference to a P-value or with reference to a region of acceptance.

- P-value. The strength of evidence in support of a null hypothesis is measured by the **P-value**. Suppose the test statistic is equal to S . The P-value is the probability of observing a test statistic as extreme as S , assuming the null hypothesis is true. If the P-value is less than the significance level, we reject the null hypothesis.
- Region of acceptance. The **region of acceptance** is a range of values. If the test statistic falls within the region of acceptance, the null hypothesis is not rejected. The region of acceptance is defined so that the chance of making a Type I error is equal to the significance level.

The set of values outside the region of acceptance is called the **region of rejection**. If the test statistic falls within the region of rejection, the null hypothesis is rejected. In such cases, we say that the hypothesis has been rejected at the α level of significance.

These approaches are equivalent. Some statistics texts use the P-value approach; others use the region of acceptance approach. On this website, we tend to use the region of acceptance approach.

One-Tailed and Two-Tailed Tests

A test of a statistical hypothesis, where the region of rejection is on only one side of the sampling distribution, is called a **one-tailed test**. For example, suppose the null hypothesis states that the mean is less than or equal to 10. The alternative hypothesis would be that the mean is greater than 10. The region of rejection would consist of a range of numbers located on the right side of sampling distribution; that is, a set of numbers greater than 10.

A test of a statistical hypothesis, where the region of rejection is on both sides of the sampling distribution, is called a **two-tailed test**. For example, suppose the null hypothesis states that the mean is equal to 10. The alternative hypothesis would be that the mean is less than 10 or greater than 10. The region of rejection would consist of a range of numbers located on both sides of sampling distribution; that is, the region of rejection would consist partly of numbers that were less than 10 and partly of numbers that were greater than 10.

Hypothesis Test for a Mean

This lesson explains how to conduct a hypothesis test of a mean, when the following conditions are met:

- The sampling method is simple random sampling.
- The sampling distribution is normal or nearly normal.

Generally, the sampling distribution will be approximately normally distributed if any of the following conditions apply.

- The population distribution is normal.

- The population distribution is symmetric, unimodal, without outliers, and the sample size is 15 or less.
- The population distribution is moderately skewed, unimodal, without outliers, and the sample size is between 16 and 40.
- The sample size is greater than 40, without outliers.

This approach consists of four steps: (1) state the hypotheses, (2) formulate an analysis plan, (3) analyze sample data, and (4) interpret results.

State the Hypotheses

Every hypothesis test requires the analyst to state a null hypothesis and an alternative hypothesis. The hypotheses are stated in such a way that they are mutually exclusive. That is, if one is true, the other must be false; and vice versa.

The table below shows three sets of hypotheses. Each makes a statement about how the population mean μ is related to a specified value M . (In the table, the symbol \neq means "not equal to".)

Set	Null hypothesis	Alternative hypothesis	Number of tails
1	$\mu = M$	$\mu \neq M$	2
2	$\mu > M$	$\mu < M$	1
3	$\mu < M$	$\mu > M$	1

The first set of hypotheses (Set 1) is an example of a two-tailed test, since an extreme value on either side of the sampling distribution would cause a researcher to reject the null hypothesis. The other two sets of hypotheses (Sets 2 and 3) are one-tailed tests, since an extreme value on only one side of the sampling distribution would cause a researcher to reject the null hypothesis.

Formulate an Analysis Plan

The analysis plan describes how to use sample data to accept or reject the null hypothesis. It should specify the following elements.

- Significance level. Often, researchers choose significance levels equal to 0.01, 0.05, or 0.10; but any value between 0 and 1 can be used.

- Test method. Use the one-sample t-test to determine whether the hypothesized mean differs significantly from the observed sample mean.

Analyze Sample Data

Using sample data, conduct a one-sample t-test. This involves finding the standard error, degrees of freedom, test statistic, and the P-value associated with the test statistic.

- Standard error. Compute the standard error (SE) of the sampling distribution.

$$SE = s * \sqrt{\left(\frac{1}{n} \right) * \left[\left(N - n \right) / \left(N - 1 \right) \right]}$$

where s is the standard deviation of the sample, N is the population size, and n is the sample size. When the population size is much larger (at least 20 times larger) than the sample size, the standard error can be approximated by:

$$SE = s / \sqrt{n}$$

- Degrees of freedom. The degrees of freedom (DF) is equal to the sample size (n) minus one. Thus, $DF = n - 1$.
- Test statistic. The test statistic is a t statistic (t) defined by the following equation.

$$t = (x - \mu) / SE$$

where x is the sample mean, μ is the hypothesized population mean in the null hypothesis, and SE is the standard error.

- P-value. The P-value is the probability of observing a sample statistic as extreme as the test statistic. Since the test statistic is a t statistic, use the t Distribution Calculator to assess the probability associated with the t statistic, given the degrees of freedom computed above. (See sample problems at the end of this lesson for examples of how this is done.)

Interpret Results

If the sample findings are unlikely, given the null hypothesis, the researcher rejects the null hypothesis. Typically, this involves comparing the P-value to the significance level, and rejecting the null hypothesis when the P-value is less than the significance level.

Test Your Understanding

In this section, two sample problems illustrate how to conduct a hypothesis test of a mean score. The first problem involves a two-tailed test; the second problem, a one-tailed test.

Problem 1: Two-Tailed Test

An inventor has developed a new, energy-efficient lawn mower engine. He claims that the engine will run continuously for 5 hours (300 minutes) on a single gallon of regular gasoline. From his stock of 2000 engines, the inventor selects a simple random sample of 50 engines for testing. The engines run for an average of 295 minutes, with a standard deviation of 20 minutes. Test the null hypothesis that the mean run time is 300 minutes against the alternative hypothesis that the mean run time is not 300 minutes. Use a 0.05 level of significance. (Assume that run times for the population of engines are normally distributed.)

Solution: The solution to this problem takes four steps: (1) state the hypotheses, (2) formulate an analysis plan, (3) analyze sample data, and (4) interpret results. We work through those steps below:

- **State the hypotheses.** The first step is to state the null hypothesis and an alternative hypothesis.

Null hypothesis: $\mu = 300$

Alternative hypothesis: $\mu \neq 300$

Note that these hypotheses constitute a two-tailed test. The null hypothesis will be rejected if the sample mean is too big or if it is too small.

- **Formulate an analysis plan.** For this analysis, the significance level is 0.05. The test method is a one-sample t-test.
- **Analyze sample data.** Using sample data, we compute the standard error (SE), degrees of freedom (DF), and the t statistic test statistic (t).

$$SE = s / \sqrt{n} = 20 / \sqrt{50} = 20/7.07 = 2.83$$

$$DF = n - 1 = 50 - 1 = 49$$

$$t = (x - \mu) / SE = (295 - 300)/2.83 = -1.77$$

where s is the standard deviation of the sample, x is the sample mean, μ is the hypothesized population mean, and n is the sample size.

Since we have a two-tailed test, the P-value is the probability that the t statistic having 49 degrees of freedom is less than -1.77 or greater than 1.77.

We use the t Distribution Calculator to find $P(t < -1.77) = 0.04$, and $P(t > 1.77) = 0.04$. Thus, the P-value = $0.04 + 0.04 = 0.08$.

- **Interpret results.** Since the P-value (0.08) is greater than the significance level (0.05), we cannot reject the null hypothesis.

Note: If you use this approach on an exam, you may also want to mention why this approach is appropriate. Specifically, the approach is appropriate because the sampling method was simple random sampling, the population was normally distributed, and the sample size was small relative to the population size (less than 5%).

Problem 2: One-Tailed Test

Bon Air Elementary School has 1000 students. The principal of the school thinks that the average IQ of students at Bon Air is at least 110. To prove her point, she administers an IQ test to 20 randomly selected students. Among the sampled students, the average IQ is 108 with a standard deviation of 10. Based on these results, should the principal accept or reject her original hypothesis? Assume a significance level of 0.01. (Assume that test scores in the population of engines are normally distributed.)

Solution: The solution to this problem takes four steps: (1) state the hypotheses, (2) formulate an analysis plan, (3) analyze sample data, and (4) interpret results. We work through those steps below:

- **State the hypotheses.** The first step is to state the null hypothesis and an alternative hypothesis.

Null hypothesis: $\mu \geq 110$

Alternative hypothesis: $\mu < 110$

Note that these hypotheses constitute a one-tailed test. The null hypothesis will be rejected if the sample mean is too small.

- **Formulate an analysis plan.** For this analysis, the significance level is 0.01. The test method is a one-sample t-test.
- **Analyze sample data.** Using sample data, we compute the standard error (SE), degrees of freedom (DF), and the t statistic test statistic (t).

$$SE = s / \sqrt{n} = 10 / \sqrt{20} = 10/4.472 = 2.236$$

$$DF = n - 1 = 20 - 1 = 19$$

$$t = (x - \mu) / SE = (108 - 110) / 2.236 = -0.894$$

where s is the standard deviation of the sample, x is the sample mean, μ is the hypothesized population mean, and n is the sample size.

Here is the logic of the analysis: Given the alternative hypothesis ($\mu < 110$), we want to know whether the observed sample mean is small enough to cause us to reject the null hypothesis.

The observed sample mean produced a t statistic test statistic of -0.894. We use the t Distribution Calculator to find $P(t < -0.894) = 0.19$. This means we would expect to find a sample mean of 108 or smaller in 19 percent of our samples, if the true population IQ were 110. Thus the P-value in this analysis is 0.19.

- **Interpret results.** Since the P-value (0.19) is greater than the significance level (0.01), we cannot reject the null hypothesis.

Note: If you use this approach on an exam, you may also want to mention why this approach is appropriate. Specifically, the approach is appropriate because the sampling method was simple random sampling, the population was normally distributed, and the sample size was small relative to the population size (less than 5%).

What is Analysis Of Variance - ANOVA

Analysis of variance (ANOVA) is an analysis tool used in statistics that splits the aggregate variability found inside a data set into two parts: systematic factors and random factors. The systematic factors have a statistical influence on the given data set, but the random factors do not. Analysts use the analysis of the variance test to determine the result that independent variables have on the dependent variable amid a regression study.

BREAKING DOWN Analysis Of Variance – ANOVA

The analysis of variance test is the initial step in analyzing factors that affect a given data set. Once the analysis of variance test is finished, an analyst performs additional testing on the methodical factors that measurably contribute to the data set's inconsistency. The analyst utilizes the analysis of the variance test results in an f-test to generate additional data that aligns with the proposed regression models.

The test allows comparison of more than two groups at the same time to determine whether a relationship exists between them. The test analyzes multiple groups to determine the types between and within samples. For example, a researcher might test students from multiple colleges to see if students from one of the colleges consistently outperform the others. Also, an R&D researcher might test two different processes of creating a product to see if one process is better than the other in terms of cost efficiency.

How to Use ANOVA

The type of ANOVA run depends on a number of factors. It is applied when data needs to be experimental. Analysis of variance is employed if there is no access to statistical software resulting in computing ANOVA by hand. It is simple to use and best suited for small samples. With many experimental designs, the sample sizes have to be the same for the various factor level combinations.

Analysis of variances is helpful for testing three or more variables. It is similar to multiple two-sample t-tests. However, it results in fewer type I errors and is appropriate for a range of issues. ANOVA groups differences by comparing the means of each group,

and includes spreading out the variance into diverse sources. It is employed with subjects, test groups, between groups and within groups.

Types of ANOVA

There are two types of analysis of variance: one-way (or unidirectional) and two-way. One-way or two-way refers to the number of independent variables in your Analysis of Variance test. A one-way ANOVA evaluates the impact of a sole factor on a sole response variable. It determines whether all the samples are the same. The one-way ANOVA is used to determine whether there are any statistically significant differences between the means of three or more independent (unrelated) groups.

A two-way ANOVA is an extension of the one-way ANOVA. With a one-way, you have one independent variable affecting a dependent variable. With a two-way ANOVA, there are two independents. For example, a two-way ANOVA allows a company to compare worker productivity based on two independent variables, say salary and skill set. It is utilized to observe the interaction between the two factors. It tests the effect of two factors at the same time.

F Distribution

The F distribution is the probability distribution associated with the f statistic. In this lesson, we show how to compute an f statistic and how to find probabilities associated with specific f statistic values.

The f Statistic

The **f statistic**, also known as an **f value**, is a random variable that has an F distribution. (We discuss the F distribution in the next section.)

Here are the steps required to compute an **f statistic**:

- Select a random sample of size n_1 from a normal population, having a standard deviation equal to σ_1 .
- Select an independent random sample of size n_2 from a normal population, having a standard deviation equal to σ_2 .
- The f statistic is the ratio of s_1^2/σ_1^2 and s_2^2/σ_2^2 .

The following equivalent equations are commonly used to compute an f statistic:

$$f = [s_1^2/\sigma_1^2] / [s_2^2/\sigma_2^2]$$

$$f = [s_1^2 * \sigma_2^2] / [s_2^2 * \sigma_1^2]$$

$$f = [X^2_1 / v_1] / [X^2_2 / v_2]$$

$$f = [X^2_1 * v_2] / [X^2_2 * v_1]$$

where σ_1 is the standard deviation of population 1, s_1 is the standard deviation of the sample drawn from population 1, σ_2 is the standard deviation of population 2, s_2 is the standard deviation of the sample drawn from population 2, X^2_1 is the chi-square statistic for the sample drawn from population 1, v_1 is the degrees of freedom for X^2_1 , X^2_2 is the chi-square statistic for the sample drawn from population 2, and v_2 is the degrees of freedom for X^2_2 . Note that degrees of freedom $v_1 = n_1 - 1$, and degrees of freedom $v_2 = n_2 - 1$.

The F Distribution

The distribution of all possible values of the f statistic is called an **F distribution**, with $v_1 = n_1 - 1$ and $v_2 = n_2 - 1$ degrees of freedom.

The curve of the F distribution depends on the degrees of freedom, v_1 and v_2 . When describing an F distribution, the number of degrees of freedom associated with the standard deviation in the numerator of the f statistic is always stated first. Thus, $f(5, 9)$ would refer to an F distribution with $v_1 = 5$ and $v_2 = 9$ degrees of freedom; whereas $f(9, 5)$ would refer to an F distribution with $v_1 = 9$ and $v_2 = 5$ degrees of freedom. Note that the curve represented by $f(5, 9)$ would differ from the curve represented by $f(9, 5)$.

The F distribution has the following properties:

- The mean of the distribution is equal to $v_2 / (v_2 - 2)$ for $v_2 > 2$.
- The variance is equal to $[2 * v_2^2 * (v_1 + v_1 - 2)] / [v_1 * (v_2 - 2)^2 * (v_2 - 4)]$ for $v_2 > 4$.

Cumulative Probability and the F Distribution

Every f statistic can be associated with a unique cumulative probability. This cumulative probability represents the likelihood that the f statistic is less than or equal to a specified value.

Statisticians use f_a to represent the value of an f statistic having a cumulative probability of $(1 - a)$. For example, suppose we were interested in the f statistic having a cumulative probability of 0.95. We would refer to that f statistic as $f_{0.05}$, since $(1 - 0.95) = 0.05$.

Of course, to find the value of f_a , we would need to know the degrees of freedom, v_1 and v_2 . Notationally, the degrees of freedom appear in parentheses as follows: $f_a(v_1, v_2)$. Thus, $f_{0.05}(5, 7)$ refers to value of the f statistic having a cumulative probability of 0.95, $v_1 = 5$ degrees of freedom, and $v_2 = 7$ degrees of freedom.

The easiest way to find the value of a particular f statistic is to use the F Distribution Calculator.

The use of the F Distribution Calculator is illustrated below in Problem 2.

Test Your Understanding

Problem**1**

Suppose you randomly select 7 women from a population of women, and 12 men from a population of men. The table below shows the standard deviation in each sample and in each population.

Population	Population standard deviation	Sample standard deviation
Women	30	35
Men	50	45

Compute the f statistic.

Solution A: The f statistic can be computed from the population and sample standard deviations, using the following equation:

$$f = [s_1^2/\sigma_1^2] / [s_2^2/\sigma_2^2]$$

where σ_1 is the standard deviation of population 1, s_1 is the standard deviation of the sample drawn from population 1, σ_2 is the standard deviation of population 2, and s_2 is the standard deviation of the sample drawn from population 2.

As you can see from the equation, there are actually two ways to compute an f statistic from these data. If the women's data appears in the numerator, we can calculate an f statistic as follows:

$$f = (35^2 / 30^2) / (45^2 / 50^2)$$

$$f = (1225 / 900) / (2025 / 2500)$$

$$f = 1.361 / 0.81 = 1.68$$

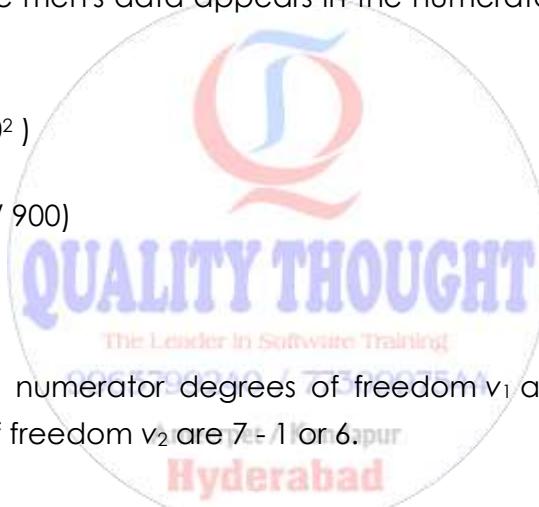
For this calculation, the numerator degrees of freedom v_1 are $7 - 1$ or 6; and the denominator degrees of freedom v_2 are $12 - 1$ or 11.

On the other hand, if the men's data appears in the numerator, we can calculate an f statistic as follows:

$$f = (45^2 / 50^2) / (35^2 / 30^2)$$

$$f = (2025 / 2500) / (1225 / 900)$$

$$f = 0.81 / 1.361 = 0.595$$



For this calculation, the numerator degrees of freedom v_1 are $12 - 1$ or 11; and the denominator degrees of freedom v_2 are $7 - 1$ or 6.

When you are trying to find the cumulative probability associated with an f statistic, you need to know v_1 and v_2 . This point is illustrated in the next example.

Problem 2

Find the cumulative probability associated with each of the f statistics from Example 1, above.

Solution: To solve this problem, we need to find the degrees of freedom for each sample. Then, we will use the F Distribution Calculator to find the probabilities.

- The degrees of freedom for the sample of women is equal to $n - 1 = 7 - 1 = 6$.
- The degrees of freedom for the sample of men is equal to $n - 1 = 12 - 1 = 11$.

Therefore, when the women's data appear in the numerator, the numerator degrees of freedom v_1 is equal to 6; and the denominator degrees of freedom v_2 is equal to 11. And, based on the computations shown in the previous example, the f statistic is equal to 1.68. We plug these values into the F Distribution Calculator and find that the cumulative probability is 0.78.

On the other hand, when the men's data appear in the numerator, the numerator degrees of freedom v_1 is equal to 11; and the denominator degrees of freedom v_2 is equal to 6. And, based on the computations shown in the previous example, the f statistic is equal to 0.595. We plug these values into the F Distribution Calculator and find that the cumulative probability is 0.22.



AI - Data Science Diploma

Python Programming

2019
EDITION

QualityThought®

Reach Us:

Quality Thought Infosystems Pvt. Ltd.
#208B, 2nd Floor, Nilgiri Block, Aditya Enclave
Ameerpet, Hyderabad, Telangana - 38

+ 91 - 9515151992

About Institute?

Quality Thought is a professional Training Organization for software developers, IT administrators, and other professionals. It's Located in Hyderabad, India. The training is offered in Four major modes: Classroom Room Trainings, Online instructor Led Trainings, Self-paced e-learning trainings, and Corporate Trainings.

About Course?

This is an Artificial Intelligence Engineer Master Course that is a comprehensive learning approach for mastering the domains of Artificial Intelligence, Data Science, Business Analytics, Business Intelligence, Python coding, and Deep Learning with TensorFlow. Upon completion of the training, you will be able to take on challenging roles in the artificial intelligence domain.

why we take course?

Artificial intelligence is one of the hottest domains being heralded as the one with the ability to disrupt companies cutting across industry sectors. This Quality Thought Artificial Intelligence Engineer Master Course will equip you with all the necessary skills needed to take on challenging and exciting roles in the artificial intelligence, data science, business analytics, Python, R statistical computing domains and grab the best jobs in the industry at top-notch salaries.

Courses Covered in AI Diploma:

Course 1: As a Future ready IT employee I am interested to learn foundations of Data Analytics with statistics and Tableau, R

Course 2: I want to redefine my Analytical knowledge into python programming for Machine Learning Engineer

Course 3: I want to apply statistics and Python on Machine Learning models for Predictions& classifications of Data in various industry segments for intelligent Business

Course 4: So now I am ready to apply machine Learning models to implement Recommendation systems and Creating Machine Learning service in the cloud(IBM WATSON,AWS)

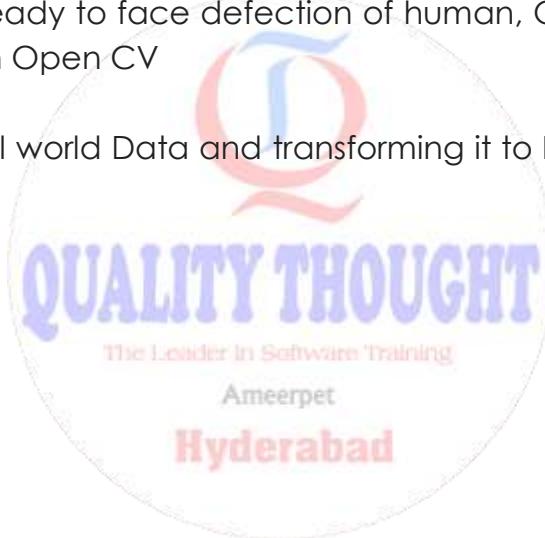
Course 5: Understanding tools of Bigdata and implementation using Machine Learning

Course 6: Applying Machine learning to speech Analytics to process Text using NLP

Course 7: Start Learning Neural Networks using Tensor flows and Keras for image classification and Data Extraction from Image(OCR)

Course 8: Now I am ready to face detection of human, Object using computer vision Technology with Open CV

Course 9: Sensing Real world Data and transforming it to Intelligent actions using IOT



Introduction

Python is an easy programming language and popular programming language too. Python is open-source and can get those libraries from python website python.org. In python, function and datatypes were implemented in C, C++. It can be used for many applications like data cleaning, databases and high-performance computing etc. It holds Data libraries like SciPy, NumPy etc.

Python is being used in Data Science technology development like Neural Networks, Artificial Intelligence, Statistics etc.

Python – Installation

You can understand the Installation of python on windows and Ubuntu in this tutorial. Before that let me give provide you the download link for downloading the Python.

For Linux and Unix Systems, below is the link to download Python:

<https://docs.python.org/3/using/unix.html>

For Windows system, below is the link to download Python:

<https://docs.python.org/3/using/windows.html>

Installation on Ubuntu or Linux Systems:

Python will come pre-installed on most of the Linux distributions. Just you need to give the python3 to start programming in the terminal.

If you don't have the Python, then get the source from

<https://www.python.org/downloads/source/>

follow the below commands.

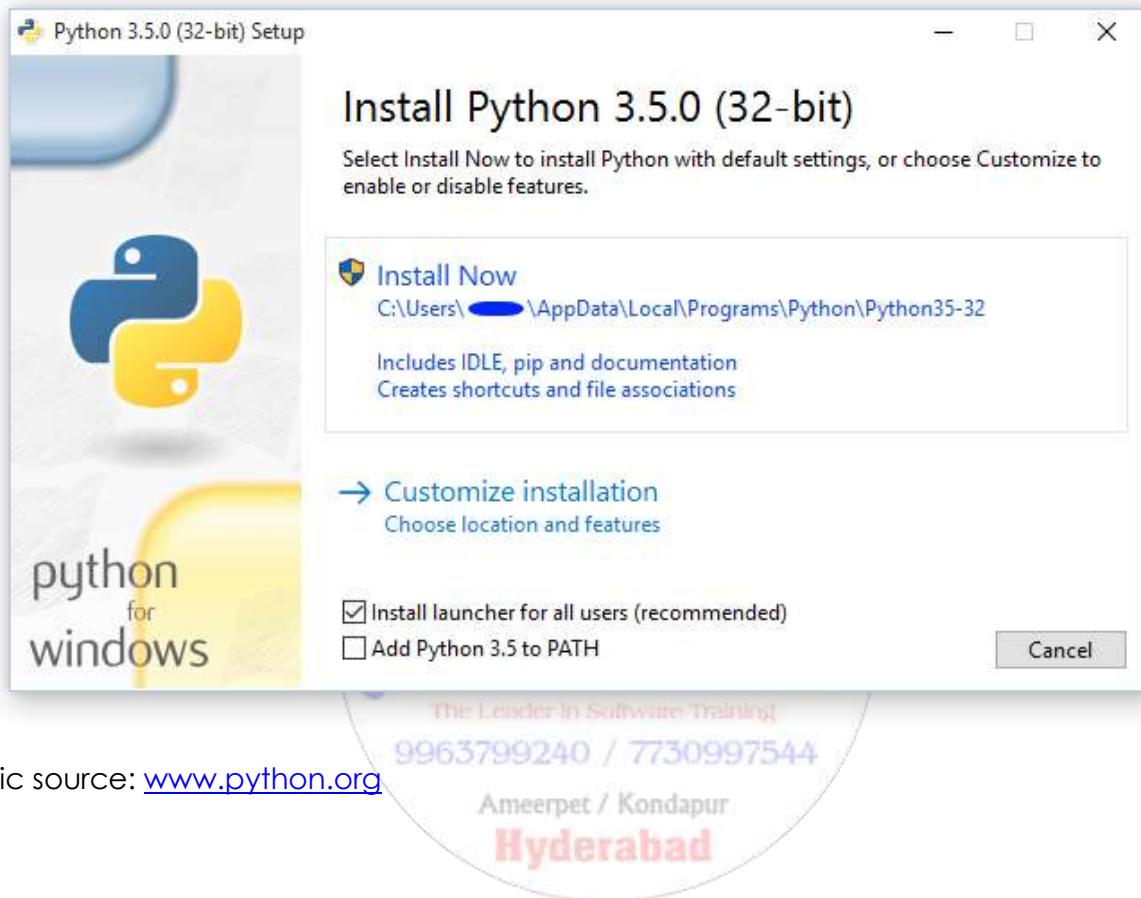
./configure

make

make install

Installation on windows systems:

Python installers are available to install python on 32-bit and 64-bit versions. Just download the installer and install the software. Once the installation opens the command prompt and give the python command to test.



Pic source: www.python.org

Python – Syntax

In this tutorial, you will learn the python syntax and an example about how to write a basic but popular Hello World print program.

Note: In this tutorial, we are using python version 3.5. Hence all the examples reflect the same results as like we execute in Python 3.5

Example:

```
>>> x="Hello World"
```

```
>>> print(x)
```

```
Hello World
```

```
>>>
```

In above Python terminal, x is a variable where we have assigned a value as hello world in double quotation as it is the string value. Then, we used print function to print the variable x which is in parenthesis. Do remember that we have given parenthesis for variable in print function.

Example:

```
>>> x=2
```

```
>>> print(x)
```

```
2
```

```
>>>
```

Python – Variables and Data types**Variable:**

A variable is the location in the memory to store the values. A variable may hold different types of values like numbers, strings etc. In Python, no need to declare a datatype for a variable. It will understand by the value that is assigned to the variable.

The name of variable or a function that we define can be called as Identifier. An Identifier must obey the below rules.

1. Identifiers can have letters, digits, underscores.
2. there is no definite in length.
3. All identifiers must start with letter or underscore. You cannot use digits.
4. Identifier should not be a keyword. (which is a reserved word for python)

Examples of value assignment to a variable:

```
>>>x = 10          # x is Integer
```

```
>>>y = 10.1        # y is Float
```

```
>>>z = "Hello"      # z is String
```

```
>>>x,y = y,x  # assign x value to y and y value to x
```

```
>>>a,b,c = 10,20,30 # assign a,b,c values sperated by comma at a time
```

```
>>>print(x,y,a,b,c)  # printing all the values
```

Data Types:

Python has different types of data types as below.

1. Numbers
2. String
3. List
4. Tuple
5. Dictionary
6. Boolean

Python – Numbers

In Python, Numbers can be defined in three ways.

1. Integer
2. Float
3. Complex

Let us understand how to work with Integer numbers

```
>>>x = 10 # assigning the integer value to variable x
```

```
>>>x
```

```
10
```

we can identify which type of value the variable x is holding

```
>>>x=10
```

```
>>>type(x)
```

```
<class 'int'>
```

Let us understand how to work with Float numbers

```
>>>y=10.1 # assigning the float value to variable y
```

```
>>>y
```

```
10.1
```

Let us identify the type of value the variable y is holding

```
>>>y=10.1
```

```
>>>type(y)
```

```
<class 'float'>
```

Let us understand how to work with complex type

```
>>>x=4+2j # 4 is the real part and 2j is the imaginary part
```

We can perform the various calculations with these numbers. Let us see few examples below.

```
>>>5+5
```

```
10
```

```
>>>5*5
```

```
25
```

```
>>>5-4
```

```
1
```

Python – Strings

In this tutorial, we will work on the Python Strings where we can learn about the manipulation of Strings, using String Operators and string methods and Functions. First, let us understand that how do we declare the strings in python programming language. We can declare and print the strings by placing them in single Quotes ('..'), Double Quotes (".."), and using the print function too. Python Strings are Immutable (An Object with a fixed value).

Using the Strings in single quotes ('...')

```
>>> 'hello world'
```

```
'hello world'
```

Below command will give an error.

```
>>> 'let's start' # this will give us an error
```

```
File "<stdin>", line 1
```

```
'let's do it'
```

```
^
```

```
SyntaxError: invalid syntax
```

**To overcome that we must use escape character **

```
>>> 'let\'start'  
"let's start"
```

Using the Strings in Double quotes ("...")

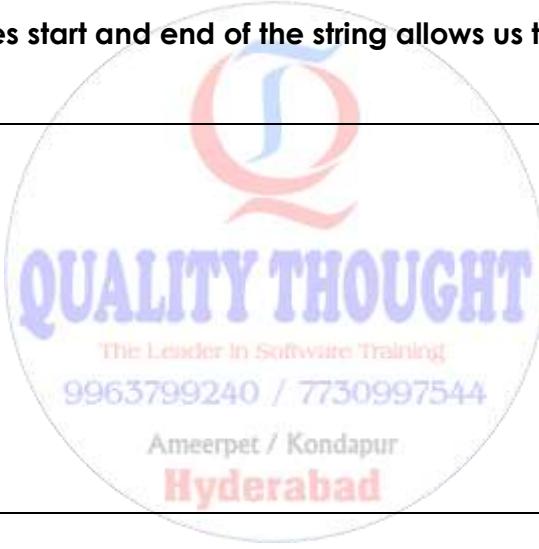
```
>>> "let's start"      # using double quotes to avoid escape character  
"let's start"
```

Using the Strings in Print() function

```
>>>print("let's start")      # we have enclose the strings in double quotation inside print  
function  
let's start
```

Using the 3 double quotes start and end of the string allows us to print the data including spaces and newlines.

```
>>>print("""let's  
...start  
...Now""")  
let's  
start  
now
```



String Concatenation:

Multiple Strings can be concatenated using (+) symbol. Let us see the example of concatenating the strings.

Example:

```
>>> x="hello"  
>>> y="world"  
>>>x+y  
'helloworld'
```

String Repetition:

String repetition can be performed by using the (*) symbol. Let us see the example of repetition of strings.

Example:

```
>>> 3*"hello"  
'hellohellohello'
```

Strings are indexed with each character in a memory location when assigned to a variable. The indexed number starts from zero '0' from first character till the end of the string. Whereas, reverse indexing starts with '-1' from right to left until the starting character. Let us try few examples of retrieving the characters from a word PYTHON in either ways.

Example:

```
>>> x="P Y T H O N"  # Word python is written without spaces  
0 1 2 3 4 5  
-6-5-4-3-2-1  
>>>x[3]  
'H'  
>>>x[-5]  
'Y'  
>>>x[:4]          # Starting from first character, 4th position excluded  
'PYTH'  
>>>x[:-4]         # Starting from fourth character from right, 4th position excluded  
'PY'  
>>>x[0:]          # Starting from first character till end  
'PYTHON'  
>>>x[-6:]         # Starting from -6th position until start ie., -1 position  
'PYTHON'
```

String Methods in Python:

Python String Methods	Description
capitalize()	Returns the String with first Character as Capital Letter
casefold()	Returns a casefolded copy
center(width[, fillchar])	This will pads the string with a character specified
count(sub[, start[, end]])	Returns the number of occurrences of substring in string
encode(encoding="utf-8", errors="strict")	returns an encoded string
endswith(suffix[, start[, end]])	Check the string if it ends with the specified
expandtabs(tabsize=8)	Replace the tab with space
find(sub[, start[, end]])	returns the highest index
format(*args, **kwargs)	formats the string
format_map(mapping)	formats the string except the mapping is directly used
index(sub[, start[, end]])	returns the index of substring
isalnum()	checks for alphanumeric Char
isalpha()	Checks if all characters are Alphabets
isdecimal()	Checks for decimal characters
isdigit()	Checks for digit char
isidentifier()	checks for valid Identifier
islower()	checks for lowercase of all alphabets in string
isnumeric()	Checks for Numeric Char
isprintable()	Checks for Printable Char
isspace()	Checks for Whitespace Characters
istitle()	Returns true if the string is titlecased
isupper()	Checks if all characters are Uppercase
join(iterable)	returns concatenated string
ljust(width[, fillchar])	returns left-justified string

lower()	returns lowercased string
lstrip([chars])	Removes Leading Characters
partition(sep)	returns a tuple
replace(old, new[, count])	replaces the substring
rfind(sub[, start[, end]])	Returns the Highest Index
rindex(sub[, start[, end]])	Returns Highest Index but raises when substring is not found
rjust(width[, fillchar])	Returns the string right justified
rpartition(sep)	Returns a tuple
rsplit(sep=None, maxsplit=-1)	Splits String From Right
rstrip([chars])	Removes Trailing Characters
split(sep=None, maxsplit=-1)	Splits String from Left
splitlines([keepends])	Splits String at Lines
startswith(prefix[, start[, end]])	Checks if String Starts with the Specified String
strip([chars])	Removes Both Leading and Trailing Characters
swapcase()	swaps uppercase characters to lowercase and vice versa
title()	Returns a Title Cased String
translate(table)	returns mapped charactered string
upper()	returns uppercased string
zfill(width)	Returns a Copy of The String Padded With Zeros

Sequence in Python can be defined with a generic term as an ordered set which can be classified as two sequence types. They are mutable and immutable. There are different types of sequences in python. They are Lists, Tuples, Ranges.

Lists: Lists will come under mutable type in which data elements can be changed.

Tuples: Tuples are also like Lists which comes under immutable type which cannot be changed.

Ranges: Ranges is mostly used for looping operations and this will come under immutable type.

The common Sequence Operations are listed below:

x in s: Returns true if an item in s is equal to x

x not in s : Returns false if an item in s is equal to x

s+t : concatenation

s*n : adding s to itself n number of times

s[i] : ith item of s, index starts from zero

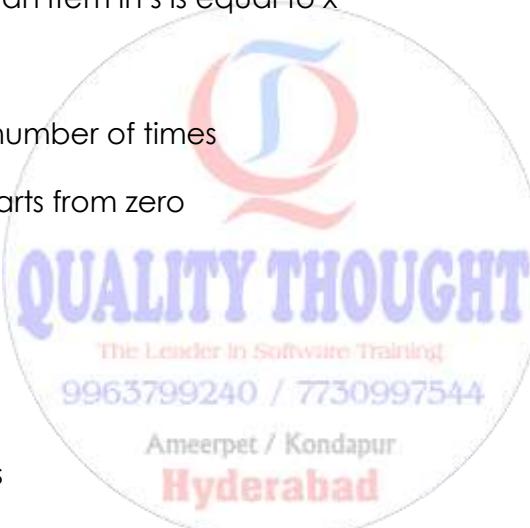
s[i:j]: slice of s from i to j

len(s) : length of s

max(s) : largest item in s

min(s) : smallest item of s

s.count(x): total number of occurrences of x in s



Python – Lists

Python Lists holds the data of any datatype like an array of elements and these are mutable means the possibility of changing the content or data in it. List can be created by giving the values that are separated by commas and enclosed in square brackets. Let us see different types of value assignments to a list.

Example:

```
List1=[10,20,30,40,50];
```

```
List2=['A','B','C','D'];
```

```
List3=[10.1,11.2,12.3];
```

```
List4=['html','java','oracle'];  
List5=['html',10.1,10,'A'];
```

As we know the way strings can be accessed, same way Lists can be accessed. Below is example of indexing in python for your understanding again.

Example:

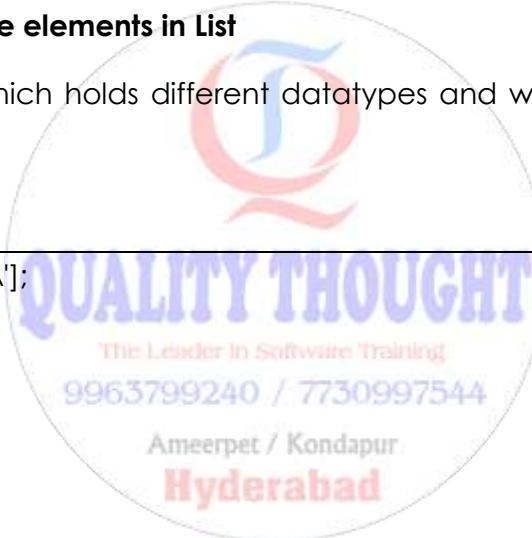
```
List1=[10,20,30,40,50];  
0 1 2 3 4 ---> Forward Indexing  
-5 -4 -3 -2 -1 ---> Backward Indexing
```

Accessing and slicing the elements in List

Now let us take a list which holds different datatypes and will access the elements in that list.

Example:

```
>>> list5=['html',10.1,10,'A'];  
>>> list5[0]  
'html'  
>>> list5[1:2];  
[10.1]  
>>> list5[-2:-1];  
[10]  
>>> list5[:-1];  
['html', 10.1, 10]  
>>> list5[:-2];  
['html', 10.1]  
>>> list5[1:-2];  
[10.1]
```



```
>>>list5[1:-1];  
[10.1, 10]  
>>> list5[-1];  
'A'  
>>> list5[3:];  
['A']
```

Using Functions with Lists:**Example:**

```
>>> list5=['html',10.1,10,'A'];  
>>>len(list5)  
4  
>>> 10 in list5  
True  
>>> 'html' in list5  
True  
>>>num=[10,20,30,40];  
>>> sum(num)  
100  
>>> max(num)  
40  
>>> min(num)  
10
```

**Checking if the Lists are mutable:**

Example:

```
>>>score=[10,20,30,80,50]  
>>> score  
[10, 20, 30, 80, 50]  
>>>score[3]=40  
>>> score  
[10, 20, 30, 40, 50]
```

List Comprehension:

List comprehension works like iterate operations as mentioned below.

Syntax:

```
[x for x in iterable]
```

Example:

```
>>>var=[x for x in range(5)]; 9963799240 / 7730997544  
>>>var  
[0, 1, 2, 3, 4]  
>>>var=[x+1 for x in range(5)];  
>>>var  
[1, 2, 3, 4, 5]  
>>>var=[x for x in range(5) if x%3==0];  
>>>var  
[0, 3]
```

Adding Elements to a list:

We can add two lists as shown in the below example.

Example:

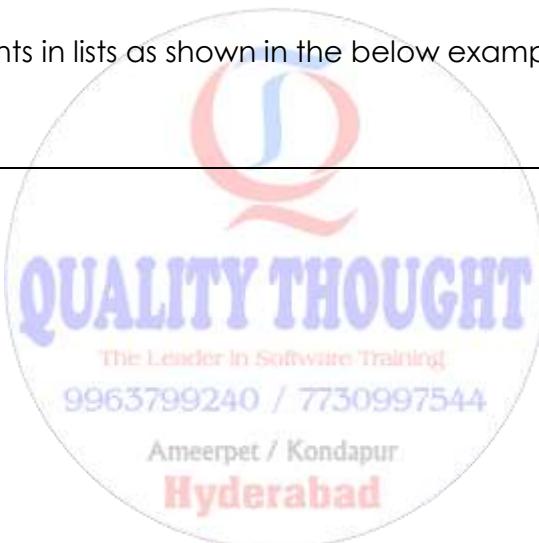
```
>>> var1=[10,20]  
>>> var2=[30,40]  
>>> var3=var1+var2  
>>> var3  
[10, 20, 30, 40]
```

Replicating elements in Lists:

we can replicate elements in lists as shown in the below example.

Example:

```
>>> var1*2  
[10, 20, 10, 20]  
  
>>> var1*3  
[10, 20, 10, 20, 10, 20]  
  
>>> var1*4  
[10, 20, 10, 20, 10, 20, 10, 20]
```

**Appending elements in Lists:**

We can append an element to an existing list as shown in the below example.

Example:

```
>>> var1.append(30)  
>>> var1
```

```
[10, 20, 30]
```

```
>>> var1.append(40)
```

```
>>> var1
```

```
[10, 20, 30, 40]
```

```
>>> var2
```

```
[30, 40]
```

Python – Tuples

Tuples are generally used to store the heterogeneous data which is immutable. Even Tuple looks like Lists but Lists are mutable. To create a tuple, we need to use the comma which separates the values enclosed parentheses.

Example:

```
>>> tup1=()      # Creating an empty tuple
>>> tup1
()
>>> tup1=(10)
>>> tup1
10
>>> tup1=(10,20,30);
>>> tup1
(10, 20, 30)
>>> tup1=tuple([1,1,2,2,3,3])
>>> tup1
(1, 1, 2, 2, 3, 3)
>>> tup1=("tuple")
```

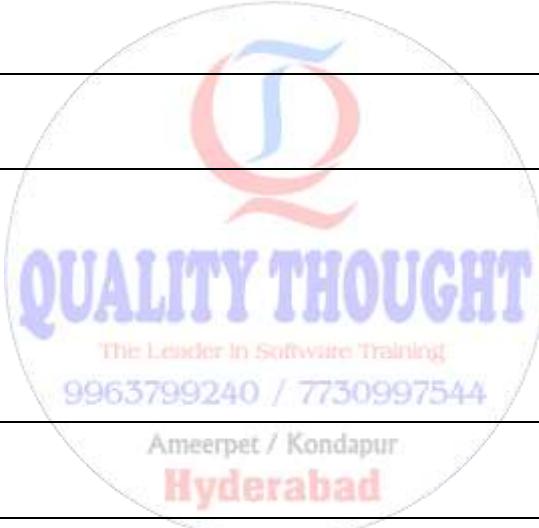
```
>>> tup1  
'tuple'
```

Using Functions with Tuples:

```
>>> tup1=(10,20,30);  
>>> max(tup1)  
30  
>>> min(tup1)  
10  
>>>len(tup1)  
3
```

Operators with Tuples:

```
>>> 20 in tup1  
True  
>>> 30 not in tup1  
False
```

**Slicing in Tuples:**

```
>>> tup1[0:4]  
(10, 20, 30)  
>>> tup1[0:1]  
(10,)  
>>> tup1[0:2]  
(10, 20)
```

Python – Dictionary

Dictionaries are created or indexed by key-value pairs. In which keys are immutable type.

Tuples can be used as keys, but lists cannot be used as keys. Just because lists are mutable.

Generally, the key-value pairs which are stored in the Dictionary can be accessed with the

key. we can delete a key value too. Let us see some examples.

Example:

```
>>> score={'maths':80,'physics':70,'chemistry':85}

>>> score

{'physics': 70, 'maths': 80, 'chemistry': 85}

>>> score['maths']

80

>>> del score['maths']

>>> score['maths']

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'maths'

>>> score

{'physics': 70, 'chemistry': 85}

>>> score.keys()

dict_keys(['physics', 'chemistry'])

>>> keys=score.keys()

>>> keys

dict_keys(['physics', 'chemistry'])

>>> list(keys)

['physics', 'chemistry']
```

**Python – Ranges**

Range a kind of data type in python which is an immutable. Range will be used in for loops for number of iterations. Range is a constructor which takes arguments and those must be integers. Below is the syntax.

Syntax:

```
class range(stop)
class range(start, stop[, step])
```

stop: the value of stop parameter.

step: the value of step parameter. If the value is omitted, it defaults to 1.

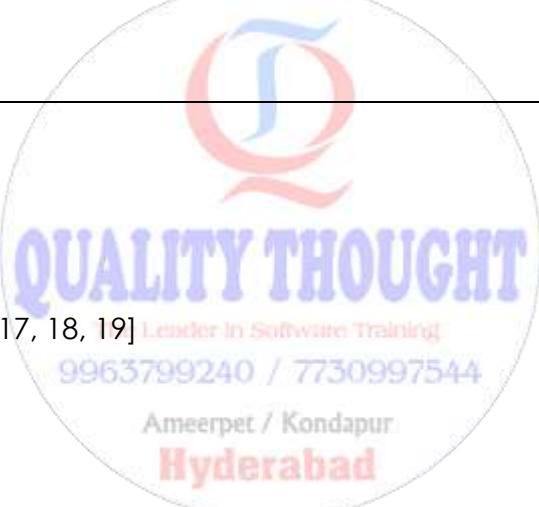
start: the value of start parameter. If the value is omitted, it defaults to zero.

Examples:

```
>>>list(range(5))
[0, 1, 2, 3, 4]

>>>list(range(10,20))
[10, 11, 12, 13, 14, 15, 16, 17, 18, 19]

>>>list(range(10,20,5))
[10, 15]
```



```
>>>list(range(10,20,2))
[10, 12, 14, 16, 18]
```

It will not take the float numbers

```
>>>list(range(0,0.1))
```

Traceback (most recent call last):

```
  File "<stdin>", line 1, in <module>
```

```
TypeError: 'float' object cannot be interpreted as an integer
```

```
>>>list(range(0,2))
```

```
[0, 1]
```

```
>>>list(range(0,1))
```

```
[0]
```

```
>>>list(range(0,10,5))
```

```
[0, 5]
```

Python – Sets

In this tutorial, we will learn about sets in python. A set is a datatype which holds an unordered collection with immutable and no duplicate elements. By the name, Set can be used for various mathematical operations. Mathematical operation may be union, intersection or difference, etc. Let us see the example of using the Set below.

Example:

```
>>> set1={'html','c','java','python','sql'}  
>>> print(set1)  
{'c', 'python', 'sql', 'html', 'java'}  
# Below we have given duplicates  
>>> set1={'html','c','java','python','sql','java'}  
# we can observe that duplicates are ignored  
>>> print(set1)  
{'c', 'python', 'java', 'html', 'sql'}  
>>> set1  
{'c', 'python', 'java', 'html', 'sql'}
```

Membership testing in Sets:

```
>>> set1={'html','java','python','sql','java'}  
>>> set1  
{'python', 'java', 'html', 'sql'}  
>>> print(set1)  
{'python', 'java', 'html', 'sql'}
```

```
>>> 'c' in set1
```

```
False
```

```
>>> 'java' in set1
```

```
True
```

Sets Operations in Python:

```
>>> set1={'html','java','python','sql','java'}
```

```
>>> set2={'html','oracle','ruby'}
```

```
# Unique words in set1
```

```
>>> set1
```

```
{'python', 'java', 'html', 'sql'}
```

```
>>> set2
```

```
{'ruby', 'html', 'oracle'}
```

```
# words in set1 but not in set2
```

```
>>> set1-set2
```

```
{'python', 'java', 'sql'}
```

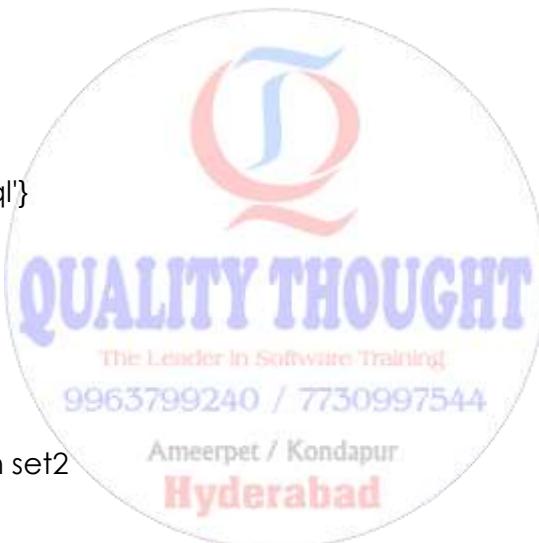
```
# Words in set1 or set2 or both
```

```
>>> set1 | set2
```

```
{'ruby', 'html', 'oracle', 'python', 'java', 'sql'}
```

```
# Words in both set1 and set2
```

```
>>> set1 & set2
```



```
{'html'}
```



```
# Words in set1 or set2 but not both
```



```
>>> set1^set2
```



```
{'oracle', 'python', 'sql', 'ruby', 'java'}
```

Python - Operators

Operators in Python helps us to perform the mathematical operations with numbers. There are different operators in Python as below.

Operators	Description
//	Integerdivision
+	addition
-	subtraction
*	multiplication
/	Float division
%	Provide remainder after division(Modulus)
**	Perform exponent (raise to power)

Let us try implementing every operator now.

1. Addition: symbol used (+)

```
>>> 10+10
```

```
20
```

```
>>>20+30
```

```
50
```

```
>>>50+50
```

```
100
```

2. Substration: symbol used (-)

```
>>>20-10
```

10

```
>>>50-40
```

10

```
>>>100-30
```

70

3. multiplication: Symbol used (*)

```
>>>5*2
```

10

```
>>>10*2
```

20

```
>>>20*2
```

40



4. Float Division: This will divide and provide the result in floating value and the symbol used (/)

```
>>>5/2
```

2.5

```
>>>10/2
```

5.0

5. Integer Division: This will divide and truncate the decimal and provide the Integer value and the symbol used (//)

```
>>>5//2
```

2

```
>>>7//2
```

6. Exponentiation Operator: This will help us to calculate a power b and return the result

```
>>>10**3 # This mean 10*10*10
```

```
1000
```

7. Modulus Operator: This will provide the remainder after the calculation and symbol used (%)

```
>>>10%3
```

```
1
```

What if we want to work with multiple operators at a time. Here comes the Operator precedence in Python.

Operator	Description
lambda	Lambda expression
if – else	Conditional expression
or	Boolean OR
and	Boolean AND
not x	Boolean NOT
in, not in, is, is not, <, <=, >, >=, !=, ==	Comparisons, including membership tests and identity tests
	Bitwise OR
^	Bitwise XOR
&	Bitwise AND
<<, >>	Shifts

+, -	Addition and subtraction
*, @, /, //, %	Multiplication, matrix multiplication division, remainder [5]
+x, -x, ~x	Positive, negative, bitwise NOT
**	Exponentiation [6]
await x	Await expression
x[index], x[index:index], x(arguments...), x.attribute	Subscription, slicing, call, attribute reference
(expressions...), [expressions...], {key: val ue...}, {expressions...}	Binding or tuple display, list display, dictionary display, set display

Python -If.. Else.. Statements

In this tutorial, we will discuss about the "if" Condition statement. Let us understand how this "if" statements work. There will be 2 parts in the "if" statement. They are "if" and "elif", "else" which is optional. when the "if" condition satisfies then it executes the program inside that else it execute the program inside "elif" or "else" statements.

Syntax:

if Condition:

 program of if

elif test expression:

 program of elif

else:

 program of else

Below is the example of If.. Else:

Example 1:

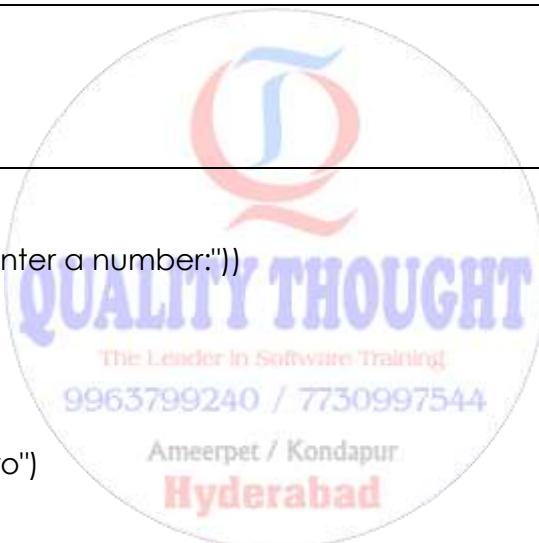
```
>>> x=int(input("Please enter a number:"))
```

```
Please enter a number:10
```

```
>>> if x<0:  
...     print("negative is zero")  
... elif x==1:  
...     print("single")  
... else:  
...     print("positive and greater than 1")  
...  
positive and greater than 1
```

Example 2:

```
>>> x=int(input("Please enter a number:"))  
Please enter a number:1  
>>> if x<0:  
...     print("negative is zero")  
... elif x==1:  
...     print("single")  
... else:  
...     print("positive and greater than 1")  
...  
Single
```



Python – For Loop

In this tutorial, we will learn about for loop. In Python, for loop is used to iterate over the sequence of elements (the sequence may be list, tuple or strings.. etc). Below is the syntax.

Syntax:

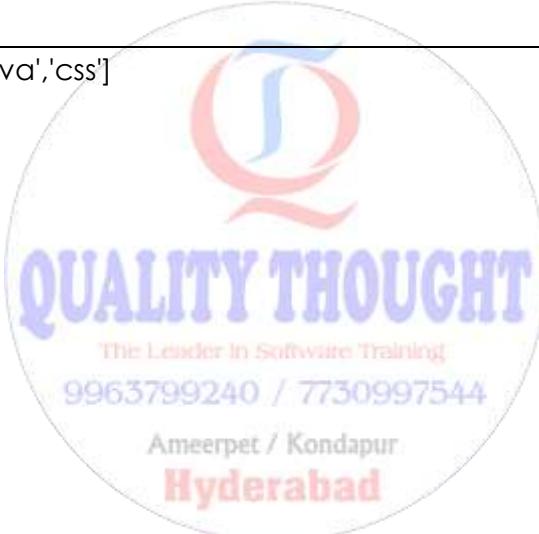
```
for_stmt ::= "for" target_list "in" expression_list ":" suite
           ["else" ":" suite]
```

In the below example, we have given the list of strings as courses and the for loop created to iterate through all the strings to print the course and the length of the course name.

Example:

```
>>> courses=['html','c','java','css']

>>> for i in courses:
...     print(i, len(i))
...
html 4
c 1
java 4
css 3
>>>
```



In the below example, for loop iterates through the list of numbers. In the immediate step, if statement filters only the numbers less than 50, else it will display "no values" for rest of the iterations.

Example:

```
>>> x=[10,20,30,40,50,60]

>>> x
[10, 20, 30, 40, 50, 60]
```

```
>>> for i in x:  
...     if i<50:  
...         print(i)  
...     else:  
...         print("no values")
```

Below is the output:

```
10  
20  
30  
40  
no values  
no values
```

Python – While Loop

In this tutorial, we will learn about while loop. In python, while loop is used to iterate until the condition is satisfied. If the condition given is not satisfied in the first iteration itself, the block of code inside the loop will not get executed.

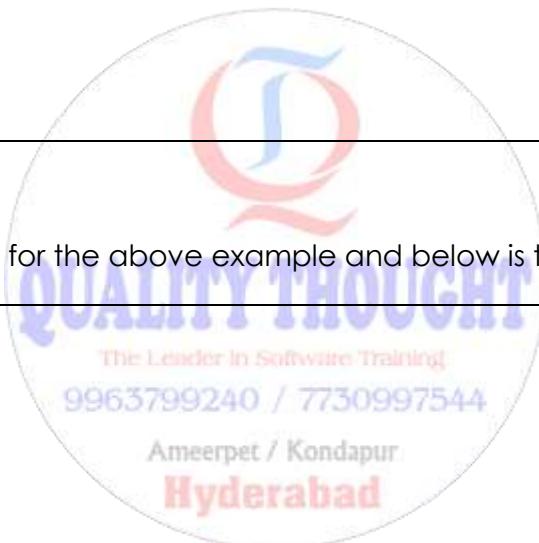
In the below example, we have assigned the value of x as zero and started the while loop until the value of x is less than 10 and print the values.

Example:

```
>>> x=0  
  
>>> while x<10:  
...     x=x+1  
...     print(x)  
...
```

Below is the Output:

```
1
2
3
4
5
6
7
8
9
10
```



Just changed the values for the above example and below is the output.

```
>>> x=100
>>> while x<110:
...     x=x+1
...     print(x)
...
```

Below is the output:

```
101
102
103
104
105
106
```

107
108
109
110

Python – Break

In this tutorial, we will learn about the Break statement in Python. Break statement is used to terminate the loop program at a point.

Let us understand the below example which do not have "break" statement will go through multiple iterations till the value of i becomes 109.

Example:

```
>>>for i in range(100,110):  
...     print('Assigned value of i is ',i)  
...     for num in range(100,i):  
...         print(i,num)  
...
```

Below is the output:

```
Assigned value of i is 100  
Assigned value of i is 101  
101 100  
Assigned value of i is 102  
102 100  
102 101  
Assigned value of i is 103  
103 100  
103 101
```



103 102

Assigned value of i is 104

104 100

104 101

104 102

104 103

Assigned value of i is 105

105 100

105 101

105 102

105 103

105 104

Assigned value of i is 106

106 100

106 101

106 102

106 103

106 104

106 105

Assigned value of i is 107

107 100

107 101

107 102

107 103

107 104



```
107 105
107 106
Assigned value of i is 108
108 100
108 101
108 102
108 103
108 104
108 105
108 106
108 107
Assigned value of i is 109
109 100
109 101
109 102
109 103
109 104
109 105
109 106
109 107
109 108
```



Now, let us break the loop when the value of i becomes 105. Below is the code and output for clarification.

Example:

```
>>>for i in range(100,110):
...     print('Assigned value of i is ',i)
```

```
... fornum in range(100,i):  
...     print(i,num)  
...     if i==105:  
...         break  
...
```

Below is the Ouput:

Assigned value of i is 100

Assigned value of i is 101

101 100

Assigned value of i is 102

102 100

102 101

Assigned value of i is 103

103 100

103 101

103 102

Assigned value of i is 104

104 100

104 101

104 102

104 103

Assigned value of i is 105

105 100

105 101

105 102



105 103
105 104

Python – Continue

In this tutorial, we will learn about the Continue statement in Python. Continue Statement is used to take the control to top of the loop for next iteration leaving the rest of the statements in the loop without execution.

Below is the similar example of a for loop:

```
>>>for i in range(10):  
...     if (i==4 or i ==8):  
...         continue  
...     print(i)  
...
```

Below is the Output:

```
0  
1  
2  
3  
5  
6  
7  
9
```



Below is another example where the printing of even numbers are ignored:

```
>>>for i in range(100,110):  
...     if i%2==0:  
...         continue
```

```
...     print(i,' is an even number')
...
... else:
...
...     print(i,' is an Odd number')
```

Below is the output:

```
101 is an Odd number
103 is an Odd number
105 is an Odd number
107 is an Odd number
109 is an Odd number
```

Python – Pass

Pass statement is used when there is a situation where a statement is required for syntax in the code, but which should not to be executed. So that, When the program executes that portion of code will not be executed.

In the below example, we can observe that the pass statement in if condition was not executed.

Example:

```
>>>for i in range(100,104):
...
...     print('Assigned value of i is ',i)
...
...     for num in range(100,i):
...
...         print(i,num)
...
...     if num==102:
...
...         pass
```

Below is the Output:

```
Assigned value of i is 100
Assigned value of i is 101
101 100
```

Assigned value of i is 102

102 100

102 101

Assigned value of i is 103

103 100

103 101

103 102

In the below example, we can observe that the pass statement is not mentioned. Hence resulted an error.

Example:

```
>>>for i in range(100,104):  
...     print('Assigned value of i is ',i)  
...     for num in range(100,i):  
...         print(i,num)  
...     if num==102:  
...  
File "<stdin>", line 6  
    ^
```

IndentationError: expected an indented block

Python - Date & Time

In python, datetime is a module which provides different classes to work with dates and times.

The types of objects in datetime are as below.

1. date
2. time

3. datetime
4. timedelta
5. tzinfo
6. timezone

Date Object:

Date object depicts the date as date(year, month, day) in an ideal Gregorian calendar. Syntax of the Date class is represented as below.

Syntax:

```
class datetime.date(year,month,day)
```

All the arguments are integers. Every argument has its own range of values as below.

1. YEAR: MINYEAR - MAXYEAR (1 - 9999)
2. MONTH: 1 - 12
3. DAY: 1 - number of days in the given month and year.

Now let us work with date object and its methods which serves different requirements with an example. The below example shows the current date in different formats.

Example:

```
# need to import the date
```

```
>>> from datetime import date
```

```
# method today() shows the today's date
```

```
>>> today=date.today()
```

```
# let us see the date
```

```
>>> today
```

```
datetime.date(2017, 11, 7)
```

```
>>> today=x
```

```
>>> x
```

```
time.struct_time(tm_year=2017, tm_mon=8, tm_mday=15, tm_hour=0, tm_min=0,  
tm_sec=0, tm_wday=1, tm_yday=227, tm_isdst=-1)
```

```
# assigning the date
```

```
>>> d=date(2017,11,7)
```

```
>>> x=d.timetuple()
```

```
# Print the date values from the tuple by year, month, day ..etc
```

```
>>> for i in x:
```

```
...     print(i)
```

```
...
```

Below is the output:

```
2017
```

```
11
```

```
7
```

```
0
```

```
0
```

```
0
```

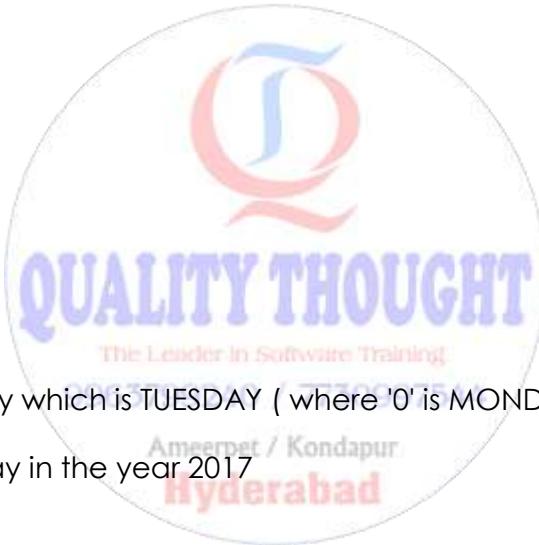
```
1
```

```
# week day which is TUESDAY ( where '0' is MONDAY)
```

```
311
```

```
# 311 th day in the year 2017
```

```
-1
```



```
# print the iso format of date
```

```
>>> d.isoformat()
```

```
'2017-11-07'
```

```
# String formats of date
```

```
>>> d.strftime("%d/%m/%u")
```

```
'07/11/2'
```

```
>>> d.strftime("%A%d.%B %Y")
```

```
'Tuesday07.November 2017'
```

```
>>> 'The {1} is {0:%d}, the {2} is {0:%B}'.format(d, "day", "month")  
'The day is 07, the month is November.'
```

Time object:

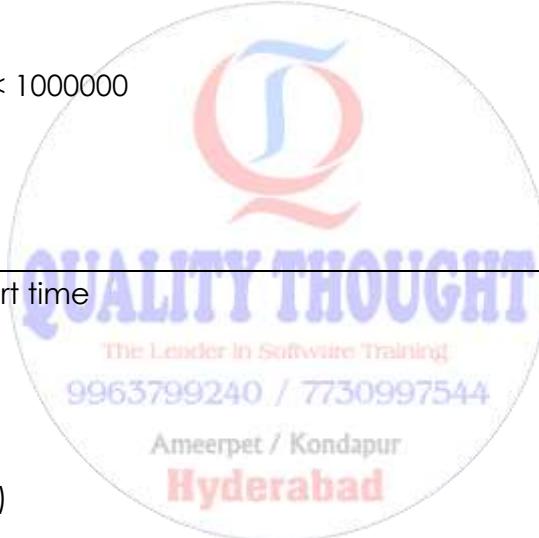
A time object which gives the information about time of any particular day subject to the requirements. The syntax of the time object constructor is given below.

Syntax:

1. HOUR: 0 - 24
2. MINUTE: 0 to < 60
3. SECOND: 0 to < 60
4. MICROSECOND: 0 to < 1000000
5. fold in [0,1]

Example:

```
>>> from datetime import time  
>>> t=time(12,12,12)  
>>> t  
datetime.time(12, 12, 12)  
>>>t.isoformat()  
'12:12:12'
```

**Datetime Object:**

Datetime object is a combination of both date and time information which can provide the functions from date object and time object.

Syntax:

```
class datetime.datetime(year, month, day, hour=0, minute=0, second=0,  
microsecond=0, tzinfo=None, *, fold=0)
```

All the arguments are integers. Each argument has its own range of values as below.

1. YEAR: MINYEAR – MAXYEAR(1 - 9999)
2. MONTH: 1 - 12
3. DAY: 1 - number of days in the given month and year.
4. HOUR: 0 - 24
5. MINUTE: 0 to < 60
6. SECOND: 0 to < 60
7. MICROSECOND: 0 to < 1000000
8. fold in [0,1]

Now let us work with datetime object and its methods which serves different requirements with an example.

Example:

```
# import the datetime, date, time
```

```
>>> from datetime import datetime, date, time
```

```
# date
```

```
>>> d=date(2017,11,7)
```

```
# time
```

```
>>> t=time(10,10)
```

```
# combine both date and time
```

```
>>>datetime.combine(d,t)
```

```
datetime.datetime(2017, 11, 7, 10, 10)
```

```
# current date and time
```

```
>>>datetime.now()
```

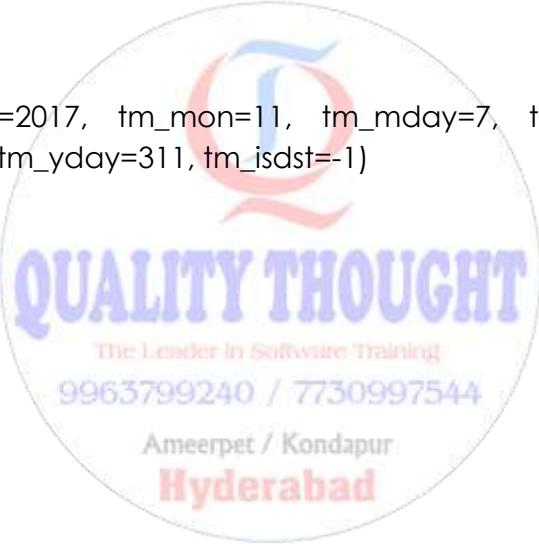
```
datetime.datetime(2017, 11, 7, 17, 21, 24, 338804)
```

```
>>>datetime.utcnow()  
datetime.datetime(2017, 11, 7, 11, 51, 37, 256627)
```

```
>>>dt=datetime.now()  
>>>dt  
datetime.datetime(2017, 11, 7, 17, 22, 58, 626832)
```

displaying the time tuple

```
>>>tt=dt.timetuple()  
>>>tt  
time.struct_time(tm_year=2017, tm_mon=11, tm_mday=7, tm_hour=17, tm_min=22,  
tm_sec=58, tm_wday=1, tm_yday=311, tm_isdst=-1)  
>>> for i in tt:  
...     print(i)  
...  
2017 # year  
11 # month  
7 # day  
17 # hour  
22 # minute  
58 # second  
1 # weekday ( 0 = Monday)  
311 # 311 th day in the year 2017  
-1
```



string format of date and time

```
>>>dt.strftime("%A, %d. %B %Y %I:%M%p")
```

```
'Tuesday, 07. November 2017 05:22PM'
```

```
>>> 'The {1} is {0:%d}, the {2} is {0:%B}, the {3} is {0:%I:%M%p}'.format(dt, "day", "month", "time")
```

```
'The day is 07, the month is November, the time is 05:22PM.'
```

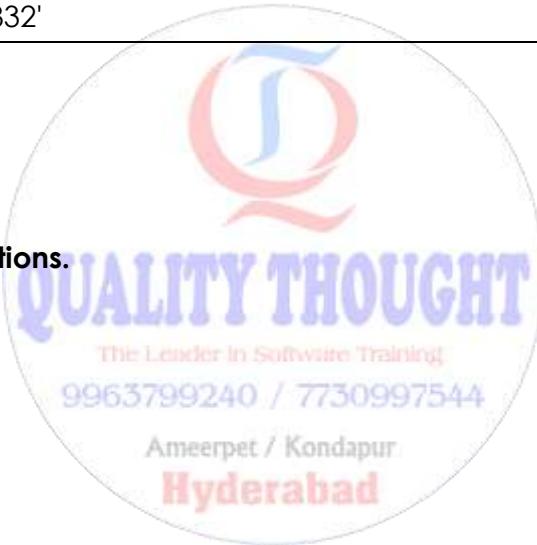
```
>>>dt.isoformat()
```

```
'2017-11-07T17:22:58.626832'
```

Python – Functions

There are 2 types of functions.

1. User-defined functions
2. Pre-defined functions



User-Defined functions:

In Python, User-defined function is a block of code which can be reusable. Once they are defined or written, that can be used multiple times and in other applications too.

Generally, the syntax of user-defined functions is represented by keyword def. Below is the syntax for defining the function without arguments.

Syntax:

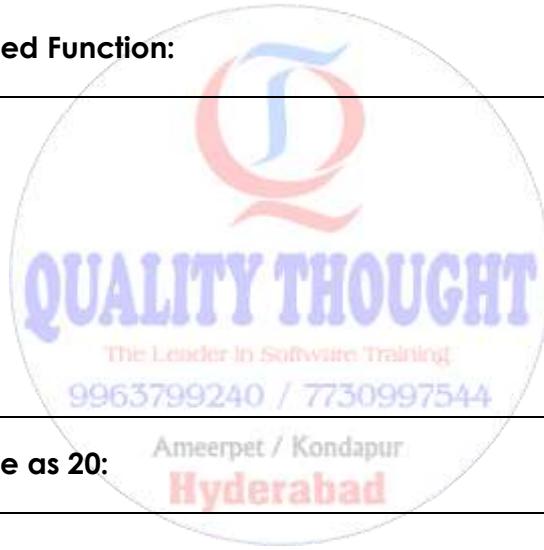
```
def function_name( args ):  
    statement 1  
    statement 2  
    return
```

Let us create a Function for returning a value of x after processing a loop.

```
>>> def prim(n):  
...     for x in range(2,n):  
...         for i in range(2,x):  
...             if x%i==0:  
...                 break  
...         else:  
...             print(x)
```

Let us call the User-defined Function:

```
>>>prim(10)  
2  
3  
5  
7
```



Using the argument value as 20:

```
>>>prim(20)  
2  
3  
5  
7  
11  
13  
17  
19
```

Using the argument value as 50:

```
>>>prim(50)  
2  
3  
5  
7  
11  
13  
17  
19  
23  
29  
31  
37  
41  
43  
47
```



Pre-defined Functions:

Pre-defined functions are already existing functions which cannot be changed. But still we can make our own custom functions using those pre-defined functions.

abs()
all()
any()
ascii()
bin()
bool()

bytearray()

bytes()

callable()

chr()

classmethod()

compile()

complex()

delattr()

dict()

dir()

divmod()

enumerate()

eval()

exec()

filter()

float()

format()

frozenset()

getattr()

globals()

hasattr()

hash()

help()

hex()

id()



input()

int()

isinstance()

issubclass()

iter()

len()

list()

locals()

map()

max()

memoryview()

min()

next()

object()

oct()

open()

ord()

pow()

print()

property()

range()

repr()

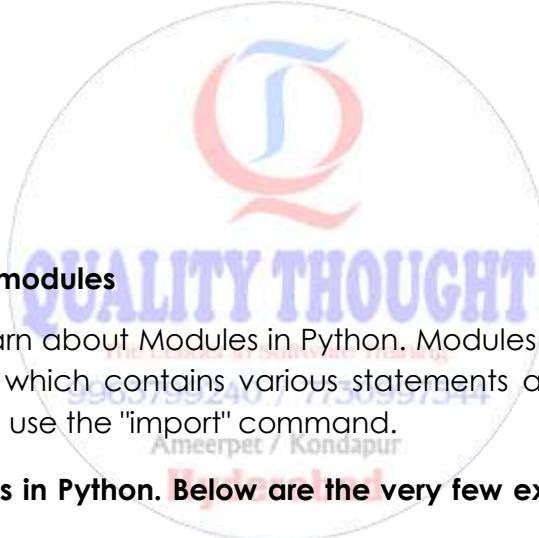
reversed()

round()

set()



setattr()
slice()
sorted()
staticmethod()
str()
sum()
super()
tuple()
type()
vars()
zip()
__import__()



Python - Packages and modules

In this tutorial, we will learn about Modules in Python. Modules in Python are none other than .py extension files which contains various statements and functions. In order to import a module, we will use the "import" command.

There are many modules in Python. Below are the very few examples just for giving an idea for you.

1. math - this is a mathematical module
2. statistics - This is for Statistical functions
3. pathlib - this is for filesystem paths
4. urllib - this is for handling urls
5. zlib - this is for data compression
6. csv - this is for file reading and writing

The above all are few pre-existing modules which are in python.

We can also write our own modules. Let us see how to create a module which helps us to process the prime numbers under any given value.

First create a file called "prime.py" and write the below code into the file.

```
def prim(n):  
    for x in range(2,n):  
        for i in range(2,x):  
            if x%i==0:  
                break  
        else:  
            print(x)
```

Now connect to python3 and import the module called "prime" using the keyword import. Then, call the function by passing the integer value as an argument to list the prime numbers for the given value.

```
>>> import prime  
>>>prime.prim(10)  
2  
3  
5  
7
```

Calling the function with value 20:

```
>>>prime.prim(20)  
2  
3
```

```
5
7
11
13
17
19
>>>
```

`dir(module_name)` will list us all types of variables, modules, functions used for the given module.

```
>>>dir(prime)
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__',
 '__spec__', 'prime']
```

Packages:

Packages are the namespaces within which consists many modules and packages. Every package is none other than a directory that consists a file called "`__init__.py`". This file describes the directories as packages.

For example, we have created the `prime.py` module in the above example. Now let us create the package for the same.

To create a package, we will to follow the below steps.

1. first have the module ready, that is `prime.py`
2. Create a directory called "`primenum`" and keep the above module in that directory.
3. Create a file called `__init__.py`

Let us try accessing the same.

Now we can import the prime module in 2 ways as below.

```
>>> import primenum.prime  
>>> from primenum import prime
```

Python – Reading a File

Reading a File in Python:

In this tutorial, we will learn about how to read the date from the file in python. In Python, we will use the method called "open" to open the file and "read" method to read the contents of the file.

Open Method take 2 arguments.

1. File name - refers to the name of the file.
2. Mode - refers to the mode of opening the file which may be either write mode or read mode.

Modes:

r - opens for reading the file

w - opens for writing the file

r+ - opens for both reading and writing

Read Methods:

read() - This method reads the entire data in the file. if you pass the argument as read(1), it will read the first character and return the same.

readline() - This method reads the first line in the file.

Let us work on an example:

First creating the data in a file called "techsal.csv". Below is the data how it looks.

designers, 100, salary, 10000

programmers, 1000, salary, 15000

Dataadmins, 10, salary, 12000

Now let us import the module called "csv", open the file and read the data:

```
>>>import csv  
>>> file=open("techsal.csv","r")  
>>>print(file.read())
```

Below is the output:

```
designers, 100, salary, 10000  
programmers, 1000, salary, 15000  
Dataadmins, 10, salary, 12000
```

Let us read the first character of the data by passing the numeric argument:

```
>>> file=open("techsal.csv","r")  
>>>print(file.read(1))  
d
```

Let us read the first line of the file by using the readline() method:

```
>>> file=open("techsal.csv","r")  
>>>print(file.readline())  
designers, 100, salary, 10000
```

Below is another way of writing the code to read the data:

```
>>> with open('techsal.csv') as data:  
...     out=data.read()  
...  
>>> print(out)  
designers, 100, salary, 10000  
programmers, 1000, salary, 15000  
Dataadmins, 10, salary, 12000
```

Once we complete the work, we need to close the file. Otherwise, it is just waste of memory. So, below is the way to close the file. Once you close the file, you cannot read the data. To read it again, you need to open it again.

```
>>>data.closed  
True  
>>>
```

Python – Writing into File

In this tutorial, we will learn how to write into a file in Python. Writing the data into a file can be done by using the "write()" method. The write() method takes string as an argument. Let us see an example of writing the data into the file called "techsal.csv".

First creating the data in a file called "techsal.csv". Below is the data how it looks.

designers, 100, salary, 10000

programmers, 1000, salary, 15000

Dataadmins, 10, salary, 12000

Let us write a line into the same file.

```
>>> data=open('techsal.csv','r+')
>>>data.write('\nthis is the new line entry\n')
28
```

Let us read the data from the file.

```
>>> with open('techsal.csv') as data:
...     out=data.read()
...     print(out)
...
```

Below is the output:

```
this is the new line entry  
0  
programmers, 1000, salary, 15000  
Dataadmins, 10, salary, 12000
```

Finally closing the file:

```
>>>data.close()
```

Python - Class & Objects

In this tutorial, we will understand about the Classes and Objects in Python. A class is depicted in different syntax as how the functions are defined. Below is the syntax of the Class.

Syntax:

```
class ClassName:  
<statement-1>  
.  
.  
.  
<statement-N>
```

The statements inside the class are function definitions and also contain other required statements. When a class is created, that creates a local namespace where all data variables and functions are defined.

Below is an example of a class.

```
>>> class MyFirstClass:  
...     """ This is an example class """  
...     data=127  
...     def f(self):
```

```
...     return 'hello !! you just accessed me'  
...
```

Accessing the variable data:

```
>>>print(MyFirstClass.data)
```

Output:

```
127
```

Accessing the function f:

```
>>>print(MyFirstClass.f)
```

Output:

```
<function MyFirstClass.f at 0x7f79c1de5158>
```

Accessing the doc String:

```
>>>print(MyFirstClass.__doc__)
```

Output:

```
This is an example class
```

Object:

Now let us see how to create an object. Creation of an object is an instance of the class. Below is how we creating an Object of the class MyFirstClass.

```
>>> x = MyFirstClass()
```

In the above line of code, we have created an Object for the class "MyFirstClass" and its name is "x".

Just trying to access the object name and it gives you information about object.

```
>>> x  
<__main__.MyFirstClass object at 0x7f79c1d62fd0>  
>>>print(x.f)  
<bound method MyFirstClass.f of <__main__.MyFirstClass object at 0x7f79c1d62fd0>>
```

Below is how you can access the attributes like data variables and functions inside the class using the Object name, which return some value.

```
>>>x.f()  
'hello !! you just accessed me'  
>>>x.data  
127
```

Python – Exceptions

In this tutorial, we will learn about the handling exceptions in Python. It is quite common that for any program written in any programming language may hit the error during the execution due to any reasons.

Reasons may be the syntactical error, or conditional or operational errors caused due to filesystem or due to the lack required resources to execute the program.

So, we need to handle those kind of exceptions or errors by using different clauses while we do programming.

In Python, we can handle an exception by using the raise exception statement or using the try, except clauses.

Syntax:

```
try:  
    raise statement  
except x:  
    statement
```

Exceptions are 2 types:

1. Pre-defined exceptions:

These are the Exceptions which are existing within the python programming language as Built-in Exceptions. Some of them are arithmetic errors like ZeroDivisionError, FloatingPointError, EOF errors... etc.

2. User-defined exceptions:

These exceptions are created by programmer which are derived from Exception class.

Example of Handling an Exception:

```
>>> while True:  
...     try:  
...         x = int(input("Please enter a number: "))  
...         print("The number entered is :", x)  
...         break  
...     except ValueError:  
...         print("Sorry !! the given number is not valid number. Try again...")
```

Output:

```
Please enter a number: abc  
Sorry !! the given number is not valid number. Try again...  
Please enter a number: apple  
Sorry !! the given number is not valid number. Try again...  
Please enter a number: a123  
Sorry !! the given number is not valid number. Try again...  
Please enter a number: 123  
The number entered is : 123
```

Example by using the Predefined Exception

```
>>> try:  
...     print(100/0)  
... except ZeroDivisionError as error:  
...     print(" we Handled predefined error:", error)  
...
```

Output:

we Handled predefined error: division by zero

Python - Regular Exp

Regular Expressions can be used for searching a word or a character or digits from the given data and several patterns. These can be called as RREs, regex patterns. We just need to import the module "re" to work with regular expressions.

Regular expression syntax:

\d - Matches any decimal digit [0-9].

\D - Matches any non-digit character [^0-9].

\s - Matches any whitespace character [\t\n\r\f\v].

\S - Matches any non-whitespace character [^ \t\n\r\f\v].

\w - Matches any alphanumeric character [a-zA-Z0-9_].

\W - Matches any non-alphanumeric character [^a-zA-Z0-9_].

Handling White Spaces:

\n = Used for new line

\s = Used for space

\t = Used for tab

\e = Used for escape

\f = Used for form feed

\r = Used for carriage return

Let us see the below example which uses the find all method to get the search result of salary from the data given as input to techdata variable.

Example:

```
import re
>>>techdata = """
... working on design technologies gives you $10000 salary.
... working on programming technologies gives you $15000 salary.
... working on Latest technologies give you $20000 salary.
...
...
>>>salary = re.findall(r'\d{1,10}',techdata)
```

Let us print the search result from data.

```
>>> print(salary)
['10000', '15000', '20000']
```

Python – Mathematics

In python, we have the module called "math" which provides the access to mathematical functions which are defined in C programming.

There are 2 types of modules for mathematical functions.

1. math - This is used for normal numbers
2. cmath - This is used for complex numbers

There are different types of mathematical functions available in math module.

1. Number Functions
2. Power and logarithmic Functions
3. Trigonometric Functions
4. Angular conversion
5. Hyperbolic Functions

6. Special Functions

7. Constants

Examples of Number Functions:

```
>>> from math import ceil, factorial, floor, gcd, fsum, trunc
```

ceiling the value

```
>>>ceil(10.3)
```

11

```
>>>ceil(9.9)
```

10

Factorial of given value

```
>>>factorial(3)
```

6

```
>>>factorial(10)
```

3628800



Floor of value given

```
>>>floor(10.3)
```

10

```
>>>floor(10.9)
```

10

calculates the GCD of given numbers

```
>>>gcd(5,10)
```

5

```
>>>gcd(3,7)
```

1

Floating point sum of of values

```
>>>fsum([5.4,5,1])
```

```
15.0
```

Truncating the values

```
>>>trunc(9.4)
```

```
9
```

```
>>>trunc(10.5)
```

```
10
```

```
>>>
```

Examples of Power and logarithmic Functions:

```
>>>math.exp(2)
```

```
7.38905609893065
```

```
>>>math.log(2,10)
```

```
0.30102999566398114
```

```
>>>math.log(2,4)
```

```
0.5
```

```
>>> math.log2(4)
```

```
2.0
```

```
>>> math.log10(2)
```

```
0.3010299956639812
```



```
>>>math.pow(2,3)
```

```
8.0
```

```
>>>math.sqrt(64)
```

```
8.0
```

Examples of Trigonometric Functions:

```
>>> from math import sin, cos, tan
```

```
# Values are in radians
```

```
>>>sin(30)
```

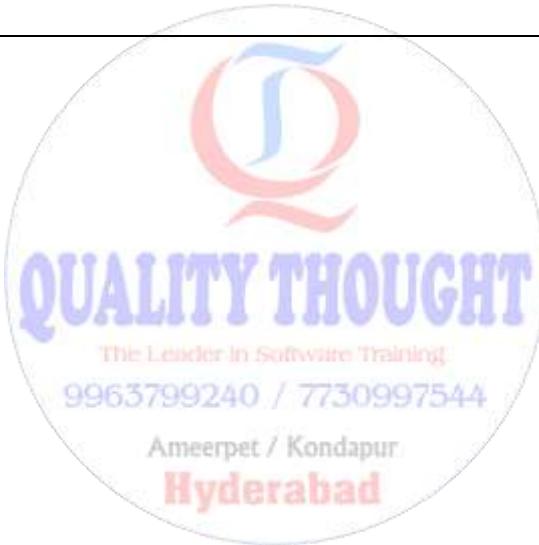
```
-0.9880316240928618
```

```
>>>cos(90)
```

```
-0.4480736161291701
```

```
>>>tan(0)
```

```
0.0
```

**Examples of Angular conversion:**

```
>>>from math import degrees,radians
```

```
>>>degrees(10)
```

```
572.9577951308232
```

```
>>>radians(572)
```

```
9.983283321407566
```

Python - Internet access

In this tutorial, we will learn about how to access the internet using the python. In python, we will have a module called "urllib" that provides various Objects and functions to access the internet. We can perform many activities using this "urllib" module like accessing the webpage data of any website, sending an email... etc.

Let us try fetching the 100 bytes of code behind the google.com page. Below is the example.

Example:

```
>>> import urllib.request  
  
>>> f = urllib.request.urlopen("http://google.com/")  
  
>>> print(f.read(100).decode('utf-8'))
```

Output:

```
<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="en-IN"><head><meta cont
```

Let us try fetching the 500 bytes of code behind the google.com page.

```
>>> print(f.read(500).decode('utf-8'))
```

Output:

```
ent="text/html; charset=UTF-8" http-equiv="Content-Type"><meta content="/logos/doodles/2017/sitara-devi-s-97th-birthday-6469056130449408.5-l.png" itemprop="image"><meta content="Sitara Devi's 97th birthday" property="twitter:title"><meta content="Celebrating Sitara Devi's 97th birthday! #GoogleDoodle" property="twitter:description"><meta content="Celebrating Sitara Devi's 97th birthday! #GoogleDoodle" property="og:description"><meta content="summary_large_image" property="twitte
```

Python - Data Compression

In this tutorial, we will learn about the data compression in Python programming language. In python, the data can be archived, compressed using the modules like zlib, gzip, bz2, lzma, zipfile and tarfile. To use the respective module, you need to import the module first. Let us look at below example.

Example:

```
>>> import zlib

>>> s = b'you learn learnt learning the data daily'

>>>len(s)

41

>>>

>>> t = zlib.compress(s)

>>>len(t)

39

>>>

>>>zlib.decompress(t)

b'you learn learnt learning the data daily'

>>>

>>>zlib.crc32(s)

2172471860

>>>
```



Numpy

1. NumPy - Introduction
2. NumPy – Installation
3. NumPy – Import Module

4. NumPy – Data Types
5. NumPy – Arrays
6. Numpy – Array using arange
7. NumPy – Indexing & Slicing
8. NumPy – Advanced Indexing
9. NumPy – Broadcasting
10. NumPy – Iterating Over Array
11. NumPy – Array Manipulation
12. NumPy – String Functions
13. NumPy – Mathematical Functions
14. NumPy – Arithmetic Operations
15. NumPy – Statistical Functions
16. Numpy – Sort, Search & Counting Functions
17. NumPy – Matrix Library
18. NumPy – Linear Algebra
19. Numpy – Histogram using Matplotlib



NumPy – Introduction

Numpy is one of the libraries available for Python programming language. This library or module provides numerical and mathematical functions which are pre-compiled.

Numpy is designed to be used for multidimensional arrays and for scientific computing which are memory efficient.

Here we have 2 packages

1. Numpy – This provides basic calculations with multi-dimensional arrays and matrices of numeric data.
2. Scipy – This package provides functionality of Numpy with added algorithms like , regression, minimization, Fourier transforms, statistical operations, random simulation and applied mathematical techniques.

NumPy – Installation

In this tutorial, we will understand that how to do the installation of Numpy on both linux and windows platforms.

It is best to use the pre-built packages to install the Numpy. Otherwise, you can install the python distributions like Anaconda, python(x,y), Pyzo for installing all necessary packages which ever needed.

Installing Numpy on Linux Platform:

To install Numpy on Linux platform, we need have the Python already installed.

In most Linux platforms, Python would come by default. If not, you can use the Yum utility to install python and other packages which ever needed with below command on RedHat or cent OS.

```
$ yum install python-numpy
```

Note:

Use SUDO, if you are not the root user.

For Ubuntu &Debian systems Use the below command to get all the necessary packages installed.

```
sudo apt-get install python-numpy python-scipy python-matplotlib python-ipython python-notebook python-pandas python-sympy python-nose
```

Installing Numpy on Windows Platform:

To install Numpy on Windows platform, we need have the Python already installed.

If not please go through Python installation process from the Python tutorial.

Once we have the Python installed on the windows system, we need to make sure that the Path to Python was set in environmental variables.

There is package manager or installer name called pip. Just give the below command to install any package.

```
pip install numpy
```

Once after the installation, you can either connect to python in command by giving the Python command or you can start the "IDLE (Python GUI)" from start button which is GUI tool.

NumPy – Import Module

In this tutorial, we will learn how to import and use Numpy. You have to use the keyword "import" to import the numpy module.

Below is command to import the Numpy Module.

```
>>> import numpy
```

But, the better way of importing the Numpy is given below.

```
>>> import numpy as np
```

It is better to give an alias name and try using the same alias name for every call to numpy. Otherwise it will become write numpy.X file every time. By giving the alias name, it will use the np.X instead of using numpy.X.

There is another method of importing entire Numpy in a single call.

```
>>> from numpy import *
```

Anyhow this is not much preferred everytime.

NumPy – Data Types

Numpy provides the below datatypes more than what exactly python holds.

Data type	Description
bool_	Boolean (True or False) stored as a byte
int_	Default integer type (same as C long; normally either int64 or int32)
intc	Identical to C int (normally int32 or int64)
intp	Integer used for indexing (same as C ssize_t; normally either int32 or int64)
int8	Byte (-128 to 127)
int16	Integer (-32768 to 32767)
int32	Integer (-2147483648 to 2147483647)
int64	Integer (-9223372036854775808 to 9223372036854775807)
uint8	Unsigned integer (0 to 255)
uint16	Unsigned integer (0 to 65535)
uint32	Unsigned integer (0 to 4294967295)
uint64	Unsigned integer (0 to 18446744073709551615)
float_	Shorthand for float64.
float16	Half precision float: sign bit, 5 bits exponent, 10 bits mantissa
float32	Single precision float: sign bit, 8 bits exponent, 23 bits mantissa
float64	Double precision float: sign bit, 11 bits exponent, 52 bits mantissa

complex_	Shorthand for complex128.
complex64	Complex number, represented by two 32-bit floats (real and imaginary components)
complex128	Complex number, represented by two 64-bit floats (real and imaginary components)

Let us see some example below:

How identify the datatype ?

Below is the command. we will use the “dtype” method to identify the datatype

```
>>> import numpy as np
```

```
>>> x=np.array([1,2,3,4,5])
```

```
>>>x
```

```
array([1, 2, 3, 4, 5])
```

```
>>>x.dtype
```

```
dtype('int32')
```

What if you want to assign the integers as float datatype ?

Below is the command. Please observe that we have given the dtype as floating datatype.

```
>>> x=np.array([1,2,3,4,5], dtype=float)
```

```
>>>x.dtype
```

```
dtype('float64')
```

```
>>>x
```

```
array([ 1., 2., 3., 4., 5.])
```

Below example shows the floating datatype values.

```
>>> y=np.array([.1,.2,.3,.4,.5])
```

```
>>>y
```

```
array([ 0.1, 0.2, 0.3, 0.4, 0.5])
```

```
>>>y.dtype
```

```
dtype('float64')
```

Now let us see some more data types too.

This example will show the Boolean datatype.

```
>>>eq=np.array([True, False])
```

```
>>>eq
```

```
array([ True, False], dtype=bool)
```

```
>>>eq.dtype
```

```
dtype('bool')
```

This example will show the String datatype.

```
>>>str=np.array(["This", "is", "NumPy"])
```

```
>>>str
```

```
array(['This', 'is', 'NumPy'],
```

```
dtype='<U5')
```

```
>>>str.dtype
```

```
dtype('<U5')
```

This example will show the Complex datatype.

```
>>> j=2
```

```
>>> solve=np.array([2+3j, 5+j, 9+2*3j])
```

```
>>> solve
```

```
array([ 2.+3.j,  7.+0.j,  9.+6.j])
```

```
>>>solve.dtype
```

```
dtype('complex128')
```

```
>>>solve.real
```

```
array([ 2.,  7.,  9.])
```

NumPy – Arrays

In this tutorial, we will discuss about arrays in Numpy.

In NumPy, Array is an Object class little similar to lists in Python. But, here the array holds the elements which are same numeric data type like int or float.

This array object can process the large numeric data efficiently when compared to lists.

Below is the sample examples of creating an array.

Creating an One-Dimensional(1-D) Array:

```
>>> a=np.array([1,2,3,4],float)  
  
>>>a  
  
array([ 1., 2., 3., 4.])  
  
>>> a=np.array([1,2,3,4],int)  
  
>>>a  
  
array([1, 2, 3, 4])  
  
>>>type(a)  
<class 'numpy.ndarray'>  
# ndim will be used to find the type of dimension of that array.  
>>>a.ndim  
  
1  
  
# shape property will be used to find out the size of each array dimension  
  
>>>a.shape  
  
(4,)  
  
# len function will be used to get the length of first array axis in the dimension  
  
>>>len(a)  
  
4
```

Arrays are Multi-dimensional (mostly like Matrices in Mathematics) and let us see creating an Two-Dimensional(2-D) Array:

```
>>> b=np.array([[1,2,3,4],[5,6,7,8]],int)
```

```
>>>b
```

```
array([[1, 2, 3, 4],
```

```
       [5, 6, 7, 8]])
```

```
>>>type(b)
```

```
<class 'numpy.ndarray'>
```

```
>>>b.ndim
```

```
2
```

```
>>>b.shape
```

```
(2, 4)
```

```
>>>len(b)
```

```
2
```

Creating an Three-Dimensional(3-D) Array: Software Training

9963799240 / 7730997544

Ameerpet / Kondapur

Hyderabad

```
>>> c=np.array([[1,2,3,4],[5,6,7,8],[9,8,7,6]],int)
```

```
>>>type(c)
```

```
<class 'numpy.ndarray'>
```

```
>>>c
```

```
array([[1, 2, 3, 4],
```

```
       [5, 6, 7, 8],
```

```
       [9, 8, 7, 6]])
```

```
>>>c.ndim
```

```
2
```

```
>>>c.shape
```

(3, 4)

```
>>>len(c)
```

3

Accessing the values in above three-Dimensional Array:

```
>>>c[1,1]
```

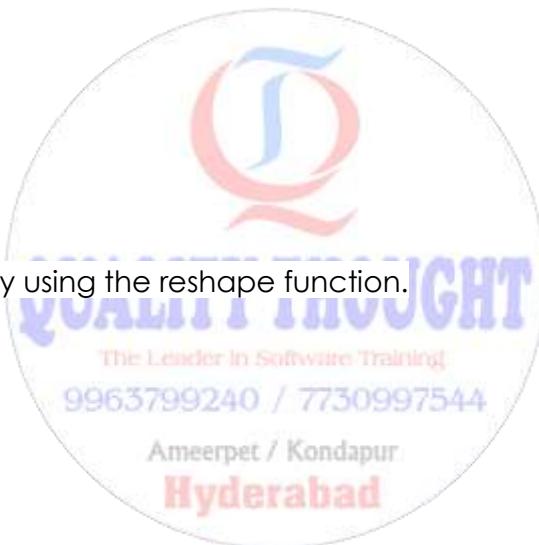
6

```
>>>c[2,3]
```

6

```
>>>c[2,0]
```

9



We can reshape an array using the reshape function.

```
>>>c.reshape(4,3)
```

```
array([[1, 2, 3],
```

```
[4, 5, 6],
```

```
[7, 8, 9],
```

```
[8, 7, 6]])
```

Numpy – Array using arange

The arange function which almost like a Range function in Python. The arange function will return an array as a result.

let us see an example.

```
>>>a=np.arange(5)
```

```
>>>a
```

```
array([0, 1, 2, 3, 4])
```

```
>>>a[0]
```

```
0
```

```
>>>a[1]
```

```
1
```

```
>>>a[4]
```

```
4
```

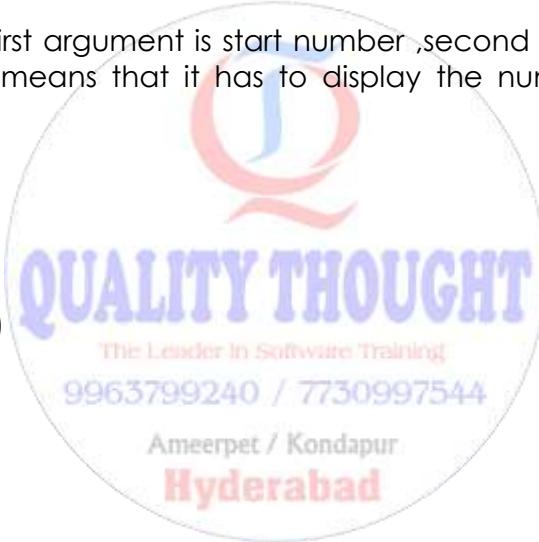
In the below example, first argument is start number ,second is ending number, third is nth position number. It means that it has to display the numbers for every 5th step starting from one to 20.

Example:

```
>>> b=np.arange(1,20,5)
```

```
>>>b
```

```
array([ 1, 6, 11, 16])
```



If you want to divide it by number of points, linspace function can be used. This will reach the end number by the number of points you give as the last argument.

Example:

First argument - 0

Second argument - 1

Third Argument - 5

```
>>> b=np.linspace(0,1,5)
```

```
>>>b  
  
array([ 0. , 0.25, 0.5 , 0.75, 1. ])  
  
>>>  
  
>>>b=np.linspace(0,1,6)  
  
>>>b  
  
array([ 0. , 0.2, 0.4, 0.6, 0.8, 1. ])  
  
>>>b=np.linspace(0,1,7)  
  
>>>b  
  
array([ 0. , 0.16666667, 0.33333333, 0.5 , 0.66666667, 0.83333333, 1. ])
```

Creating Arrays using other functions like ones, zeros, eye.
Below example is using the ones function.

```
>>>a=np.ones([3,3])  
  
>>>a  
  
array([[ 1.,  1.,  1.],  
       [ 1.,  1.,  1.],  
       [ 1.,  1.,  1.]])
```

Below example is using the zeros function.

```
>>>a=np.zeros([2,3])  
  
>>>a  
  
array([[ 0.,  0.,  0.],  
       [ 0.,  0.,  0.]])
```

Accessing the value of a[1,1]

```
>>>a[1,1]  
  
0.0
```

```
>>>np.ones(5)  
array([ 1., 1., 1., 1., 1.])  
  
>>>np.zeros(5)  
array([ 0., 0., 0., 0., 0.])
```

Below example is using the eye function.

```
>>> a=np.eye(3)  
  
>>>a  
  
array([[ 1., 0., 0.],  
       [ 0., 1., 0.],  
       [ 0., 0., 1.]])
```

NumPy – Indexing & Slicing

In this tutorial, we will learn about indexing and slicing of data in NumPy.

In NumPy, very efficient and optimized indexing can be done by using various functions which are provided in the package.

Let us understand indexing with Numpy by creating an array of one-dimensional and accessing all the values.

Example:

```
>>> a=np.arange(5)  
  
>>>a  
  
array([0, 1, 2, 3, 4])  
  
>>>a[0]  
  
0  
  
>>>a[1]
```

1

>>>a[2]

2

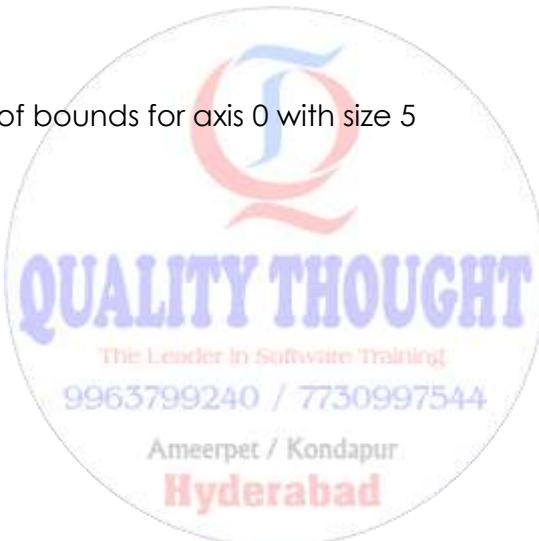
5th position does not exists. Hence the error below.

>>>a[5]

Traceback (most recent call last):

File "<pyshell#112>", line 1, in <module>

a[5]



IndexError: index 5 is out of bounds for axis 0 with size 5

>>>a[-1]

4

>>>a[-2]

3

>>>a[-3]

2

>>>a[-4]

1

>>>a[-5]

0

>>>

Now let us try with Multi-Dimensional Array.

We are creating an array with diag function to create a 3 by 3 matrix form and the values given will be placed in the diagonal section of the matrix.

In the Multi-dimensional Array,

The dimension corresponding to rows can be interpreted by using a[0] (First row of all elements)

The column axis

```
>>> a=np.diag(np.arange(3))
```

```
>>>a
```

```
array([[0, 0, 0],
```

```
[0, 1, 0],
```

```
[0, 0, 2]])
```

```
>>>a[0]
```

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

```
>>>a[1]
```

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

```
>>>a[2]
```

```
array([0, 0, 2])
```

To access the column axis, we need to mention the specified index number to access the value.

```
>>>a[1,1]
```

```
1
```

```
>>>a[1,2]
```

```
0
```

below is the error just because we tried to access the 3rd position value in the column which is not existing.

```
>>>a[1,3]
```

Traceback (most recent call last):

```
File "<pyshell#122>", line 1, in <module>
```

```
a[1,3]
```

```
IndexError: index 3 is out of bounds for axis 1 with size 3
```



```
>>>a[1,0]
```

```
0
```

```
>>>a[0,0]
```

```
0
```

```
>>>
```

Accessing the column values from the matrix.we can use ellipsis(...) to get the column or row values in particular. If we place the ellipsis in the row position, it will get you the all the values of particular column.

```
>>>a[...,1]
```

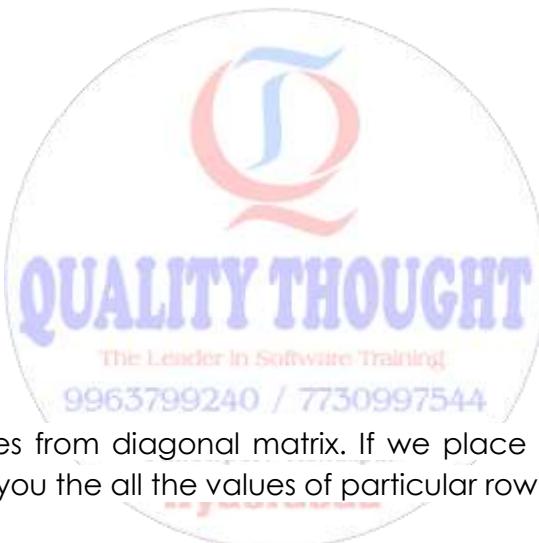
```
array([0, 1, 0])
```

```
>>>a[...,0]
```

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

```
>>>a[...,2]
```

```
array([0, 0, 2])
```



Accessing the row values from diagonal matrix. If we place the ellipsis in the column value position, it will get you the all the values of particular row mentioned.

```
>>>a[1]
```

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

```
>>>a[1,...]
```

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

```
>>>
```

slicing:

```
>>> a=np.arange(20)
```

```
>>>a
```

```
array([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,  
17, 18, 19])  
  
>>>a[5:20:5]  
  
array([ 5, 10, 15])  
  
>>>a[:4]  
  
array([0, 1, 2, 3])  
  
>>>a[:10]  
  
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])  
  
>>>a[1:5]  
  
array([1, 2, 3, 4])  
  
>>>a[5:]  
  
array([ 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19])  
  
>>>a[::-5]  
  
array([ 0, 5, 10, 15])  
  
>>>a[0]  
  
0  
  
>>>a[1:5]  
  
array([1, 2, 3, 4])
```



NumPy – Advanced Indexing

Let us say `x[obj]` is an array which holds `obj` as the selection object. When the object is a non-tuple sequence object or a tuple with atleast one sequence object which is ndarray of type integer or Boolean.

Note:

Advanced indexing always returns a copy of data but not view as like basic slicing.

Here, there are 2 types of advanced indexing.

1. Integer Array Indexing
2. Boolean Indexing

Integer Array Indexing:

Syntax:

```
result[i_1, ..., i_M] == x[ind_1[i_1, ..., i_M], ind_2[i_1, ..., i_M], ..., ind_N[i_1, ..., i_M]]
```

Example:

Below, we have created an array with integers.

```
>>> x=np.array([[1,2],[3,4],[5,6]])
```

Output:

```
>>>x  
array([[1, 2], [3, 4], [5, 6]])
```

Let us try to select specific elements like [0,1,2] which is a row index and column index [0,1,0] each element for the corresponding row.

```
>>>x[[0,1,2],[0,1,0]]
```

```
array([1, 4, 5])
```

Let us select with 0 which gives you the first row.

```
>>>x[0]
```

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

Let us select the 0 as row index and 1 as column index which gives as array value of 2

```
>>>x[[0],[1]]
```

```
array([2])
```

In the same way, if you select for 0,2 will give us an error. As, there is no value for index of 2.

```
>>>x[[0],[2]]
```

Traceback (most recent call last):

```
File "<pyshell#8>", line 1, in <module>
```

```
x[[0],[2]]
```

IndexError: index 2 is out of bounds for axis 1 with size 2

You can do the add operation which returns the value of particular index after performing the addition.

```
>>>x
```

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

```
[3, 4],
```

```
[5, 6]])
```

```
>>>x[[0],[1]]+1
```

```
array([3])
```

Below operation will change the values in the array and returns the new copy of an array.

```
>>>x
```

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

```
[3, 4],
```

```
[5, 6]])
```

```
>>>x[[0],[1]]+=1
```

```
>>>x  
  
array([[1, 3],  
       [3, 4],  
       [5, 6]])
```

Boolean Indexing:

Boolean Indexing will be used when the result is going to be the outcome of boolean operations.

Example:

```
>>>x=np.array([[0,1,2],  
              [3,4,5],  
              [6,7,8],  
              [9,10,11]])  
  
>>>x  
  
array([[ 0,  1,  2],  
       [ 3,  4,  5],  
       [ 6,  7,  8],  
       [ 9, 10, 11]])
```

returns the values which are 0.

```
>>>x[x==0]
```

```
array([0])
```

returns the values which are even numbers.

```
>>>x[x%2==0]
```



```
array([ 0,  2,  4,  6,  8, 10])
```

```
>>>
```

NumPy – Broadcasting

Operations on Numpy arrays are generally happened on element wise. It means the arrays of same size works better for operations.

But it is also possible to do the operations on those arrays which are different in size.

How to do this?

Nothing to do anything by us. Numpy can transform the arrays of different sizes into the same sizes. That kind of formulation or conversion is known as broadcasting in Numpy.

Let us observe the below example:

Creating an array of size 1 column * 4 row

```
>>> a=np.array([[0],[10],[20],[30]])
```

```
>>>a
```

```
array([[ 0],  
       [10],  
       [20],  
       [30]])
```

Creating an array of size 3 column * 1 row

```
>>> b=np.array([0,1,2])
```

```
>>>b
```

```
array([0, 1, 2])
```

Adding the both of them will give us the resultant Array after broadcasting

```
>>>a+b
```

```
array([[ 0,  1,  2],  
       [10, 11, 12],  
       [20, 21, 22],  
       [30, 31, 32]])
```

Substracting both of them will give us below result.

```
>>>a-b  
  
array([[ 0, -1, -2],  
       [10,  9,  8],  
       [20, 19, 18],  
       [30, 29, 28]])
```

Multiplying the both of them will give us below result.

```
>>>a*b  
  
array([[ 0,  0,  0],  
       [ 0, 10, 20],  
       [ 0, 20, 40],  
       [ 0, 30, 60]])
```



NumPy – Iterating Over Array

In this tutorial, we will learn about array iteration in Numpy.

In python, we have used iteration through lists. In the same way, we can iterate over arrays in Numpy.

Let us see an example by giving an array of integer elements.

Example:

```
>>> a=np.array([1,2,3,4,5,6,7,8,9,10],int)
```

```
# print the array a.  
  
>>>a  
  
array([ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
```

```
# iterate over the array a of integer numbers.  
  
>>>for i in a:  
  
    print (i)  
  
# Below is the Output:
```

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10
```



How this works with Multidimensional array ?

Here the iteration will go over the first axis so that each loop returns you the subset of the array.

Let us try with 2 – Dimensional Array.

Example:

```
# 2-D Array  
  
>>> a=np.array([[1,2],[3,4],[5,6],[7,8],[9,10]],int)
```

```
# Print the 2-D Array
```

```
>>>a
```

```
array([[ 1,  2],  
       [ 3,  4],  
       [ 5,  6],  
       [ 7,  8],  
       [ 9, 10]])
```

```
# Iterating Through 2-D Array
```

```
>>>for i in a:
```

```
    print (i)  
  
[1 2]  
[3 4]  
[5 6]  
[7 8]  
[ 9 10]
```



```
# Doing the calculation along with the array iteration.
```

```
>>>for (i,j) in a:
```

```
    print (i+j)  
  
3  
7  
11  
15
```

Let us work with the 3-Dimensional Array.

```
# 3-Dimensional Array  
>>> a=np.array([[1,2,3],[4,5,6],[7,8,9]],int)
```

```
# Print 3-Dimensional Array  
>>>a  
array([[1, 2, 3],  
       [4, 5, 6],  
       [7, 8, 9]])
```

```
# Iterating Through 3-Dimensional Array  
>>>for i in a:  
    print (i)  
[1 2 3]  
[4 5 6]  
[7 8 9]
```

Doing the calculation along with the array iteration.

```
>>>for (x,y,z) in a:  
    print (x+y+z)  
6  
15  
24  
>>>for (x,y,z) in a:
```



```
print (x*y*z)
```

```
6
```

```
120
```

```
504
```

NumPy – Array Manipulation

We can change the array shape using the Array manipulation routines like reshape and ravel.

reshape: This will give us the new shape for an array without changing its data

ravel: This will return the contiguous flattened array.

We can transpose an array using the ndarray.T Operation which will be same if self.ndim is less than 2

Let us work on an example.

Example:

```
# creating array of 2 * 3 dimension
```

```
>>> a=np.array([[1,2,3],[4,5,6]])
```

```
>>>a
```

```
array([[1, 2, 3],
```

```
[4, 5, 6]])
```

```
# using ravel to flatten the array
```

```
>>>a.ravel()
```

```
array([1, 2, 3, 4, 5, 6])
```

```
# transpose the array from 2 * 3 dimension to 3 * 2 dimension
```

```
>>>a.T
```

```
array([[1, 4],
```

[2, 5],

[3, 6]])

ravel the transposed array

```
>>>a.T.ravel()
```

```
array([1, 4, 2, 5, 3, 6])
```

Using the reshape option to revert from ravel.

```
>>>a.T.ravel().reshape((2,3))
```

```
array([[1, 4, 2],  
[5, 3, 6]])
```

Using the shape and reshape

```
>>> a=np.arange(4*3*2)
```

```
>>>a
```

```
array([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,  
17, 18, 19, 20, 21, 22, 23])
```

```
>>>a.reshape(4,3,2)
```

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

```
 [ 2, 3],
```

```
 [ 4, 5]],
```

```
 [[ 6, 7],
```

```
 [ 8, 9],
```

```
 [10, 11]],
```



```
[[12, 13],  
 [14, 15],  
 [16, 17]],  
 [[18, 19],  
 [20, 21],  
 [22, 23]]])  
  
>>>a.shape  
  
(24,)  
  
>>>b=a.reshape(4,3,2)  
  
>>>b  
  
array([[[ 0, 1],  
 [ 2, 3],  
 [ 4, 5]],  
 [[ 6, 7],  
 [ 8, 9],  
 [10, 11]],  
 [[12, 13],  
 [14, 15],  
 [16, 17]],  
 [[18, 19],  
 [20, 21],  
 [22, 23]]])  
  
>>>b.shape  
  
(4, 3, 2)
```



NumPy – String Functions

In Numpy, we can handle the string operations with provided functions. some of which we can discuss here.

add: This will return element-wise string concatenation for two arrays of str.

Example:

```
>>> x=np.array(("iam a numpy"))
>>> y=np.array(("program"))
>>>np.char.add(x,y)
array('iam a numpyprogram',dtype='<U18')
```

multiply: This will return element-wise string multiple concatenation.

Example:

```
>>> import numpy as np
>>> x=np.char.multiply("numpy",5)
>>>x
array('numpynumpynumpynumpynumpy', dtype='<U25')
>>>print(x)
```

Numpynumpynumpynumpynumpy

capitalize: This will return a copy of string with first character of each element capitalized.

Example:

```
>>> import numpy as np
>>> x=np.char.capitalize("numpy")
>>>x
array('Numpy', dtype='<U5')
```

```
>>>print(x)
```

Numpy

split: This will return a list of words in the string.

Example:

```
>>> import numpy as np  
>>> x=np.char.split("iam a numpy program")  
>>>print(x)  
['iam', 'a', 'numpy', 'program']
```



lower: This will return an array with elements converted to lowercase.

Example:

```
>>> import numpy as np  
>>> x=np.char.lower("NUMPY")  
>>>print(x)
```

Numpy

upper: This will return an array with elements converted to uppercase.

Example:

```
>>> import numpy as np  
>>> x=np.char.upper("numpy")  
>>>print(x)
```

NUMPY

equal: This will return a element-wise comparision.

Example:

```
>>> x=np.char.equal("iam","numpy")
```

```
>>>print(x)
```

False

not_equal: This will return a element-wise comparision.

Example:

```
>>> x=np.char.not_equal("iam","numpy")
```

```
>>>print(x)
```

True

count: This will return an array with the number of non-overlapping occurrences of substring in the range.

Example:

```
>>> x=np.array(['bet','abet','alphabet'])
```

```
>>>x
```

```
array(['bet', 'abet', 'alphabet'], dtype='<U8')
```

```
>>>np.char.count(x,'bet')
```

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

```
>>>np.char.count(x,'abet')
```

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

```
>>>np.char.count(x,'alphabet')
```

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

isnumeric: This will return true if there is only numeric characters in the element.

Example:

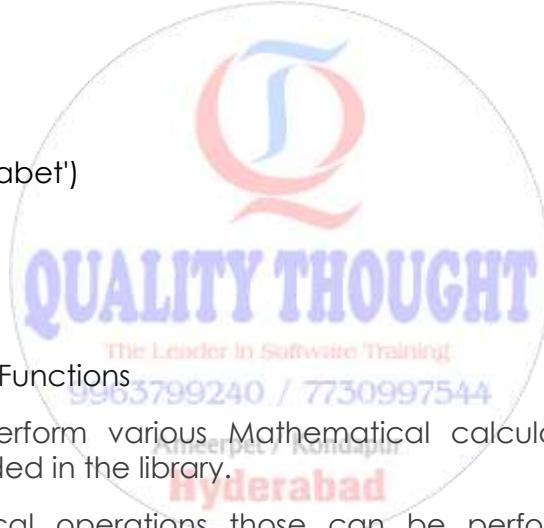
```
>>>np.char.isnumeric('bet')
```

```
array(False, dtype=bool)
```

`rfind`: This will return the highest index in the string where substring is found.

Example:

```
>>> x=np.array(['bet','abet','alphabet'])  
>>>x  
array(['bet', 'abet', 'alphabet'], dtype='<U8')  
>>>np.char.rfindex(x,'abet')  
array([-1, 0, 4])  
>>>np.char.rfindex(x,'bet')  
array([0, 1, 5])  
>>>np.char.rfindex(x,'alphabet')  
array([-1, -1, 0])
```



NumPy – Mathematical Functions

In Numpy, we can perform various Mathematical calculations using the various functions that are provided in the library.

Advanced Mathematical operations those can be performed are Trigonometric functions, Hyperbolic functions, Rounding, Sums, Products, differences, exponents, logarithms etc.

Here, let us perform some of those Mathematical functions.

Trigonometric Operations:

`sin`: This will return the element-wise Trigonometric sine calculation.

`cos`: This will return the element-wise Trigonometric cosine calculation.

`tan`: This will return the element-wise Trigonometric tangent calculation.

Example:

```
>>> import numpy as np
```

```
>>>np.sin(np.pi/2)
```

```
1.0
```

```
>>>np.cos(np.pi/2)
```

```
6.123233995736766e-17
```

```
>>>np.tan(np.pi/2)
```

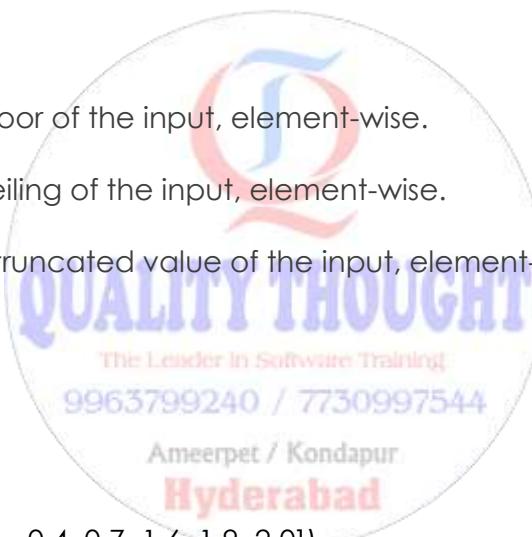
```
16331239353195370.0
```

Rounding Operations:

floor: This will return the floor of the input, element-wise.

ceil: This will return the ceiling of the input, element-wise.

trunc: This will return the truncated value of the input, element-wise.



Example:

```
>>> import numpy as np
```

```
>>> x=np.array([-1.4, -1.1, -0.4, 0.7, 1.6, 1.9, 2.0])
```

```
>>>print(x)
```

```
[-1.4 -1.1 -0.4 0.7 1.6 1.9 2.]
```

```
>>>print(np.floor(x))
```

```
[-2. -2. -1. 0. 1. 1. 2.]
```

```
>>>print(np.ceil(x))
```

```
[-1. -1. -0. 1. 2. 2. 2.]
```

```
>>>print(np.trunc(x))
```

```
[-1. -1. -0. 0. 1. 1. 2.]
```

Sums and Differences:

sum: This will return the sum of array elements for the given axis.

diff: This will return the nth-discrete difference along given axis.

Example:

```
>>> import numpy as np
```

```
>>> x=np.sum([[1,2],[3,4]])
```

```
>>>print(x)
```

```
10
```

```
>>> y=np.sum([[1,2],[3,4]], axis=0)
```

```
>>> print(y)
```

```
[4 6]
```

```
>>> z=np.sum([[1,2],[3,4]], axis=1)
```

```
>>>print(z)
```

```
[3 7]
```

```
>>> x=np.diff([[1,2],[3,4]])
```

```
>>>print(x)
```

```
[[1]
```

```
[1]]
```

```
>>> y=np.diff([[1,2],[3,4]], axis=0)
```

```
>>> print(y)
```

```
[[2 2]]
```

```
>>> z=np.diff([[1,2],[3,4]], axis=1)
```

```
>>>print(z)
```

```
[[1]]
```



[1]]

Logarithmic Operations:

log: This will return the Natural logarithm, element-wise.

log2: This will return the Base-2 logarithm of x.

Example:

```
>>> import numpy as np
```

```
>>> x=np.log([1])
```

```
>>>print(x)
```

```
[ 0.]
```

```
>>> y=np.log2([2,4,8])
```

```
>>> print(y)
```

```
[ 1. 2. 3.]
```



NumPy – Arithmetic Operations

In Numpy, we can perform various Arithmetic calculations using the various functions like add, reciprocal, negative, multiply, divide, power, subtract, remainder etc.

Add: This will return the addition of arguments, element-wise.

Multiply: This will return the multiplication of arguments, element-wise.

Divide: This will return the division of arguments, element-wise.

Power: This will return the result of first array elements raised to powers from second array, element-wise.

Remainder: This will return the addition of arguments, element-wise.

Example:

```
>>> import numpy as np  
  
>>> x=np.add(10,20)  
  
>>>print(x)  
  
30  
  
>>> x=np.multiply(10,20)  
  
>>>print(x)  
  
200  
  
>>> x=np.divide(10,20)  
  
>>>print(x)  
  
0.5  
  
>>> x=np.power(10,2)  
  
>>>print(x)  
  
100  
  
>>> x=np.remainder(10,2)  
  
>>>print(x)  
  
0  
  
>>> x=np.remainder(9,2)  
  
>>>print(x)  
  
1
```



NumPy – Statistical Functions

In Numpy, we can perform various statistical calculations using the various functions that are provided in the library like Order statistics, Averages and variances, correlating, Histograms.

Let us work on some of those statistical functions.

Order statistics:

amin: This will return the minimum of an array.

amax: This will return the maximum of an array.

Example:

```
>>> import numpy as np
```

```
>>> x=np.arange(4).reshape((2,2))
```

```
>>>x
```

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

```
[2, 3]])
```

```
>>>np.amin(x)
```

```
0
```

```
>>>np.amin(x, axis=0)
```

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

```
>>>np.amax(x)
```

```
3
```

```
>>>np.amax(x, axis=0)
```

```
array([2, 3])
```

```
>>>np.amax(x, axis=1)
```

```
array([1, 3])
```

Averages and variances:

Median: This will return the median along the specified axis.

Average: This will return the weighted average along the specified axis.



Mean: This will return the arithmetic mean along the specified axis.

std: This will return the standard deviation along the specified axis.

var: This will return the variance along the specified axis.

Example:

```
>>>import numpy as np  
  
>>> x=np.array([[1,2,3],[4,5,6]])  
  
>>>a  
  
array([[[1, 0, 1],  
[0, 0, 1]]])  
  
>>>x  
  
array([[1, 2, 3],  
[4, 5, 6]])  
  
>>>np.median(x)  
  
3.5  
  
>>>np.median(x, axis=0)  
  
array([ 2.5, 3.5, 4.5])  
  
>>>np.average(x)  
  
3.5  
  
>>>np.mean(x)  
  
3.5  
  
>>>np.std(x)  
  
1.707825127659933
```



```
>>>np.var(x)
```

```
2.916666666666665
```

Histograms:

Histogram: This will return the computed histogram of a set of data. This function mainly works with bins and set of data given as input. Numpy histogram function will give the computed result as the occurrences of input data which fall in each of the particular range of bins. That determines the range of area of each bar when plotted using matplotlib.

Example:

```
>>>importnumpy as np
```

```
>>>np.histogram([10,15,16,24,25,45,36,45], bins=[0,10,20,30,40,50])
```

```
(array([0, 3, 2, 1, 2], dtype=int32), array([ 0, 10, 20, 30, 40, 50]))
```

```
>>>importmatplotlib.pyplot as plt
```

```
>>>plt.hist([10,15,16,24,25,45,36,45], bins=[0,10,20,30,40,50])
```

```
(array([ 0., 3., 2., 1., 2.]), array([ 0, 10, 20, 30, 40, 50]), <a list of 5 Patch objects>)
```

```
>>>plt.show()
```

Numpy – Sort, Search & Counting Functions

In Numpy, we can perform various Sorting, Searching, Counting operations using the various functions that are provided in the library like sort, argmax, argmin, count_nonzero etc.

sort: This will return a sorted copy of an array.

argmax: This will return the indices of the maximum values along an axis.

argmin: This will return the indices of the minimum values along an axis.

count_nonzero: This will return count of number of non-zero values in the array.

Example:

```
>>>import numpy as np
```

```
>>>x=np.array([[1,4],[3,2]])
```

```
>>>np.sort(x)
array([[1, 4],
       [2, 3]])
>>>np.sort(x, axis=None)
array([1, 2, 3, 4])
>>>np.sort(x, axis=0)
array([[1, 2],
       [3, 4]])
>>>np.argmax(x)
1

>>>x
array([[1, 4],
       [3, 2]])
>>>np.argmax(x, axis=0)
array([1, 0], dtype=int32)
>>>np.argmax(x, axis=1)
array([1, 0], dtype=int32)
>>>np.argmin(x)
0
>>>np.argmin(x, axis=0)
array([0, 1], dtype=int32)
>>>np.count_nonzero(np.eye(4))
4
>>>np.count_nonzero([[0,1,7,0,0],[3,0,0,2,19]])
```



NumPy – Matrix Library

Matrix module contains the functions which return matrices instead of arrays.

matrix – This will return a matrix from an array kind of an object or from the string of data.

Example:

```
>>> x=np.matrix('1,2,3,4')
```

```
>>>x
```

```
matrix([[1, 2, 3, 4]])
```

```
>>>print(x)
```

```
[[1 2 3 4]]
```

asmatrix – This will interpret the input as matrix.

Example:

```
>>> x=np.array([[1,2],[3,4],[4,5]])
```

```
>>>x
```

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

```
[3, 4],
```

```
[4, 5]])
```

```
>>> y=np.asmatrix(x)
```

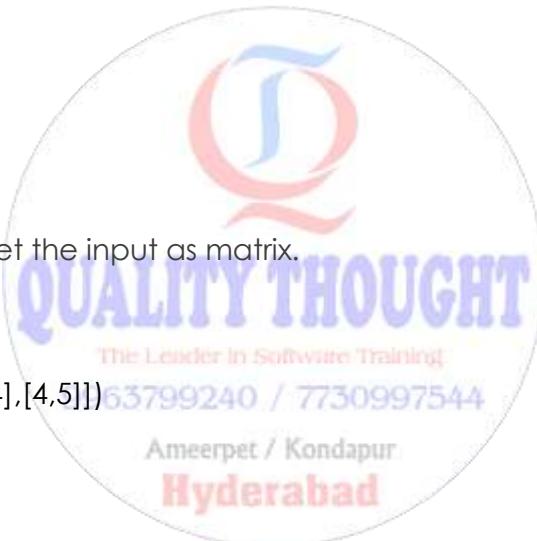
```
>>>y
```

```
matrix([[1, 2],
```

```
[3, 4],
```

```
[4, 5]])
```

```
>>>x[0,0]=5
```



```
>>>y  
matrix([[5, 2],  
[3, 4],  
[4, 5]])
```

Below are the some replacement functions in matlib.

empty: This will return a new matrix of given shape and type.

Example:

```
>> import numpy.matlib as mb  
  
# this will generate the random data  
  
>>>mb.empty((2,2))  
  
matrix([[ 4.00535364e-307, 2.33648882e-307],  
[ 3.44900029e-307, 1.78250172e-312]])  
  
>>>mb.empty((2,2),int)  
  
matrix([[0, 1],  
[2, 3]])  
  
>>>mb.empty((2,2),int)  
  
matrix([[1, 2],  
[3, 4]])  
  
>>>mb.empty((2,2),int)  
  
matrix([[0, 1],  
[2, 3]])
```

zeros: This will return a matrix of given shape and type, filled with zeros.

Example:

```
>>> import numpy.matlib as mb  
  
>>>mb.zeros((3,3))  
  
matrix([[ 0.,  0.,  0.],  
       [ 0.,  0.,  0.],  
       [ 0.,  0.,  0.]])  
  
>>>mb.zeros((3,3),int)  
  
matrix([[0, 0, 0],  
       [0, 0, 0],  
       [0, 0, 0]])
```

ones: This will return a matrix of ones.

Example:

```
>>> import numpy.matlib as mb  
  
>>>mb.ones((3,3))  
  
matrix([[ 1.,  1.,  1.],  
       [ 1.,  1.,  1.],  
       [ 1.,  1.,  1.]])  
  
>>>mb.ones((3,3),int)  
  
matrix([[1, 1, 1],  
       [1, 1, 1],  
       [1, 1, 1]])
```

eye: This will return a matrix with ones on diagonal and zeros elsewhere.

Example:

```
>>> import numpy.matlib as mb
```



```
>>>mb.eye(3)

matrix([[ 1., 0., 0.],
       [ 0., 1., 0.],
       [ 0., 0., 1.]])

>>>mb.eye(3,dtype=int)

matrix([[1, 0, 0],
       [0, 1, 0],
       [0, 0, 1]])
```

identity: This will return a square identity matrix of given size.

Example:

```
>>> import numpy.matlib as mb

>>>mb.identity(3,dtype=int)

matrix([[1, 0, 0],
       [0, 1, 0],
       [0, 0, 1]])

>>>mb.identity(4,dtype=int)

matrix([[1, 0, 0, 0],
       [0, 1, 0, 0],
       [0, 0, 1, 0],
       [0, 0, 0, 1]])
```



rand: This will return a matrix of random values with given shape.

Example:

```
>>> import numpy.matlib as mb
```

```
>>>mb.rand(2,3)  
  
matrix([[ 0.76296068, 0.20586189, 0.18435659],  
[ 0.56268214, 0.47163051, 0.21750408]])
```

if the first argument is tuple, Other arguments are ignored.

```
>>>mb.rand((2,3),int)  
  
matrix([[ 0.34026611, 0.35534349, 0.00456508],  
[ 0.32700061, 0.89152015, 0.74071634]])
```

NumPy – Linear Algebra

Performing Linear Algebra on several matrices which are stacked as an array.

Below are the some of Linear Algebra functions we are going work on.

dot: This will return the dot product of two arrays.

Example:

```
>>> import numpy as np  
  
>>>np.dot(5,4)
```

20

```
>>> a=[[1,2],[3,4]]
```

```
>>> b=[[1,1],[1,1]]
```

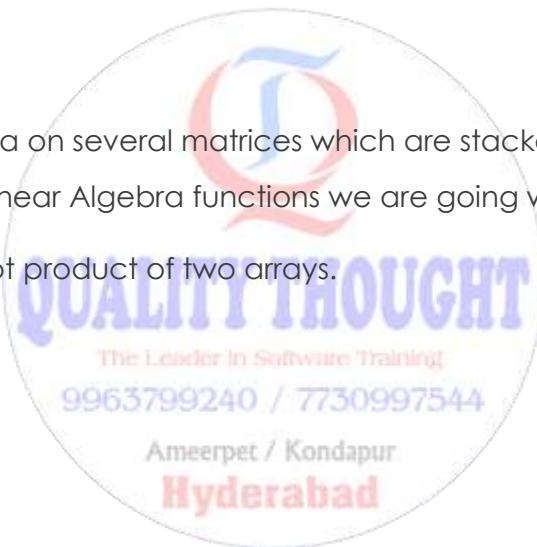
```
>>>np.dot(a,b)
```

```
array([[3, 3],
```

```
[7, 7]])
```

vdot: This will return the dot product of two vectors.

Example:



```
>>> import numpy as np
```

```
>>>np.dot(5,4)
```

20

```
>>> a=[[1,2],[3,4]]
```

```
>>> b=[[1,1],[1,1]]
```

```
>>>np.vdot(a,b)
```

10

This is how it works in the above example ==> $1*1+2*1+3*1+4*1$

inner: This will return the inner product of two arrays.

Example:

```
>>> import numpy as np
```

```
>>> a=[[1,2,3],[0,1,1]]
```

```
>>> b=[[1,2,3],[0,0,1]]
```

```
>>>np.inner(a,b)
```

```
array([[14, 3],
```

```
       [ 5, 1]])
```



outer: This will return the outer product of two vectors.

Example:

```
>>> import numpy as np
```

```
>>> a=[[1,2],[3,4]]
```

```
>>> b=[[1,1],[1,1]]
```

```
>>>np.outer(a,b)
```

```
array([[1, 1, 1, 1],  
       [2, 2, 2, 2],  
       [3, 3, 3, 3],  
       [4, 4, 4, 4]])
```

matmul: This will return the matrix product two arrays.

Example 1:

```
>>> import numpy as np  
  
>>> a=[[1,2],[3,4]]  
  
>>> b=[[1,1],[1,1]]  
  
>>>np.matmul(a,b)  
  
array([[3, 3],  
       [7, 7]])
```

Example 2:

```
>>> import numpy as np  
  
>>> a=[[1,2],[3,4]]  
  
>>> b=[1,1]  
  
>>>np.matmul(a,b)  
  
array([3, 7])  
  
>>>np.matmul(b,a)  
  
array([4, 6])
```

tensordot: This will return the computed tensor dot product along specific axes for arrays \geq 1-D.

Example:



```
>>>import numpy as np  
  
>>> a=np.random.randint(2, size=(1,2,3))  
  
>>> b=np.random.randint(2, size=(3,2,1))  
  
>>>np.tensordot(a,b,axes=((1),(1))).shape  
(1, 3, 3, 1)
```

How it works:

we have chosen 1, 1 axes

In a ==> (1,2,3), b==> (3,2,1)==> ignore 2 now ==> (1,3,3,1)

Numpy – Histogram using Matplotlib

Histogram: This will return the computed histogram of a set of data. This function mainly works with bins and set of data given as input. Numpy histogram function will give the computed result as the occurrences of input data which fall in each of the particular range of bins. That determines the range of area of each bar when plotted using matplotlib.

Example:

```
>>> import numpy as np  
  
>>>np.histogram([10,15,16,24,25,45,36,45], bins=[0,10,20,30,40,50])  
(array([0, 3, 2, 1, 2], dtype=int32), array([ 0, 10, 20, 30, 40, 50]))  
  
>>> import matplotlib.pyplot as plt  
  
>>>plt.hist([10,15,16,24,25,45,36,45], bins=[0,10,20,30,40,50])  
(array([ 0., 3., 2., 1., 2.]), array([ 0, 10, 20, 30, 40, 50]), <a list of 5 Patch objects>)  
  
>>>plt.show()
```

What is Pandas?

Pandas is an opensource library that allows to you perform data manipulation in Python. Pandas library is built on top of Numpy, meaning Pandas needs Numpy to

operate. Pandas provide an easy way to create, manipulate and wrangle the data. Pandas is also an elegant solution for time series data.

Why use Pandas?

Data scientists use Pandas for its following advantages:

- Easily handles missing data
- It uses **Series** for one-dimensional data structure and **DataFrame** for multi-dimensional data structure
- It provides an efficient way to slice the data
- It provides a flexible way to merge, concatenate or reshape the data
- It includes a powerful time series tool to work with

In a nutshell, Pandas is a useful library in data analysis. It can be used to perform data manipulation and analysis. Pandas provide powerful and easy-to-use data structures, as well as the means to quickly perform operations on these structures.

How to install Pandas?

You can install Pandas using:

- Anaconda: conda install -c anaconda pandas
- In Jupyter Notebook :

```
import sys
!conda install --yes --prefix {sys.prefix} pandas
```

Ameerpet / Kondapur
Hyderabad

What is a data frame?

A data frame is a two-dimensional array, with labeled axes (rows and columns). A data frame is a standard way to store data.

Data frame is well-known by statistician and other data practitioners. A data frame is a tabular data, with rows to store the information and columns to name the information. For instance, the price can be the name of a column and 2,3,4 the price values.

Below a picture of a Pandas data frame:

Item	Price
0	A 2
1	B 3

What is a Series?

A series is a one-dimensional data structure. It can have any data structure like integer, float, and string. It is useful when you want to perform computation or return a one-dimensional array. A series, by definition, cannot have multiple columns. For the latter case, please use the data frame structure.

Series has one parameters:

- Data: can be a list, dictionary or scalar value

```
pd.Series([1., 2., 3.])
0    1.0
1    2.0
2    3.0
dtype: float64
```

You can add the index with index. It helps to name the rows. The length should be equal to the size of the column

```
pd.Series([1., 2., 3.], index=['a', 'b', 'c'])
```

Below, you create a Pandas series with a missing value for the third rows. Note, missing values in Python are noted "NaN." You can use numpy to create missing value: np.nan artificially

```
pd.Series([1,2,np.nan])
```

Output

```
0    1.0
1    2.0
2    NaN
dtype: float64
```

Create Data frame

You can convert a numpy array to a pandas data frame with pd.DataFrame(). The opposite is also possible. To convert a pandas Data Frame to an array, you can use np.array()

```
## Numpy to pandas
import numpy as np
h = [[1,2],[3,4]]
df_h = pd.DataFrame(h)
```

```
print('Data Frame:', df_h)

## Pandas to numpy
df_h_n = np.array(df_h)
print('Numpy array:', df_h_n)
Data Frame: 0 1
0 1 2
1 3 4
Numpy array: [[1 2]
 [3 4]]
```

You can also use a dictionary to create a Pandas dataframe.

```
dic = {'Name': ["John", "Smith"], 'Age': [30, 40]}
pd.DataFrame(data=dic)
```

	Age	Name
0	30	John
1	40	Smith

Range Data

Pandas have a convenient API to create a range of date

```
pd.date_range(date,period,frequency):
```

- The first parameter is the starting date
- The second parameter is the number of periods (optional if the end date is specified)
- The last parameter is the frequency: day: 'D', month: 'M' and year: 'Y'.

```
## Create date
# Days
dates_d = pd.date_range('20300101', periods=6, freq='D')
print('Day:', dates_d)
```

Output

```
Day: DatetimeIndex(['2030-01-01', '2030-01-02', '2030-01-03', '2030-01-04', '2030-01-05',
'2030-01-06'], dtype='datetime64[ns]', freq='D')
```

```
# Months
dates_m = pd.date_range('20300101', periods=6, freq='M')
print('Month:', dates_m)
```

Output

Month: DatetimeIndex(['2030-01-31', '2030-02-28', '2030-03-31', '2030-04-30', '2030-05-31', '2030-06-30'], dtype='datetime64[ns]', freq='M')

Inspecting data

You can check the head or tail of the dataset with head(), or tail() preceded by the name of the panda's data frame

Step 1) Create a random sequence with numpy. The sequence has 4 columns and 6 rows

```
random = np.random.randn(6,4)
```

Step 2) Then you create a data frame using pandas.

Use dates_m as an index for the data frame. It means each row will be given a "name" or an index, corresponding to a date.

Finally, you give a name to the 4 columns with the argument columns

```
# Create data with date
df = pd.DataFrame(random,
index=dates_m,
columns=list('ABCD'))
```

Step 3) Using head function

```
df.head(3)
```

	A	B	C	D
2030-01-31	1.139433	1.318510	-0.181334	1.615822
2030-02-28	-0.081995	-0.063582	0.857751	-0.527374
2030-03-31	-0.519179	0.080984	-1.454334	1.314947

Step 4) Using tail function

df.tail(3)

	A	B	C	D
2030-04-30	-0.685448	-0.011736	0.622172	0.104993
2030-05-31	-0.935888	-0.731787	-0.558729	0.768774
2030-06-30	1.096981	0.949180	-0.196901	-0.471556

Step 5) An excellent practice to get a clue about the data is to use describe(). It provides the counts, mean, std, min, max and percentile of the dataset.

df.describe()

	A	B	C	D
count	6.000000	6.000000	6.000000	6.000000
mean	0.002317	0.256928	-0.151896	0.467601
std	0.908145	0.746939	0.834664	0.908910
min	-0.935888	-0.731787	-1.454334	-0.527374
25%	-0.643880	-0.050621	-0.468272	-0.327419
50%	-0.300587	0.034624	-0.189118	0.436883
75%	0.802237	0.732131	0.421296	1.178404
max	1.139433	1.318510	0.857751	1.615822

Slice data

The last point of this tutorial is about how to slice a pandas data frame.

You can use the column name to extract data in a particular column.

```
## Slice
### Using name
df['A']
```

```

2030-01-31 -0.168655
2030-02-28 0.689585
2030-03-31 0.767534
2030-04-30 0.557299
2030-05-31 -1.547836
2030-06-30 0.511551
Freq: M, Name: A, dtype: float64
    
```

To select multiple columns, you need to use two times the bracket, `[[....]]`

The first pair of bracket means you want to select columns, the second pairs of bracket tells what columns you want to return.

`df[['A', 'B']]`.

	A	B
2030-01-31	-0.168655	0.587590
2030-02-28	0.689585	0.998266
2030-03-31	0.767534	-0.940617
2030-04-30	0.557299	0.507350
2030-05-31	-1.547836	1.276558
2030-06-30	0.511551	1.572085

You can slice the rows with :

The code below returns the first three rows

```

### using a slice for row
df[0:3]
    
```

	A	B	C	D
2030-01-31	-0.168655	0.587590	0.572301	-0.031827
2030-02-28	0.689585	0.998266	1.164690	0.475975

2030-03-31	0.767534	-0.940617	0.227255	-0.341532
------------	----------	-----------	----------	-----------

The loc function is used to select columns by names. As usual, the values before the coma stand for the rows and after refer to the column. You need to use the brackets to select more than one column.

```
## Multi col
df.loc[:,['A','B']]
```

	A	B
2030-01-31	-0.168655	0.587590
2030-02-28	0.689585	0.998266
2030-03-31	0.767534	-0.940617
2030-04-30	0.557299	0.507350
2030-05-31	-1.547836	1.276558
2030-06-30	0.511551	1.572085

There is another method to select multiple rows and columns in Pandas. You can use iloc[]. This method uses the index instead of the columns name. The code below returns the same data frame as above

```
df.iloc[:, :2]
```

	A	B
2030-01-31	-0.168655	0.587590
2030-02-28	0.689585	0.998266
2030-03-31	0.767534	-0.940617
2030-04-30	0.557299	0.507350
2030-05-31	-1.547836	1.276558

2030-06-30	0.511551	1.572085
------------	----------	----------

Drop a column

You can drop columns using pd.drop()

```
df.drop(columns=['A', 'C'])
```

	B	D
2030-01-31	0.587590	-0.031827
2030-02-28	0.998266	0.475975
2030-03-31	-0.940617	-0.341532
2030-04-30	0.507350	-0.296035
2030-05-31	1.276558	0.523017
2030-06-30	1.572085	-0.594772

Concatenation

You can concatenate two DataFrame in Pandas. You can use pd.concat()

First of all, you need to create two DataFrames. So far so good, you are already familiar with dataframe creation

```
import numpy as np
df1 = pd.DataFrame({'name': ['John', 'Smith', 'Paul'],
                     'Age': ['25', '30', '50']},
                     index=[0, 1, 2])
df2 = pd.DataFrame({'name': ['Adam', 'Smith'],
                     'Age': ['26', '11']},
                     index=[3, 4])
```

Finally, you concatenate the two DataFrame

```
df_concat = pd.concat([df1, df2])
```

df_concat

	Age	name
0	25	John
1	30	Smith
2	50	Paul
3	26	Adam
4	11	Smith

Drop_duplicates

If a dataset can contain duplicates information use, `drop_duplicates` is an easy to exclude duplicate rows. You can see that `df_concat` has a duplicate observation, `Smith` appears twice in the column `name`.

df_concat.drop_duplicates('name')

	Age	name
0	25	John
1	30	Smith
2	50	Paul
3	26	Adam

Sort values

You can sort value with sort_values

df_concat.sort_values('Age')

	Age	name
4	11	Smith
0	25	John

3	26	Adam
1	30	Smith
2	50	Paul

Rename: change of index

You can use rename to rename a column in Pandas. The first value is the current column name and the second value is the new column name.

```
df_concat.rename(columns={"name": "Surname", "Age": "Age_ppl"})
```

	Age_ppl	Surname
0	25	John
1	30	Smith
2	50	Paul
3	26	Adam
4	11	Smith

Import CSV

During the TensorFlow tutorial, you will use the adult dataset. It is often used with classification task. It is available in this URL <https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data> The data is stored in a CSV format. This dataset includes eight categorical variables:

This dataset includes eight categorical variables:

- workclass
- education
- marital
- occupation
- relationship
- race
- sex
- native_country

moreover, six continuous variables:

- age
- fnlwgt
- education_num
- capital_gain
- capital_loss

hours_week

To import a CSV dataset, you can use the object pd.read_csv(). The basic argument inside is:

Syntax:

```
pandas.read_csv(filepath_or_buffer,sep=',',  
'names=None','index_col=None','skipinitialspace=False')
```

- `filepath_or_buffer`: Path or URL with the data
- `sep=' '`: Define the delimiter to use
- `'names=None'`: Name the columns. If the dataset has ten columns, you need to pass ten names
- `'index_col=None'`: If yes, the first column is used as a row index
- `'skipinitialspace=False'`: Skip spaces after delimiter.

Consider the following Example

```
## Import csv  
import pandas as pd  
## Define path data  
COLUMNS = ['age','workclass','fnlwgt','education','education_num','marital',  
'occupation','relationship','race','sex','capital_gain','capital_loss',  
'hours_week','native_country','label']  
PATH = "https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data"  
df_train = pd.read_csv(PATH,  
skipinitialspace=True,  
names = COLUMNS,  
index_col=False)  
df_train.shape
```

Output:(32561, 15)

Groupby

An easy way to see the data is to use the groupby method. This method can help you to summarize the data by group. Below is a list of methods available with groupby:

- count: count
- min: min
- max: max
- mean: mean
- median: median
- standard deviation: std
- etc

Inside groupby(), you can use the column you want to apply the method.

Let's have a look at a single grouping with the adult dataset. You will get the mean of all the continuous variables by type of revenue, i.e., above 50k or below 50k

```
df_train.groupby(['label']).mean()
```

	age	fnlwgt	Education	capital_gain	capital_loss	hours_week
label			num			
<=50K	36.783738	190340.86517	9.595065	148.752468	53.142921	38.840210
>50K	44.249841	188005.00000	11.611657	4006.142456	195.001530	45.473026

You can get the minimum of age by type of household

```
df_train.groupby(['label'])['age'].min()
```

```
label
<=50K    17
>50K    19
Name: age, dtype: int64
```

You can also group by multiple columns. For instance, you can get the maximum capital gain according to the household type and marital status.

```
df_train.groupby(['label', 'marital'])['capital_gain'].max()
```

```
label  marital
<=50K Divorced      34095
      Married-AF-spouse 2653
      Married-civ-spouse 41310
      Married-spouse-absent 6849
      Never-married     34095
```

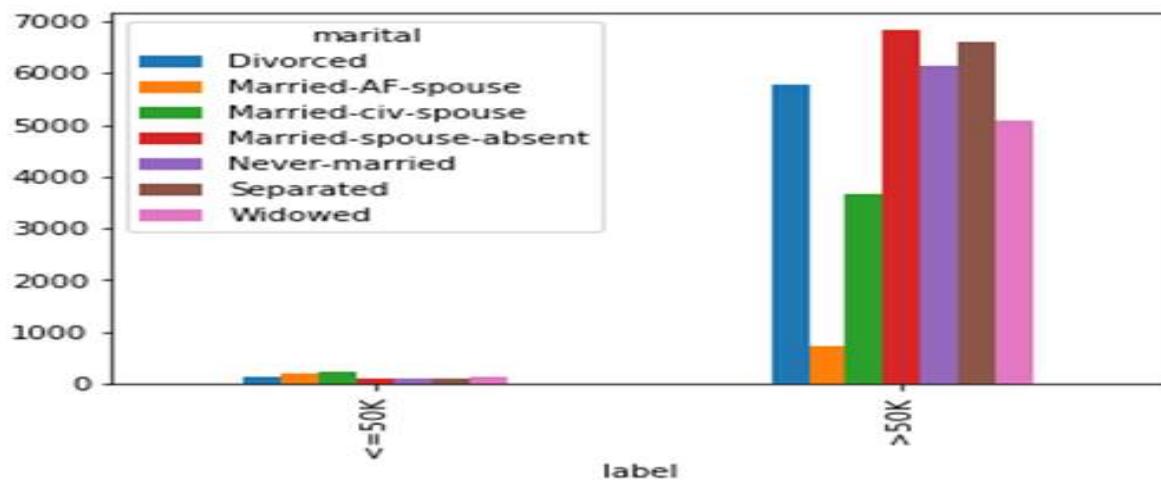
```
Separated           7443
Widowed            6849
>50K Divorced     99999
Married-AF-spouse  7298
Married-civ-spouse 99999
Married-spouse-absent 99999
Never-married      99999
Separated           99999
Widowed             99999
Name: capital_gain, dtype: int64
```

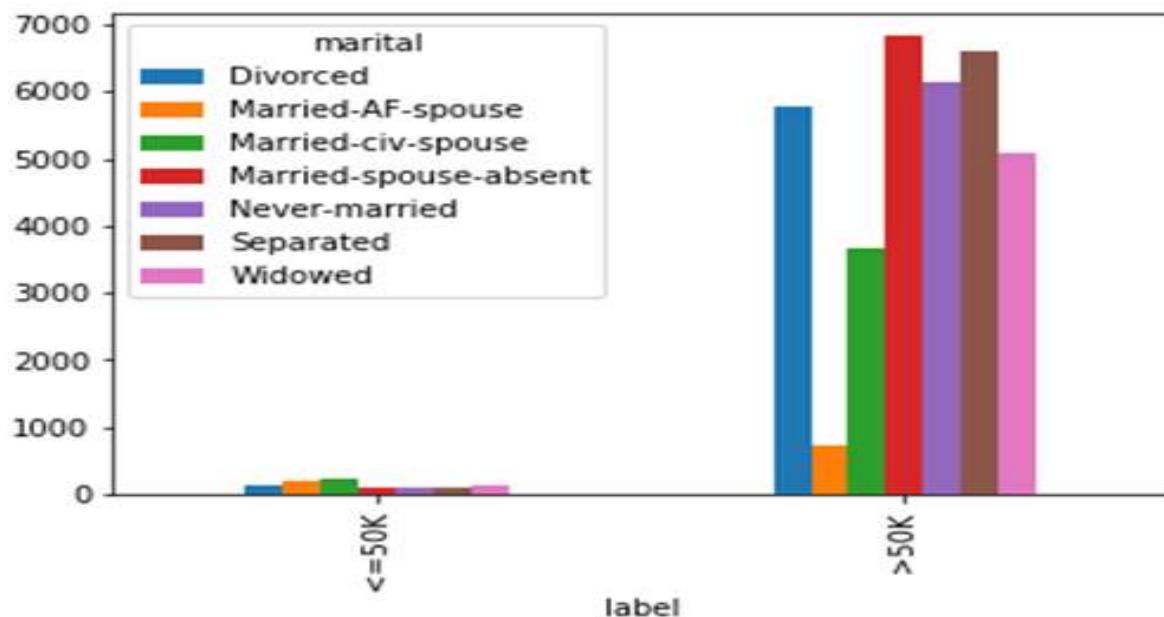
You can create a plot following groupby. One way to do it is to use a plot after the grouping.

To create a more excellent plot, you will use unstack() after mean() so that you have the same multilevel index, or you join the values by revenue lower than 50k and above 50k. In this case, the plot will have two groups instead of 14 (2*7).

If you use Jupyter Notebook, make sure to add % matplotlib inline, otherwise, no plot will be displayed

```
% matplotlib inline
df_plot = df_train.groupby(['label', 'marital'])['capital_gain'].mean().unstack()
df_plot
```





Summary

Below is a summary of the most useful method for data science with Pandas

import data	read_csv
create series	Series
Create Dataframe	DataFrame
Create date range	date_range
return head	head
return tail	tail
Describe	describe
slice using name	dataframe['columnname']
Slice using rows	dataframe[0:5]

Python Seaborn Tutorial

Seaborn is a library for making statistical infographics in Python. It is built on top of matplotlib and also supports numpy and pandas data structures. It also supports statistical units from SciPy.

Visualization plays an important role when we try to explore and understand data, Seaborn is aimed to make it easier and centre of the process. To put in perspective, if we say matplotlib makes things easier and hard things possible, seaborn tries to make that hard easy too, that too in a well-defined way. But seaborn is not an alternative to matplotlib, think of it as a complement to the previous.

As it is built on top of matplotlib, we will often invoke matplotlib functions directly for simple plots as matplotlib has already created highly efficient programs for it.

The high-level interface of seaborn and customizability and variety of backends for matplotlib combined together makes it easy to generate publication-quality figures.

Why Seaborn?

Seaborn offers a variety of functionality which makes it useful and easier than other frameworks. Some of these functionalities are:

- A function to plot statistical time series data with flexible estimation and representation of uncertainty around the estimate
- Functions for visualizing univariate and bivariate distributions or for comparing them between subsets of data
- Functions that visualize matrices of data and use clustering algorithms to discover structure in those matrices
- High-level abstractions for structuring grids of plots that let you easily build complex visualizations
- Several built-in themes for styling matplotlib graphics
- Tools for choosing color palettes to make beautiful plots that reveal patterns in your data
- Tools that fit and visualize linear regression models for different kinds of independent and dependent variables

Getting Started with Seaborn

To get started with Seaborn, we will install it on our machines.

Install Seaborn

Seaborn assumes you have a running Python 2.7 or above platform with NumPY (1.8.2 and above), SciPy(0.13.3 and above) and pandas packages on the devices.

Once we have these python packages installed we can proceed with the installation. For pip installation, run the following command in the terminal:

```
pip install seaborn
```

If you like conda, you can also use conda for package installation, run the following command:

```
conda install seaborn
```

Alternatively, you can use pip to install the development version directly from GitHub:

```
pip install git+https://github.com/mwaskom/seaborn.git
```

Using Seaborn

Once you are done with the installation, you can use seaborn easily in your Python code by importing it:

```
import seaborn
```

Controlling figure aesthetics

When it comes to visualization drawing attractive figures is important.

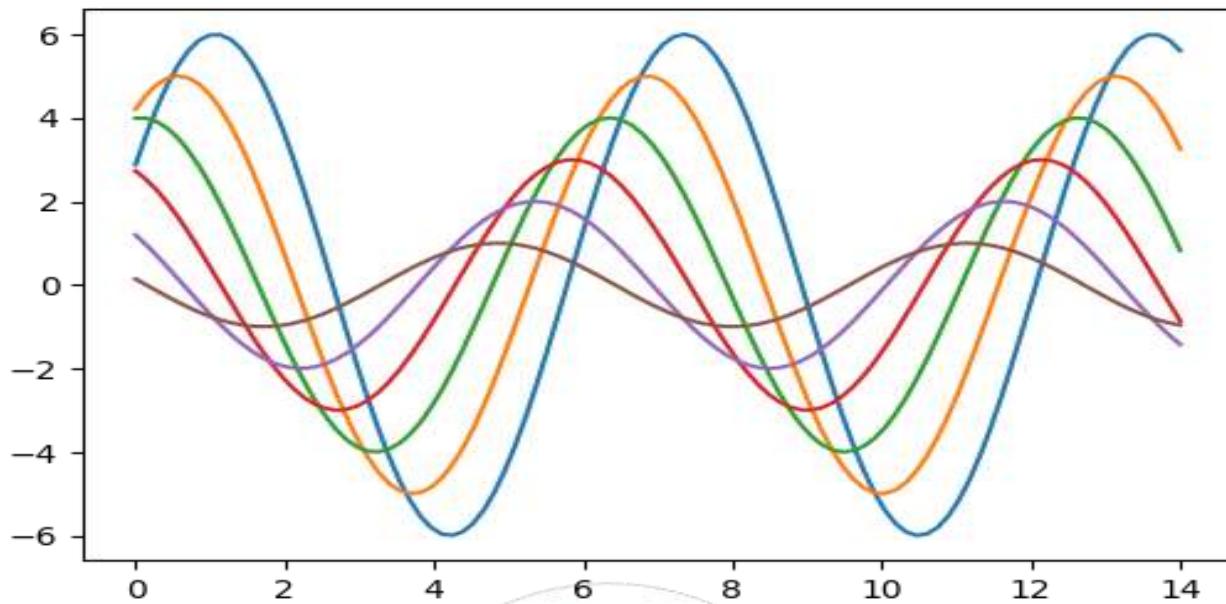
Matplotlib is highly customizable, but it can be complicated at the same time as it is hard to know what settings to tweak to achieve a good looking plot. Seaborn comes with a number of themes and a high-level interface for controlling the look of matplotlib figures. Let's see it working:

```
#matplotlib inline
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns

np.random.seed(sum(map(ord, "aesthetics")))

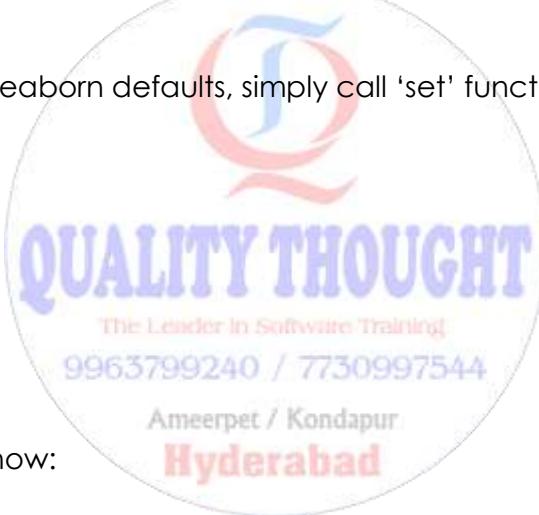
#Define a simple plot function, to plot offset sine waves
def sinplot(flip=1):
    x = np.linspace(0, 14, 100)
    for i in range(1, 7):
        plt.plot(x, np.sin(x + i * .5) * (7 - i) * flip)
    sinplot()
```

This is what the plot looks like with matplotlib defaults:

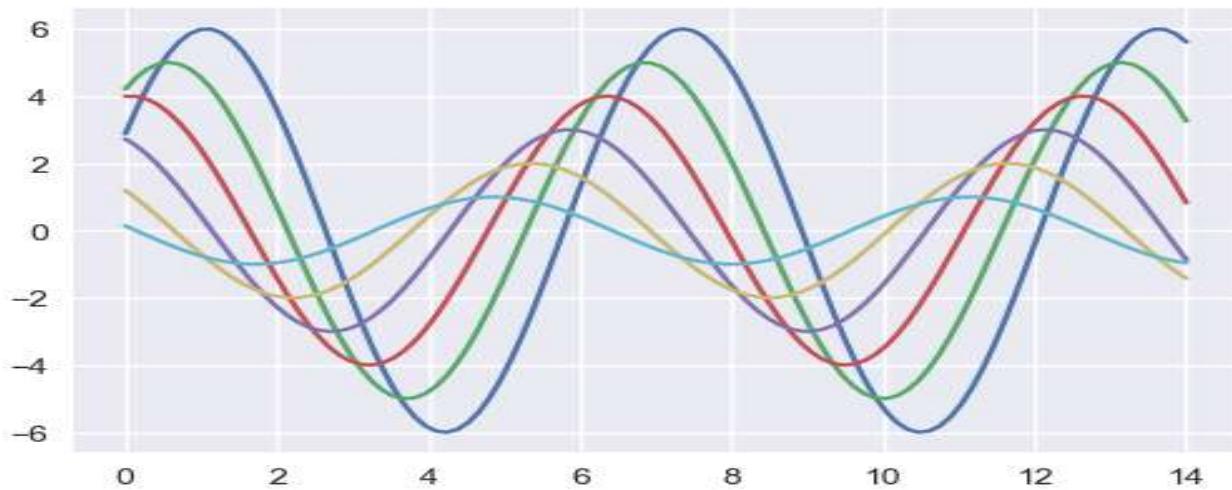


If you want to switch to seaborn defaults, simply call 'set' function:

```
sns.set()  
snsplot()
```



This is how the plot look now:



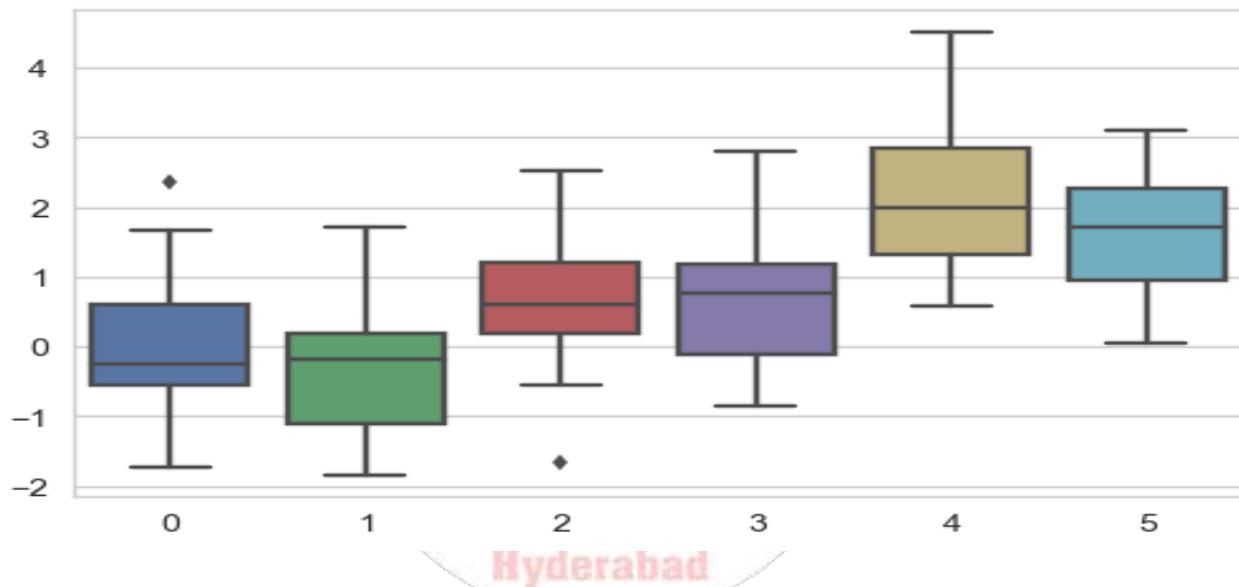
Seaborn figure styles

Seaborn provides five preset themes: white grid, dark grid, white, dark, and ticks, each suited to different applications and also personal preferences.

Darkgrid is the default one. The White grid theme is similar but better suited to plots with heavy data elements, to switch to white grid:

```
sns.set_style("whitegrid")
data = np.random.normal(size=(20, 6)) + np.arange(6) / 2
sns.boxplot(data=data)
```

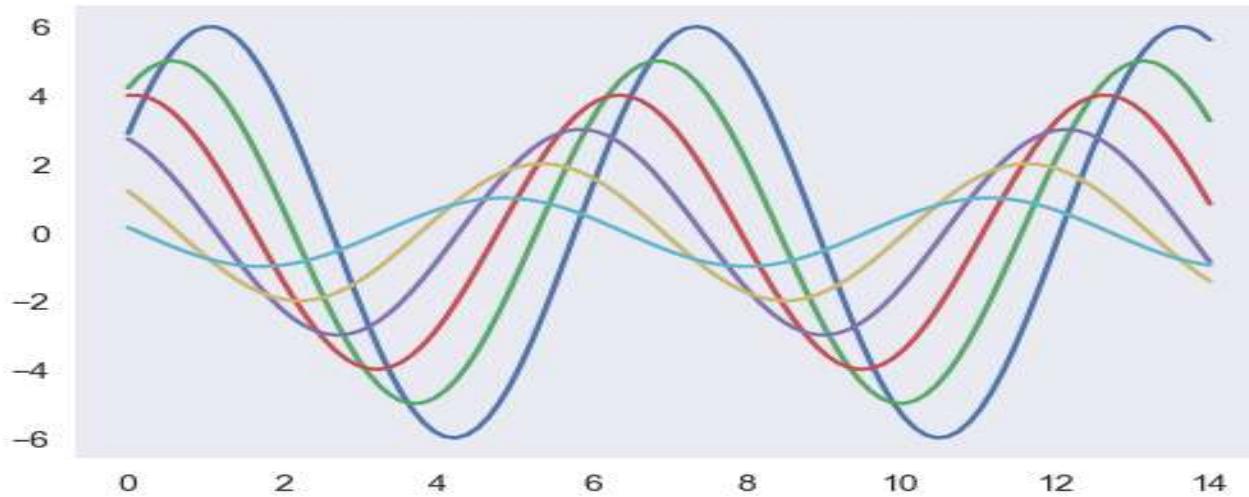
The output will be:



For many plots, the grid is less necessary. Remove it by adding this code snippet:

```
sns.set_style("dark")
sns.set_style("whitegrid")
sns.boxplot(data)
```

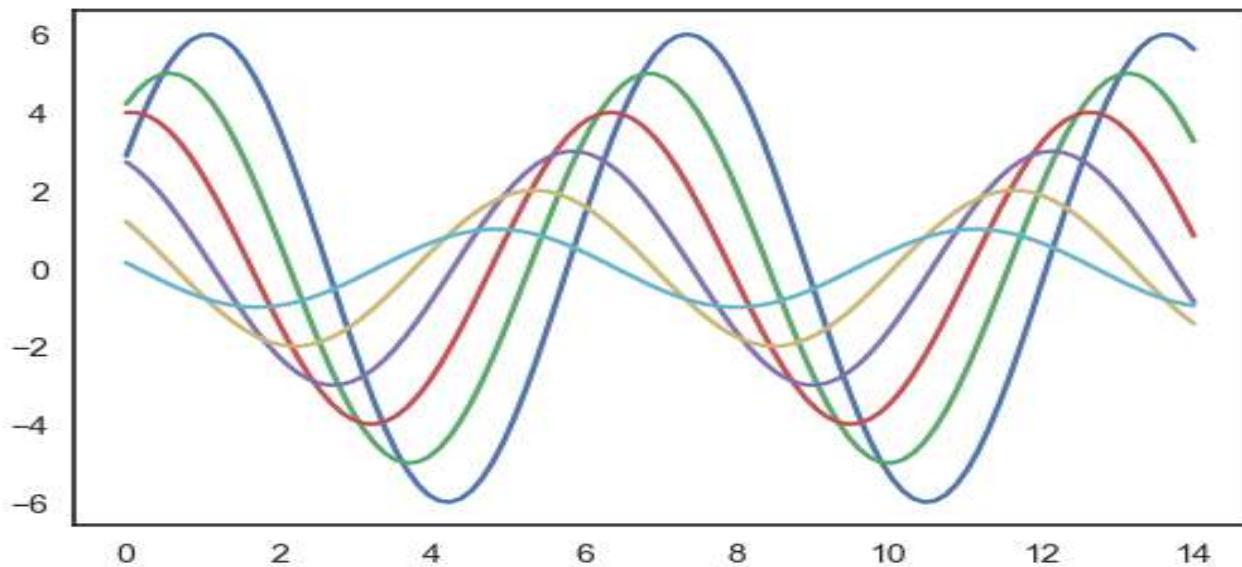
The plot looks like:



Or try the white background:

```
sns.set_style("white")
snsplot()
```

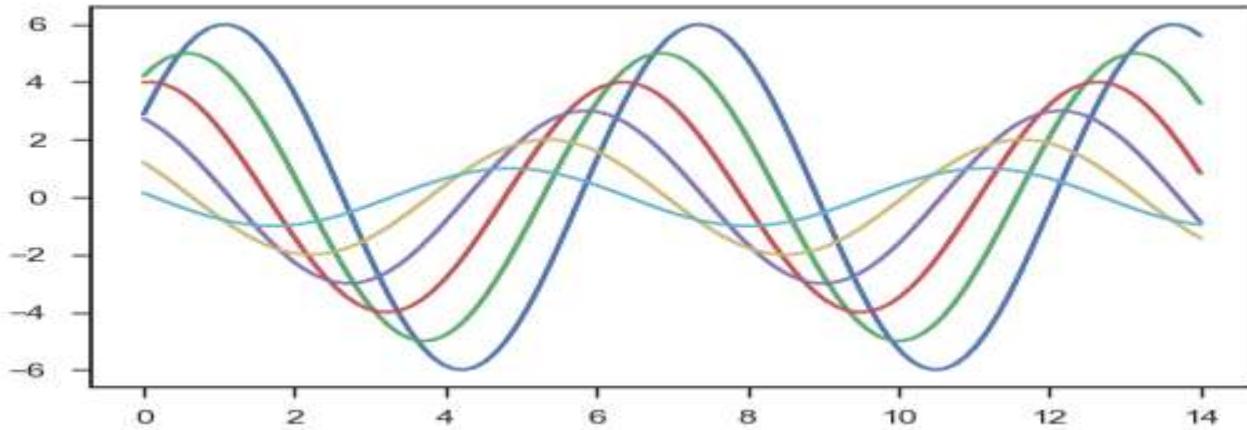
This time, the background looks like:



Sometimes you might want to give a little extra structure to the plots, which is where ticks come in handy:

```
sns.set_style("ticks")
snsplot()
```

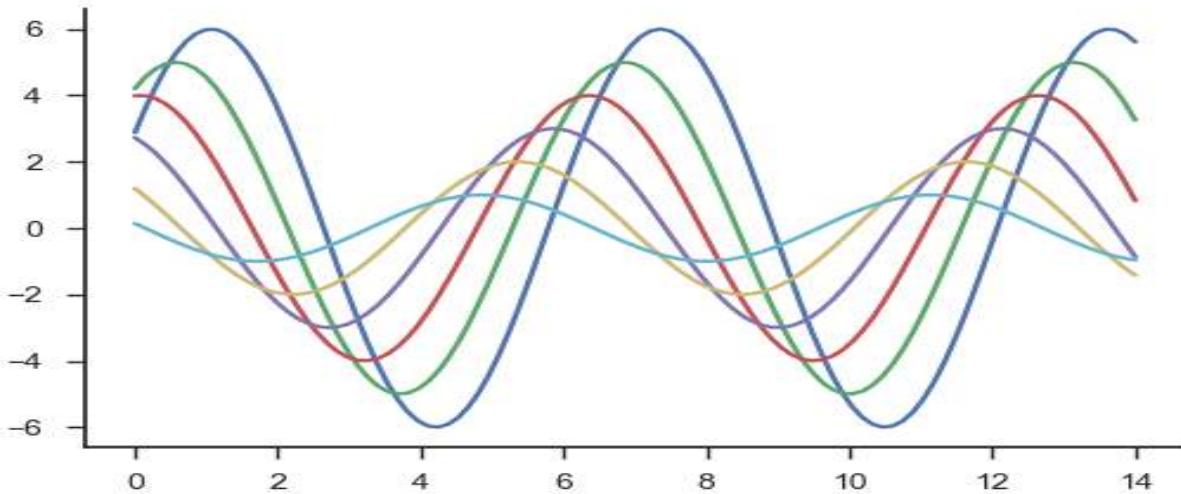
The plot looks like:

**Removing axes spines**

You can call despine function to remove them:

```
sinplot()  
sns.despine()
```

The plot looks like:

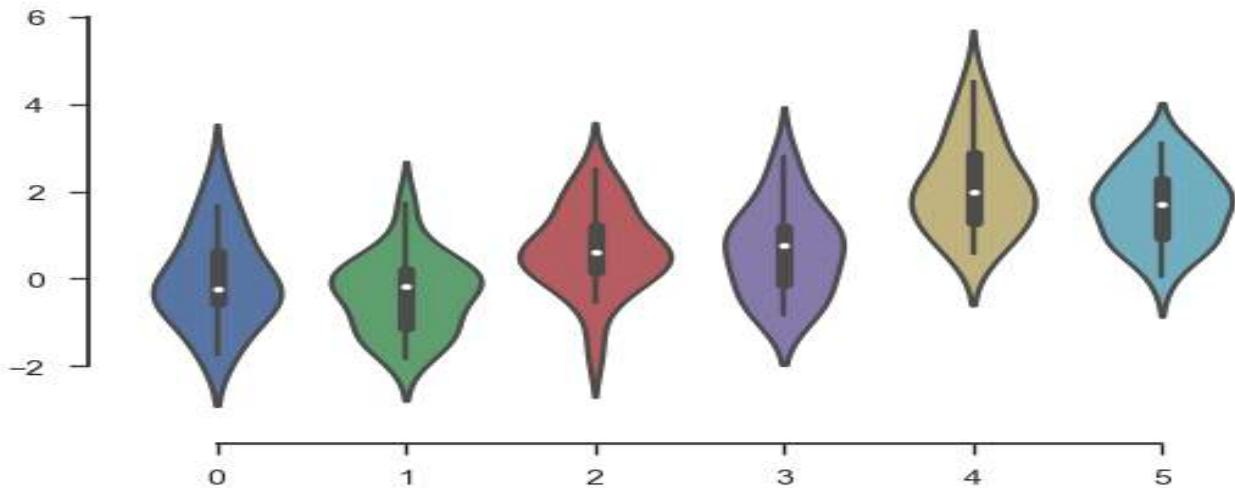


Some plots benefit from offsetting the spines away from the data. When the ticks don't cover the whole range of the axis, the trim parameter will limit the range of the surviving spines:

```
f, ax = plt.subplots()  
sns.violinplot(data=data)
```

```
sns.despine(offset=10, trim=True)
```

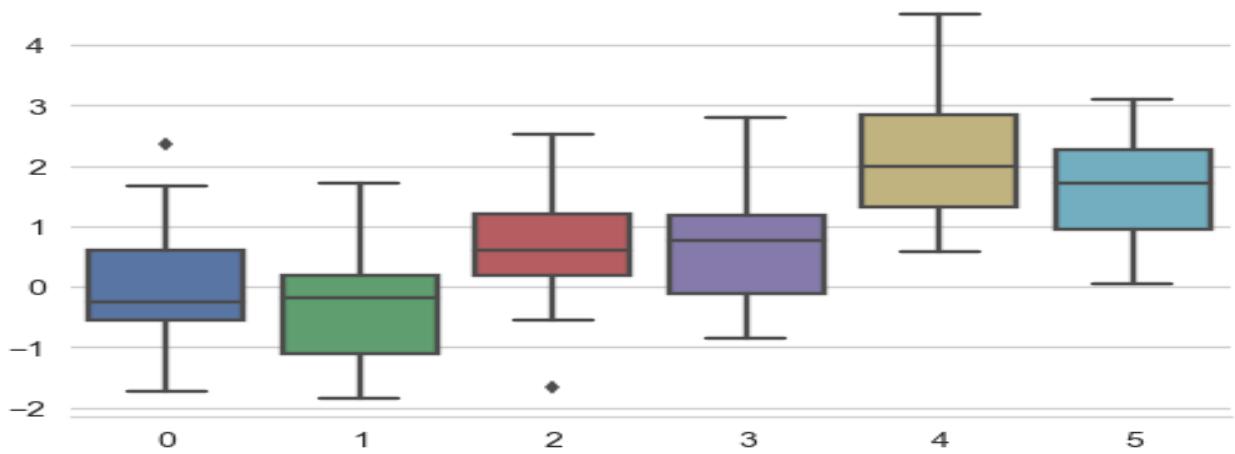
The plot looks like:



You can also control which spines are removed with additional arguments to `despine`:

```
sns.set_style("whitegrid")
sns.boxplot(data=data, palette="deep")
sns.despine(left=True)
```

The plot looks like:

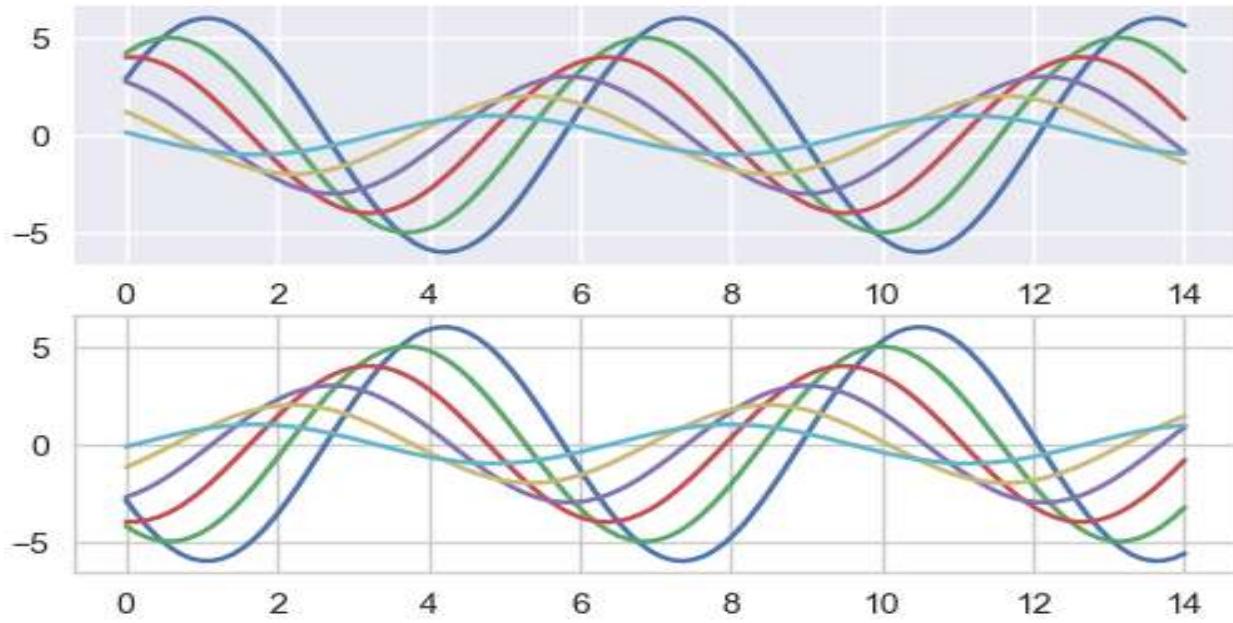


Temporarily setting figure style

`axes_style()` comes to help when you need to set figure style, temporarily:

```
with sns.axes_style("darkgrid"):
    plt.subplot(211)
    sinplot()
    plt.subplot(212)
    sinplot(-1)
```

The plot looks like:



Overriding elements of the seaborn styles

A dictionary of parameters can be passed to the `rc` argument of `axes_style()` and `set_style()` in order to customize figures.

Note: Only the parameters that are part of the style definition through this method can be overridden. For other purposes, you should use `set()` as it takes all the parameters.

In case you want to see what parameters are included, just call the function without any arguments, an object is returned:

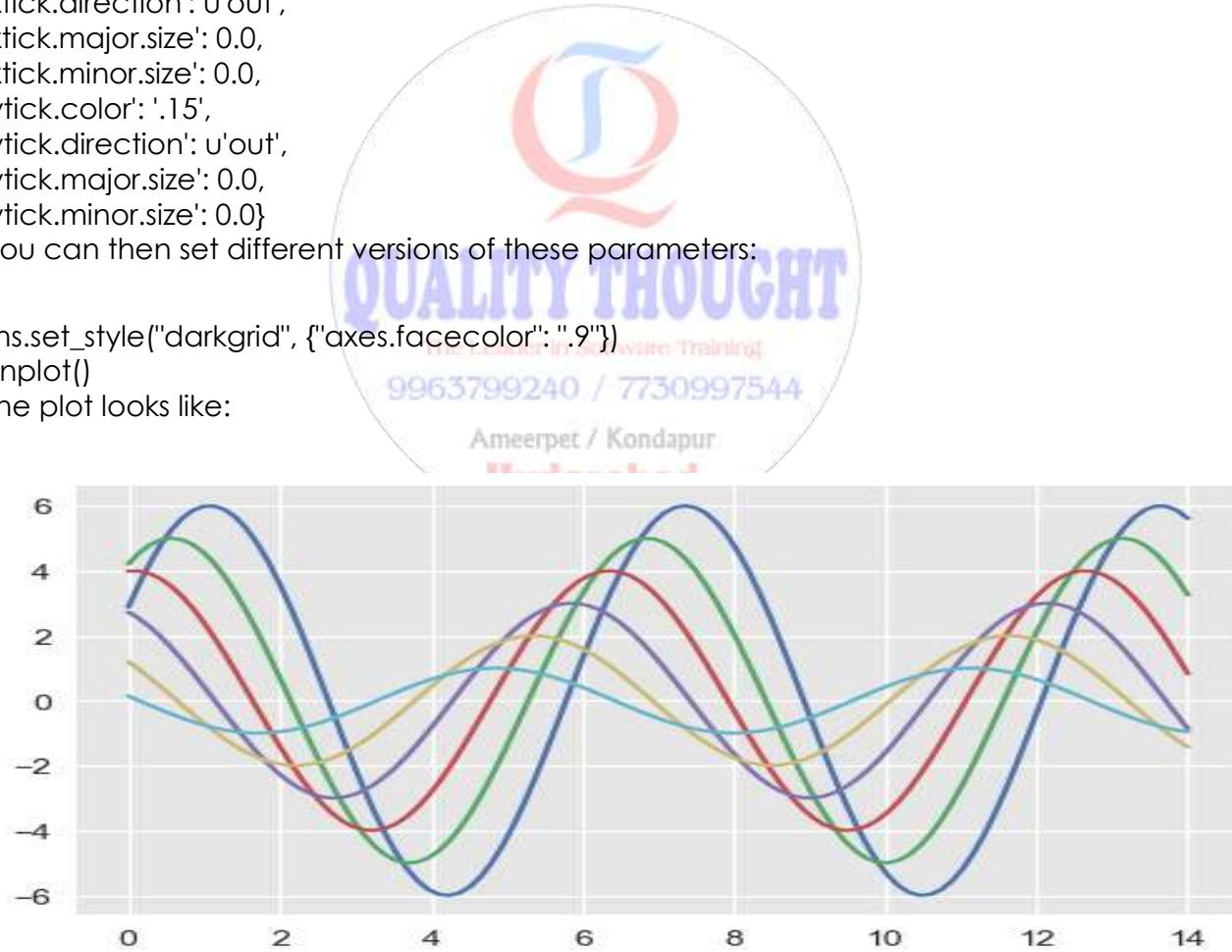
```
sns.axes_style()
{'axes.axisbelow': True,
'axes.edgecolor': '.8',
'axes.facecolor': 'white',
'axes.grid': True,
'axes.labelcolor': '.15',
'axes.linewidth': 1.0,
```

```
'figure.facecolor': 'white',
'font.family': [u'sans-serif'],
'font.sans-serif': [u'Arial',
u'DejaVu Sans',
u'Liberation Sans',
u'Bitstream Vera Sans',
u'sans-serif'],
'grid.color': '.8',
'grid.linestyle': u'-' ,
'image.cmap': u'rocket',
'legend.frameon': False,
'legend.numpoints': 1,
'legend.scatterpoints': 1,
'lines.solid_capstyle': u'round',
'text.color': '.15',
'xtick.color': '.15',
'xtick.direction': u'out',
'xtick.major.size': 0.0,
'xtick.minor.size': 0.0,
'ytick.color': '.15',
'ytick.direction': u'out',
'ytick.major.size': 0.0,
'ytick.minor.size': 0.0}
```

You can then set different versions of these parameters:

```
sns.set_style("darkgrid", {"axes.facecolor": ".9"})  
snsplot()
```

The plot looks like:



Scaling plot elements

Let's try to manipulate scale of the plot. We can reset the default parameters by calling `set()`:

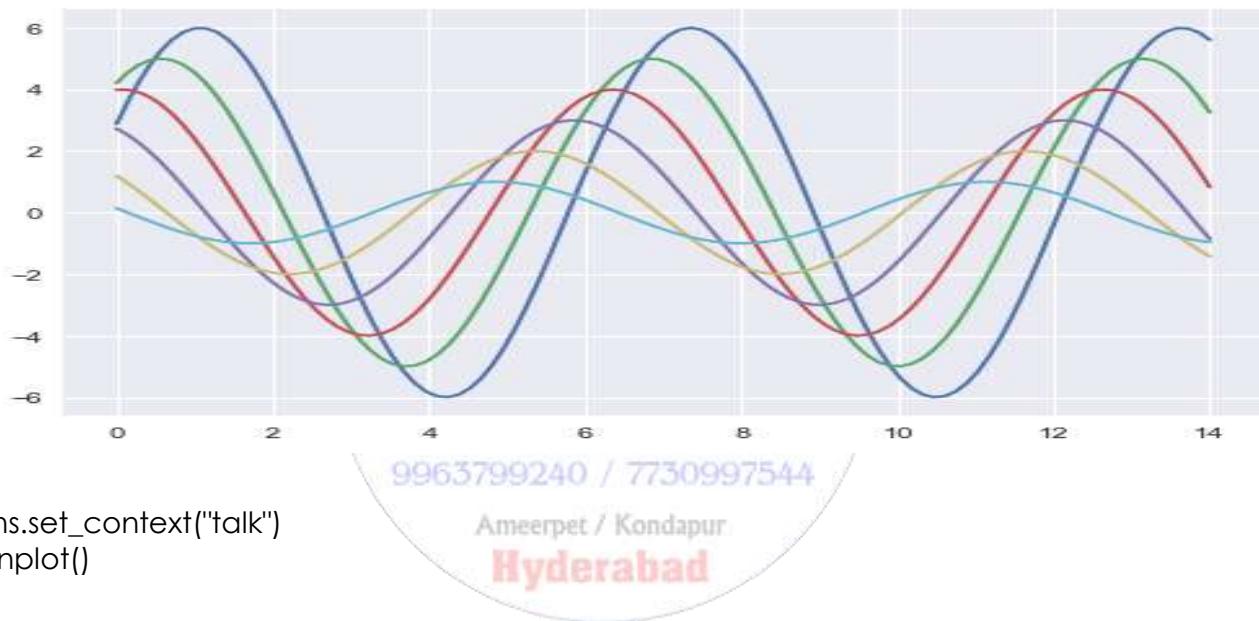
```
sns.set()
```

The four preset contexts are – paper, notebook, talk and poster. The notebook style is the default, and was used in the plots above:

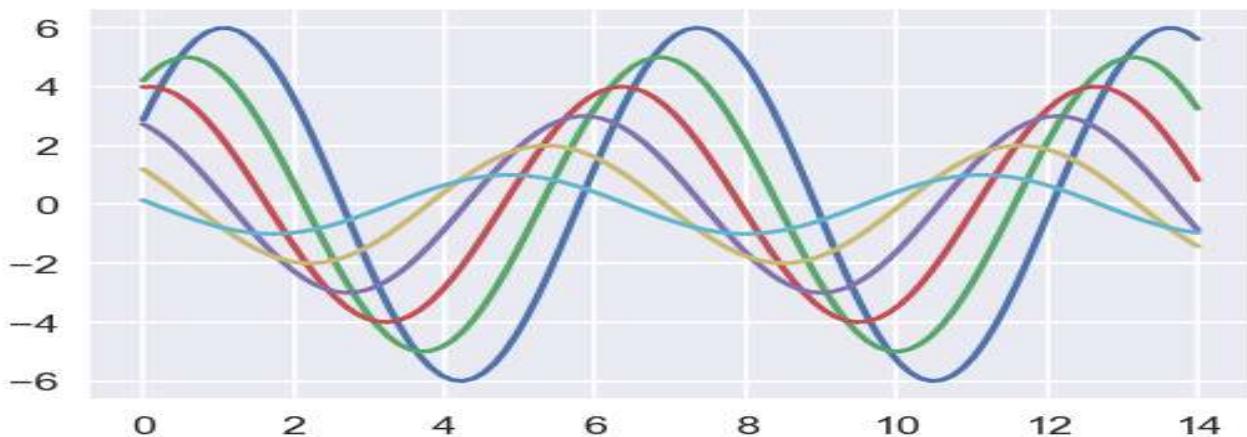
```
sns.set_context("paper")
```

```
sinplot()
```

The plot looks like:



The plot looks like:



What is Matplotlib

To make necessary statistical inferences, it becomes necessary to visualize your data and Matplotlib is one such solution for the Python users. It is a very powerful plotting library useful for those working with Python and NumPy. The most used module of Matplotlib is Pyplot which provides an interface like MATLAB but instead, it uses Python and it is open source.

Installing Matplotlib

To install Matplotlib on your local machine, open Python command prompt and type following commands:

```
python -m pip install -U pip
```

```
python -m pip install -U matplotlib
```

It installs python, Jupyter notebook and other important python libraries including Matplotlib, Numpy, Pandas, scikit-learn. Anaconda supports Windows, MacOS and Linux. To quickly get started with Matplotlib without installing anything on your local machine, check out Google Colab. It provides the Jupyter Notebooks hosted on the cloud for free which are associated with your Google Drive account and it comes with all the important packages pre-installed. You can also run your code on GPU which helps in faster computation though we don't need GPU computation for this tutorial.

General Concepts

A Matplotlib figure can be categorized into several parts as below:

Figure: It is a whole figure which may contain one or more than one axes (plots). You can think of a Figure as a canvas which contains plots.

Axes: It is what we generally think of as a plot. A Figure can contain many Axes. It contains two or three (in the case of 3D) Axis objects. Each Axes has a title, an x-label and a y-label.

Axis: They are the number line like objects and take care of generating the graph limits.

Artist: Everything which one can see on the figure is an artist like Textobjects, Line2D objects, collection objects. Most Artists are tied to Axes.

Getting Started with Pyplot

Pyplot is a module of Matplotlib which provides simple functions to add plot elements like lines, images, text, etc. to the current axes in the current figure.

Make a simple plot

```
import matplotlib.pyplot as plt
```

```
import numpy as np
```

Here we import Matplotlib's Pyplot module and Numpy library as most of the data that we will be working with will be in the form of arrays only.

We pass two arrays as our input arguments to Pyplot's `plot()` method and use `show()` method to invoke the required plot. Here note that the first array appears on the x-axis and second array appears on the y-axis of the plot. Now that our first plot is ready, let us add the title, and name x-axis and y-axis using methods `title()`, `xlabel()` and `ylabel()` respectively.

We can also specify the size of the figure using method `figure()` and passing the values as a tuple of the length of rows and columns to the argument `figsize`

With every X and Y argument, you can also pass an optional third argument in the form of a string which indicates the colour and line type of the plot. The default format is `b`- which means a solid blue line. In the figure below we use `go` which means green circles. Likewise, we can make many such combinations to format our plot.

We can also plot multiple sets of data by passing in multiple sets of arguments of X and Y axis in the `plot()` method as shown.

Multiple plots in one figure:

We can use `subplot()` method to add more than one plots in one figure. In the image below, we used this method to separate two graphs which we plotted on the same axes in the previous example. The `subplot()` method takes three arguments: they are `nrows`, `ncols` and `index`. They indicate the number of rows, number of columns and the index number of the sub-plot. For instance, in our example, we want to create two sub-plots in one figure such that it comes in one row and in two columns and hence we pass arguments `(1,2,1)` and `(1,2,2)` in the `subplot()` method. Note that we have separately used `title()` method for both the subplots. We use `suptitle()` method to make a centralized title for the figure.

If we want our sub-plots in two rows and single column, we can pass arguments (2,1,1) and (2,1,2)

The above way of creating subplots becomes a bit tedious when we want many subplots in our figure. A more convenient way is to use subplots() method. Notice the difference of 's' in both the methods. This method takes two arguments nrows and ncols as number of rows and number of columns respectively. This method creates two objects: figure and axes which we store in variables fig and ax which can be used to change the figure and axes level attributes respectively. Note that these variable names are chosen arbitrarily.

Creating different types of graphs with Pyplot

1) Bar Graphs

Bar graphs are one of the most common types of graphs and are used to show data associated with the categorical variables. Pyplot provides a method bar() to make bar graphs which take arguments: categorical variables, their values and color (if you want to specify any).

To make horizontal bar graphs use method barh(). Also we can pass an argument (with its value) xerr or yerr (in case of the above vertical bar graphs) to depict the variance in our data as follows:

To create horizontally stacked bar graphs we use the bar() method twice and pass the arguments where we mention the index and width of our bar graphs in order to horizontally stack them together. Also, notice the use of two other methods legend() which is used to show the legend of the graph and xticks() to label our x-axis based on the position of our bars.

Similarly, to vertically stack the bar graphs together, we can use an argument bottom and mention the bar graph which we want to stack below as its value.

2) Pie Charts

One more basic type of chart is a Pie chart which can be made using the method pie(). We can also pass in arguments to customize our Pie chart to show shadow, explode a part of it, tilt it at an angle as follows:

3) Histogram

Histograms are a very common type of plots when we are looking at data like height and weight, stock prices, waiting time for a customer, etc which are continuous in nature. Histogram's data is plotted within a range against its frequency. Histograms are very commonly occurring graphs in probability and

statistics and form the basis for various distributions like the normal -distribution, t-distribution, etc. In the following example, we generate a random continuous data of 1000 entries and plot it against its frequency with the data divided into 10 equal strata. We have used NumPy's random.randn()method which generates data with the properties of a standard normal distribution i.e. mean = 0 and standard deviation = 1, and hence the histogram looks like a normal distribution curve.

4) Scatter Plots and 3-D plotting

Scatter plots are widely used graphs, especially they come in handy in visualizing a problem of regression. In the following example, we feed in arbitrarily created data of height and weight and plot them against each other. We used xlim() and ylim() methods to set the limits of X-axis and Y-axis respectively.

The above scatter can also be visualized in three dimensions. To use this functionality, we first import the module mplot3d as follows:

```
from mpl_toolkits import mplot3d
```

Once the module is imported, a three-dimensional axes is created by passing the keyword projection='3d' to the axes() method of Pyplot module. Once the object instance is created, we pass our arguments height and weight to scatter3D() method.

We can also create 3-D graphs of other types like line graph, surface, wireframes, contours, etc. The above example in the form of a simple line graph is as follows: Here instead of scatter3D() we use method plot3D()

Summary

Hope this article was useful to you. If you liked this article please express your appreciation. Before we end the article here is the list of all the methods as they appeared.

- ✓ plot(x-axis values, y-axis values)—plots a simple line graph with x-axis values against y-axis values
- ✓ show()—displays the graph
- ✓ title("string")—set the title of the plot as specified by the string
- ✓ xlabel("string")—set the label for x-axis as specified by the string
- ✓ ylabel("string")—set the label for y-axis as specified by the string
- ✓ figure()—used to control a figure level attributes
- ✓ subplot(nrows, ncols, index)—Add a subplot to the current figure

- ✓ `subtitle("string")`—It adds a common title to the figure specified by the string
- ✓ `subplots(nrows, ncols, figsize)`—a convenient way to create subplots, in a single call. It returns a tuple of a figure and number of axes.
- ✓ `set_title("string")`—an axes level method used to set the title of subplots in a figure
- ✓ `bar(categorical variables, values, color)`—used to create vertical bar graphs
- ✓ `barh(categorical variables, values, color)`—used to create horizontal bar graphs
- ✓ `legend(loc)`—used to make legend of the graph
- ✓ `xticks(index, categorical variables)`—Get or set the current tick locations and labels of the x-axis
- ✓ `pie(value, categorical variables)`—used to create a pie chart
- ✓ `hist(values, number of bins)`—used to create a histogram
- ✓ `xlim(start value, end value)`—used to set the limit of values of the x-axis
- ✓ `ylim(start value, end value)`—used to set the limit of values of the y-axis
- ✓ `scatter(x-axis values, y-axis values)`—plots a scatter plot with x-axis values against y-axis values
- ✓ `axes()`—adds an axes to the current figure
- ✓ `set_xlabel("string")`—axes level method used to set the x-label of the plot specified as a string
- ✓ `set_ylabel("string")`—axes level method used to set the y-label of the plot specified as a string
- ✓ `scatter3D(x-axis values, y-axis values)`—plots a three-dimensional scatter plot with x-axis values against y-axis values
- ✓ `plot3D(x-axis values, y-axis values)`—plots a three-dimensional line graph with x-axis values against y-axis values

AI - Data Science Diploma

R

Programming

2019
EDITION

Quality Thought®

Reach Us:

Quality Thought Infosystems Pvt. Ltd.
#208B, 2nd Floor, Nilgiri Block, Aditya Enclave
Ameerpet, Hyderabad, Telangana - 38

+ 91 - 9515151992

About Institute?

Quality Thought is a professional Training Organization for software developers, IT administrators, and other professionals. It's Located in Hyderabad, India. The training is offered in Four major modes: Classroom Room Trainings, Online instructor Led Trainings, Self-paced e-learning trainings, and Corporate Trainings.

About Course?

This is an Artificial Intelligence Engineer Master Course that is a comprehensive learning approach for mastering the domains of Artificial Intelligence, Data Science, Business Analytics, Business Intelligence, Python coding, and Deep Learning with TensorFlow. Upon completion of the training, you will be able to take on challenging roles in the artificial intelligence domain.

why we take course?

Artificial intelligence is one of the hottest domains being heralded as the one with the ability to disrupt companies cutting across industry sectors. This Quality Thought Artificial Intelligence Engineer Master Course will equip you with all the necessary skills needed to take on challenging and exciting roles in the artificial intelligence, data science, business analytics, Python, R statistical computing domains and grab the best jobs in the industry at top-notch salaries.

Courses Covered in AI Diploma:

Course 1: As a Future ready IT employee I am interested to learn foundations of Data Analytics with statistics and Tableau, R

Course 2: I want to redefine my Analytical knowledge into python programming for Machine Learning Engineer

Course 3: I want to apply statistics and Python on Machine Learning models for Predictions& classifications of Data in various industry segments for intelligent Business

Course 4: So now I am ready to apply machine Learning models to implement Recommendation systems and Creating Machine Learning service in the cloud(IBM WATSON,AWS)

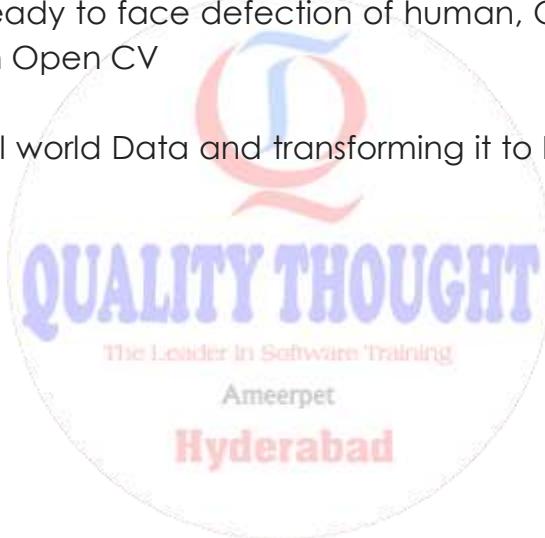
Course 5: Understanding tools of Bigdata and implementation using Machine Learning

Course 6: Applying Machine learning to speech Analytics to process Text using NLP

Course 7: Start Learning Neural Networks using Tensor flows and Keras for image classification and Data Extraction from Image(OCR)

Course 8: Now I am ready to face detection of human, Object using computer vision Technology with Open CV

Course 9: Sensing Real world Data and transforming it to Intelligent actions using IOT



R is a programming language and software environment for statistical analysis, graphics representation and reporting. R was created by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand, and is currently developed by the R Development Core Team.

The core of R is an interpreted computer language which allows branching and looping as well as modular programming using functions. R allows integration with the procedures written in the C, C++, .Net, Python or FORTRAN languages for efficiency.

R is freely available under the GNU General Public License, and pre-compiled binary versions are provided for various operating systems like Linux, Windows and Mac.

R is free software distributed under a GNU-style copy left, and an official part of the GNU project called **GNU S**.

Evolution of R

R was initially written by **Ross Ihaka** and **Robert Gentleman** at the Department of Statistics of the University of Auckland in Auckland, New Zealand. R made its first appearance in 1993.

- A large group of individuals has contributed to R by sending code and bug reports.
- Since mid-1997 there has been a core group the "RCoreTeam" the "RCoreTeam" who can modify the R source code archive.

Features of R

As stated earlier, R is a programming language and software environment for statistical analysis, graphics representation and reporting. The following are the important features of R –

- R is a well-developed, simple and effective programming language which includes conditionals, loops, user defined recursive functions and input and output facilities.
- R has an effective data handling and storage facility,
- R provides a suite of operators for calculations on arrays, lists, vectors and matrices.
- R provides a large, coherent and integrated collection of tools for data analysis.
- R provides graphical facilities for data analysis and display either directly at the computer or printing at the papers.

As a conclusion, R is world's most widely used statistics programming language. It's the # 1 choice of data scientists and supported by a vibrant and talented community of contributors. R is taught in universities and deployed in mission critical business applications. This tutorial will teach you R programming along with suitable examples in simple and easy steps.

R - ENVIRONMENT SETUP

Local Environment Setup

If you are still willing to set up your environment for R, you can follow the steps given below.

Windows Installation

You can download the Windows installer version of R from [R-3.2.2 for Windows 32/64bit](#) and save it in a local directory.

As it is a Windows installer .exe.exe with a name "R-version-win.exe". You can just double click and run the installer accepting the default settings. If your Windows is 32-bit version, it installs the 32-bit version. But if your windows is 64-bit, then it installs both the 32-bit and 64-bit versions.

After installation you can locate the icon to run the Program in a directory structure "R\R3.2.2\bin\i386\Rgui.exe" under the Windows Program Files. Clicking this icon brings up the R-GUI which is the R console to do R Programming.

Linux Installation

R is available as a binary for many versions of Linux at the location [R Binaries](#).

The instruction to install Linux varies from flavor to flavor. These steps are mentioned under each type of Linux version in the mentioned link. However, if you are in a hurry, then you can use **yum** command to install R as follows –

```
$ yum install R
```

Above command will install core functionality of R programming along with standard packages, still you need additional package, then you can launch R prompt as follows –

```
$ R
R version 3.2.0 (2015-04-16) -- "Full of Ingredients"
Copyright (C) 2015 The R Foundation for Statistical Computing
Platform: x86_64-redhat-linux-gnu (64-bit)
```

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and

'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or 'help.start()' for an HTML browser interface to help.

Type 'q()' to quit R.

>

Now you can use install command at R prompt to install the required package. For example, the following command will install **plotrix** package which is required for 3D charts.

```
> install.packages("plotrix")
```

R - BASIC SYNTAX

As a convention, we will start learning R programming by writing a "Hello, World!" program. Depending on the needs, you can program either at R command prompt or you can use an R script file to write your program. Let's check both one by one.

R Command Prompt

Once you have R environment setup, then it's easy to start your R command prompt by just typing the following command at your command prompt –

```
$ R
```

This will launch R interpreter and you will get a prompt > where you can start typing your program as follows –

```
> myString <- "Hello, World!"  
> print(myString)  
[1]"Hello, World!"
```

Here first statement defines a string variable myString, where we assign a string "Hello, World!" and then next statement print is being used to print the value stored in variable myString.

R Script File

Usually, you will do your programming by writing your programs in script files and then you execute those scripts at your command prompt with the help of R interpreter

called **Rscript**. So let's start with writing following code in a text file called test.R as under –

```
# My first program in R Programming  
myString <- "Hello, World!"  
  
print( myString)
```

Save the above code in a file test.R and execute it at Linux command prompt as given below. Even if you are using Windows or other system, syntax will remain same.

```
$ Rscript test.R
```

When we run the above program, it produces the following result.

```
[1] "Hello, World!"
```

Comments

Comments are like helping text in your R program and they are ignored by the interpreter while executing your actual program. Single comment is written using # in the beginning of the statement as follows –

```
# My first program in R Programming
```

R does not support multi-line comments but you can perform a trick which is something as follows –

```
if(FALSE){  
  "This is a demo for multi-line comments and it should be put inside either a  
  single OR double quote"  
}  
  
myString <- "Hello, World!"  
print( myString)  
[1] "Hello, World!"
```

Though above comments will be executed by R interpreter, they will not interfere with your actual program. You should put such comments inside, either single or double quote.

R - DATA TYPES

Generally, while doing programming in any programming language, you need to use various variables to store various information. Variables are nothing but reserved memory locations to store values. This means that, when you create a variable you reserve some space in memory.

You may like to store information of various data types like character, wide character, integer, floating point, double floating point, Boolean etc. Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the reserved memory.

In contrast to other programming languages like C and java in R, the variables are not declared as some data type. The variables are assigned with R-Objects and the data type of the R-object becomes the data type of the variable. There are many types of R-objects. The frequently used ones are –

- Vectors
- Lists
- Matrices
- Arrays
- Factors
- Data Frames

The simplest of these objects is the **vector object** and there are six data types of these atomic vectors, also termed as six classes of vectors. The other R-Objects are built upon the atomic vectors.

Data Type	Example	Verify
Logical	TRUE, FALSE	v <- TRUE print(class(v)) it produces the following result –

		[1] "logical"
Numeric	12.3, 5, 999	<pre>v <- 23.5 print(class(v))</pre> <p>it produces the following result –</p>
		[1] "numeric"
Integer	2L, 34L, 0L	<pre>v <- 2L print(class(v))</pre> <p>it produces the following result –</p>
		[1] "integer"
Complex	3 + 2i	<pre>v <- 2+5i print(class(v))</pre> <p>it produces the following result –</p>
		[1] "complex"
Character	'a', "good", "TRUE", '23.4'	<pre>v <- "TRUE" print(class(v))</pre> <p>it produces the following result –</p>
		[1] "character"
Raw	"Hello" is stored as 48 65 6c 6c 6f	<pre>v <- charToRaw("Hello") print(class(v))</pre> <p>it produces the following result –</p>
		[1] "raw"

In R programming, the very basic data types are the R-objects called **vectors** which hold elements of different classes as shown above. Please note in R the number of classes is not confined to only the above six types. For example, we can use many atomic vectors and create an array whose class will become array.

Vectors

When you want to create vector with more than one element, you should use **c** function which means to combine the elements into a vector.

```
# Create a vector.  
apple <- c('red','green',"yellow")  
print(apple)  
  
# Get the class of the vector.  
print(class(apple))
```

When we execute the above code, it produces the following result –

```
[1] "red"  "green" "yellow"  
[1] "character"
```

Lists

A list is an R-object which can contain many different types of elements inside it like vectors, functions and even another list inside it.

```
# Create a list.  
list1 <- list(c(2,5,3),21.3,sin)  
  
# Print the list.  
print(list1)
```

When we execute the above code, it produces the following result –

```
[[1]]  
[1] 2 5 3
```

```
[[2]]  
[1] 21.3  
  
[[3]]  
function (x) .Primitive("sin")
```

Matrices

A matrix is a two-dimensional rectangular data set. It can be created using a vector input to the matrix function.

```
# Create a matrix.  
M = matrix( c('a','a','b','c','b','a'), nrow =2, ncol =3, byrow = TRUE)  
print(M)
```

When we execute the above code, it produces the following result –

```
[,1] [,2] [,3]  
[1,] "a" "a" "b"  
[2,] "c" "b" "a"
```

Arrays

While matrices are confined to two dimensions, arrays can be of any number of dimensions. The array function takes a dim attribute which creates the required number of dimension. In the below example we create an array with two elements which are 3x3 matrices each.

```
# Create an array.  
a <- array(c('green','yellow'),dim = c(3,3,2))  
print(a)
```

When we execute the above code, it produces the following result –

```
,, 1  
  
 [,1] [,2] [,3]  
[1,] "green" "yellow" "green"  
[2,] "yellow" "green" "yellow"  
[3,] "green" "yellow" "green"
```

```
,, 2  
[1] [2] [3]  
[1] "yellow" "green" "yellow"  
[2] "green" "yellow" "green"  
[3] "yellow" "green" "yellow"
```

Factors

Factors are the r-objects which are created using a vector. It stores the vector along with the distinct values of the elements in the vector as labels. The labels are always character irrespective of whether it is numeric or character or Boolean etc. in the input vector. They are useful in statistical modeling.

Factors are created using the **factor** function. The **nlevels** functions gives the count of levels.

```
# Create a vector.  
apple_colors <- c('green','green','yellow','red','red','red','green')  
  
# Create a factor object.  
factor_apple <- factor(apple_colors)  
  
# Print the factor.  
print(factor_apple)  
print(nlevels(factor_apple))
```

When we execute the above code, it produces the following result –

```
[1] green green yellow red red red green  
Levels: green red yellow  
[1] 3
```

Data Frames

Data frames are tabular data objects. Unlike a matrix in data frame each column can contain different modes of data. The first column can be numeric while the second column can be character and third column can be logical. It is a list of vectors of equal length.

Data Frames are created using the **data.frame** function.

```
# Create the data frame.  
BMI <- data.frame(  
  gender = c("Male","Male","Female"),  
  height = c(152,171.5,165),  
  weight = c(81,93,78),  
  Age= c(42,38,26)  
)  
print(BMI)
```

When we execute the above code, it produces the following result –

```
gender height weight Age  
1 Male 152.0 81 42  
2 Male 171.5 93 38  
3 Female 165.0 78 26
```

R - VARIABLES

A variable provides us with named storage that our programs can manipulate. A variable in R can store an atomic vector, group of atomic vectors or a combination of many R objects. A valid variable name consists of letters, numbers and the dot or underscore characters. The variable name starts with a letter or the dot not followed by a number.

Variable Name	Validity	Reason
var_name2.	valid	Has letters, numbers, dot and underscore
var_name%	Invalid	Has the character '%'. Only dot.. and underscore allowed.
2var_name	invalid	Starts with a number

.var_name,	valid	Can start with a dot.. but the dot..should not be followed by a number.
var.name		
.2var_name	invalid	The starting dot is followed by a number making it invalid.
_var_name	invalid	Starts with _ which is not valid

Variable Assignment

The variables can be assigned values using leftward, rightward and equal to operator. The values of the variables can be printed using **print** or **cat** function. The **cat** function combines multiple items into a continuous print output.

```
# Assignment using equal operator.
```

```
var.1= c(0,1,2,3)
```

```
# Assignment using leftward operator.
```

```
var.2<- c("learn","R")
```

```
# Assignment using rightward operator.
```

```
c(TRUE,1)>var.3
```

```
print(var.1)
```

```
cat ("var.1 is ",var.1,"\n")
```

```
cat ("var.2 is ",var.2,"\n")
```

```
cat ("var.3 is ",var.3,"\n")
```

When we execute the above code, it produces the following result –

```
[1] 0 1 2 3
var.1 is 0 1 2 3
var.2 is learn R
var.3 is 1 1
```

Note – The vector cTRUE,1TRUE,1 has a mix of logical and numeric class. So logical class is coerced to numeric class making TRUE as 1.

Data Type of a Variable

In R, a variable itself is not declared of any data type, rather it gets the data type of the R - object assigned to it. So R is called a dynamically typed language, which means that we can change a variable's data type of the same variable again and again when using it in a program.

```
var_x <- "Hello"  
cat("The class of var_x is ",class(var_x),"\n")
```

```
var_x <- 34.5  
cat(" Now the class of var_x is ",class(var_x),"\n")
```

```
var_x <- 27L  
cat(" Next the class of var_x becomes ",class(var_x),"\n")
```

When we execute the above code, it produces the following result –

The class of var_x is character
Now the class of var_x is numeric
Next the class of var_x becomes integer

Finding Variables

To know all the variables currently available in the workspace we use the **ls** function. Also the **ls** function can use patterns to match the variable names.

```
print(ls())
```

When we execute the above code, it produces the following result –

```
[1] "my.var"   "my_new_var" "my_var"   "var.1"  
[5] "var.2"    "var.3"     "var.name"  "var_name2."  
[9] "var_x"    "varname"
```

Note – It is a sample output depending on what variables are declared in your environment.

The **ls** function can use patterns to match the variable names.

```
# List the variables starting with the pattern "var".  
print(ls(pattern ="var"))
```

When we execute the above code, it produces the following result –

```
[1] "my.var"    "my_new_var" "my_var"    "var.1"  
[5] "var.2"     "var.3"     "var.name"   "var_name2."  
[9] "var_x"     "varname"
```

The variables starting with **dot..** are hidden, they can be listed using "all.names = TRUE" argument to ls function.

```
print(ls(all.names = TRUE))
```

When we execute the above code, it produces the following result –

```
[1] ".cars"      ".Random.seed" ".var_name"    ".varname"    ".varname2"  
[6] "my.var"     "my_new_var"  "my_var"     "var.1"      "var.2"  
[11]"var.3"     "var.name"   "var_name2."  "var_x"
```

Deleting Variables

Variables can be deleted by using the **rm** function. Below we delete the variable var.3. On printing the value of the variable error is thrown.

```
rm(var.3)  
print(var.3)
```

When we execute the above code, it produces the following result –

```
[1] "var.3"  
Error in print(var.3) : object 'var.3' not found
```

All the variables can be deleted by using the **rm** and **ls** function together.

```
rm(list = ls())  
print(ls())
```

When we execute the above code, it produces the following result –

character(0)

R - OPERATORS

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. R language is rich in built-in operators and provides following types of operators.

Types of Operators

We have the following types of operators in R programming –

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Assignment Operators
- Miscellaneous Operators

Arithmetic Operators

Following table shows the arithmetic operators supported by R language. The operators act on each element of the vector.

Operator	Description	Example
+	Adds two vectors	<pre>v <- c(2,5,5,6) t <- c(8,3,4) print(v+t)</pre> <p>it produces the following result –</p> <pre>[1] 10.0 8.5 10.0</pre>
-	Subtracts second vector from the first	<pre>v <- c(2,5,5,6) t <- c(8,3,4) print(v-t)</pre>

		it produces the following result – [1] -6.0 2.5 2.0
*	Multiplies both vectors	<pre>v <- c(2,5,5,6) t <- c(8,3,4) print(v*t)</pre> it produces the following result – [1] 16.0 16.5 24.0
/	Divide the first vector with the second	<pre>v <- c(2,5,5,6) t <- c(8,3,4) print(v/t)</pre> When we execute the above code, it produces the following result – [1] 0.250000 1.833333 1.500000
%%	Give the remainder of the first vector with the second	<pre>v <- c(2,5,5,6) t <- c(8,3,4) print(v%%t)</pre> it produces the following result – [1] 2.0 2.5 2.0
/%	The result of division of first vector with second quotient	<pre>v <- c(2,5,5,6) t <- c(8,3,4) print(v%/%t)</pre> it produces the following result –

		[1] 0 1 1
^	The first vector raised to the exponent of second vector	<pre>v <- c(2,5,5,6) t <- c(8,3,4) print(v^t)</pre> <p>it produces the following result –</p> <pre>[1] 256.000 166.375 1296.000</pre>

Relational Operators

Following table shows the relational operators supported by R language. Each element of the first vector is compared with the corresponding element of the second vector. The result of comparison is a Boolean value.

Operator	Description	Example
>	<p>Checks if each element of the first vector is greater than the corresponding element of the second vector.</p>	<pre>v <- c(2,5,5,6,9) t <- c(8,2,5,14,9) print(v>t)</pre> <p>it produces the following result –</p> <pre>[1] FALSE TRUE FALSE FALSE</pre>
<	<p>Checks if each element of the first vector is less than the corresponding element of the second vector.</p>	<pre>v <- c(2,5,5,6,9) t <- c(8,2,5,14,9) print(v < t)</pre> <p>it produces the following result –</p> <pre>[1] TRUE FALSE TRUE FALSE</pre>

==	Checks if each element of the first vector is equal to the corresponding element of the second vector.	<pre>v <- c(2,5,5,6,9) t <- c(8,2,5,14,9) print(v == t)</pre> <p>it produces the following result –</p> <pre>[1] FALSE FALSE FALSE TRUE</pre>
<=	Checks if each element of the first vector is less than or equal to the corresponding element of the second vector.	<pre>v <- c(2,5,5,6,9) t <- c(8,2,5,14,9) print(v <= t)</pre> <p>it produces the following result –</p> <pre>[1] TRUE FALSE TRUE TRUE</pre>
>=	Checks if each element of the first vector is greater than or equal to the corresponding element of the second vector.	<pre>v <- c(2,5,5,6,9) t <- c(8,2,5,14,9) print(v >= t)</pre> <p>it produces the following result –</p> <pre>[1] FALSE TRUE FALSE TRUE</pre>
!=	Checks if each element of the first vector is unequal to the corresponding element of the second vector.	<pre>v <- c(2,5,5,6,9) t <- c(8,2,5,14,9) print(v != t)</pre> <p>it produces the following result –</p> <pre>[1] TRUE TRUE TRUE FALSE</pre>

Logical Operators

Following table shows the logical operators supported by R language. It is applicable only to vectors of type logical, numeric or complex. All numbers greater than 1 are considered as logical value TRUE.

Each element of the first vector is compared with the corresponding element of the second vector. The result of comparison is a Boolean value.

Operator	Description	Example
&	<p>It is called Element-wise Logical AND operator. It combines each element of the first vector with the corresponding element of the second vector and gives a output TRUE if both the elements are TRUE.</p>	<pre>v <- c(3,1,TRUE,2+3i) t <- c(4,1, FALSE,2+3i) print(v&t)</pre> <p>it produces the following result –</p> <pre>[1] TRUE TRUE FALSE TRUE</pre>
	<p>It is called Element-wise Logical OR operator. It combines each element of the first vector with the corresponding element of the second vector and gives a output TRUE if one the elements is TRUE.</p>	<pre>v <- c(3,0,TRUE,2+2i) t <- c(4,0, FALSE,2+3i) print(v t)</pre> <p>it produces the following result –</p> <pre>[1] TRUE FALSE TRUE TRUE</pre>
!	<p>It is called Logical NOT operator. Takes each element of the vector and gives the opposite logical value.</p>	<pre>v <- c(3,0,TRUE,2+2i) print(!v)</pre> <p>it produces the following result –</p> <pre>[1] FALSE TRUE FALSE FALSE</pre>

The logical operator `&&` and `||` considers only the first element of the vectors and give a vector of single element as output.

Operator	Description	Example
<code>&&</code>	Called Logical AND operator. Takes first element of both the vectors and gives the TRUE only if both are TRUE.	<pre>v <- c(3,0,TRUE,2+2i) t <- c(1,3,TRUE,2+3i) print(v&&t)</pre> <p>it produces the following result –</p> <pre>[1] TRUE</pre>
<code> </code>	Called Logical OR operator. Takes first element of both the vectors and gives the TRUE if one of them is TRUE.	<pre>v <- c(0,0,TRUE,2+2i) t <- c(0,3,TRUE,2+3i) print(v t)</pre> <p>it produces the following result –</p> <pre>[1] FALSE</pre>

Assignment Operators

These operators are used to assign values to vectors.

Operator	Description	Example

	Called Left Assignment	<pre>v1 <- c(3,1,TRUE,2+3i) v2 <<- c(3,1,TRUE,2+3i) v3 = c(3,1,TRUE,2+3i) print(v1) print(v2) print(v3)</pre>
<- or = or <<-		it produces the following result – [1] 3+0i 1+0i 1+0i 2+3i [1] 3+0i 1+0i 1+0i 2+3i [1] 3+0i 1+0i 1+0i 2+3i

	Called Right Assignment	<pre>c(3,1,TRUE,2+3i)-> v1 c(3,1,TRUE,2+3i)->> v2 print(v1) print(v2)</pre>
-> or ->>		it produces the following result – [1] 3+0i 1+0i 1+0i 2+3i [1] 3+0i 1+0i 1+0i 2+3i

Miscellaneous Operators

These operators are used to for specific purpose and not general mathematical or logical computation.

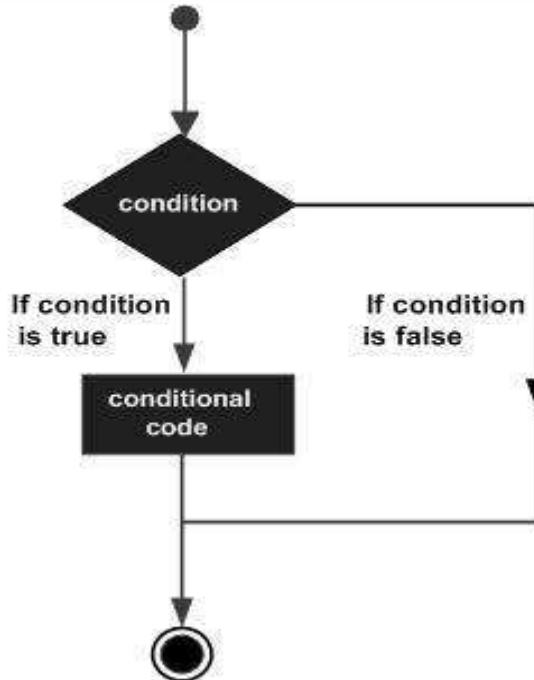
Operator	Description	Example
:	Colon operator. It creates the series of numbers in sequence for a vector.	<pre>v <- 2:8 print(v)</pre> <p>it produces the following result –</p> <pre>[1] 2 3 4 5 6 7 8</pre>
%in%	This operator is used to identify if an element belongs to a vector.	<pre>v1 <- 8 v2 <- 12 t <- 1:10 print(v1 %in% t) print(v2 %in% t)</pre> <p>it produces the following result –</p> <pre>[1] TRUE [1] FALSE</pre>
%%	This operator is used to multiply a matrix with its transpose.	<pre>M = matrix(c(2,6,5,1,10,4), nrow = 2, ncol = 3, byrow = TRUE) t = M %% t(M) print(t)</pre> <p>it produces the following result –</p> <pre>[,1] [,2] [1,] 65 82 [2,] 82 117</pre>

R - DECISION MAKING

Decision making structures require the programmer to specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be

executed if the condition is determined to be **true**, and optionally, other statements to be executed if the condition is determined to be **false**.

Following is the general form of a typical decision making structure found in most of the programming languages –



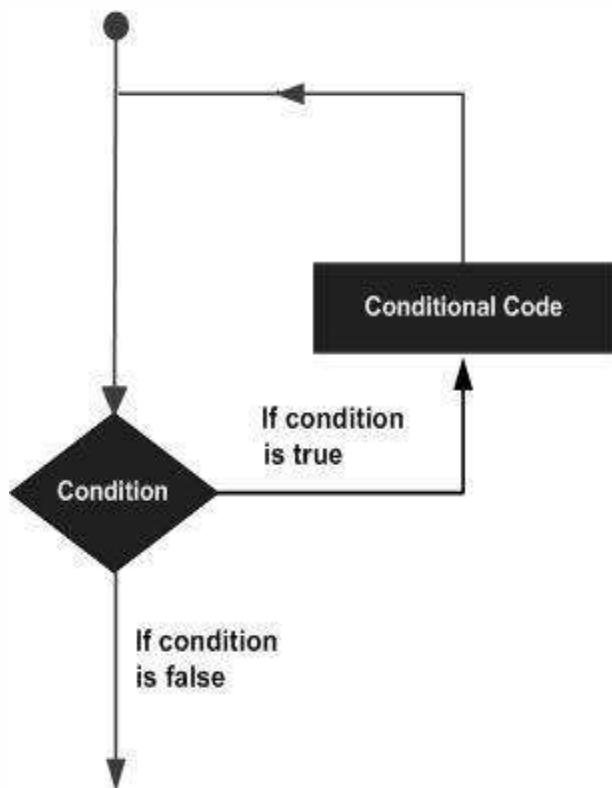
R provides the following types of decision making statements. Click the following links to check their detail.

Sr.No.	Statement & Description
1	if statement An if statement consists of a Boolean expression followed by one or more statements.
2	if...else statement An if statement can be followed by an optional else statement, which executes when the Boolean expression is false.
3	switch statement A switch statement allows a variable to be tested for equality against a list of values.

when you need to execute a block of code several number of times. In general, statements are executed sequentially. The first statement in a function is executed first, followed by the second, and so on.

Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times and the following is the general form of a loop statement in most of the programming languages –



R programming language provides the following kinds of loop to handle looping requirements. Click the following links to check their detail.

Sr.No.	Loop Type & Description

1	<u>repeat loop</u> Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
2	<u>while loop</u> Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
3	<u>for loop</u> Like a while statement, except that it tests the condition at the end of the loop body.

Loop Control Statements

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

R supports the following control statements. Click the following links to check their detail.

Sr.No.	Control Statement & Description
1	<u>break statement</u> Terminates the loop statement and transfers execution to the statement immediately following the loop.
2	<u>Next statement</u> The next statement simulates the behavior of R switch.

R - FUNCTIONS

A function is a set of statements organized together to perform a specific task. R has a large number of in-built functions and the user can create their own functions.

In R, a function is an object so the R interpreter is able to pass control to the function, along with arguments that may be necessary for the function to accomplish the actions.

The function in turn performs its task and returns control to the interpreter as well as any result which may be stored in other objects.

Function Definition

An R function is created by using the keyword **function**. The basic syntax of an R function definition is as follows –

```
function_name <- function(arg_1, arg_2, ...) {  
  Function body  
}
```

Function Components

The different parts of a function are –

- **Function Name** – This is the actual name of the function. It is stored in R environment as an object with this name.
- **Arguments** – An argument is a placeholder. When a function is invoked, you pass a value to the argument. Arguments are optional; that is, a function may contain no arguments. Also arguments can have default values.
- **Function Body** – The function body contains a collection of statements that defines what the function does.
- **Return Value** – The return value of a function is the last expression in the function body to be evaluated.

R has many **in-built** functions which can be directly called in the program without defining them first. We can also create and use our own functions referred as **user defined** functions.

Built-in Function

Simple examples of in-built functions are **seq**, **mean**, **max**, **sum** and **paste**..... etc. They are directly called by user written programs. You can refer [most widely used R functions](#).

```
# Create a sequence of numbers from 32 to 44.
```

```
print(seq(32,44))
```

```
# Find mean of numbers from 25 to 82.
```

```
print(mean(25:82))
```

```
# Find sum of numbers frm 41 to 68.
```

```
print(sum(41:68))
```

When we execute the above code, it produces the following result –

```
[1] 32 33 34 35 36 37 38 39 40 41 42 43 44  
[1] 53.5  
[1] 1526
```

User-defined Function

We can create user-defined functions in R. They are specific to what a user wants and once created they can be used like the built-in functions. Below is an example of how a function is created and used.

```
# Create a function to print squares of numbers in sequence.
```

```
new.function<-function(a){  
  for(i in 1:a){  
    b <- i^2  
    print(b)  
  }  
}
```

Calling a Function

```
# Create a function to print squares of numbers in sequence.
```

```
new.function<-function(a){  
  for(i in 1:a){  
    b <- i^2  
    print(b)  
  }  
}
```

```
# Call the function new.function supplying 6 as an argument.
```

```
new.function(6)
```

When we execute the above code, it produces the following result –

```
[1] 1  
[1] 4  
[1] 9  
[1] 16  
[1] 25  
[1] 36
```

Calling a Function without an Argument

```
# Create a function without an argument.
```

```
new.function<-function(){  
for(i in 1:5){  
print(i^2)  
}  
}
```

```
# Call the function without supplying an argument.
```

```
new.function()
```

When we execute the above code, it produces the following result –

```
[1] 1  
[1] 4  
[1] 9  
[1] 16  
[1] 25
```

Calling a Function with Argument Values by position and by name by position and byname

The arguments to a function call can be supplied in the same sequence as defined in the function or they can be supplied in a different sequence but assigned to the names of the arguments.

```
# Create a function with arguments.  
new.function<-function(a,b,c){  
  result <- a * b + c  
  print(result)  
}  
  
# Call the function by position of arguments.  
new.function(5,3,11)  
  
# Call the function by names of the arguments.  
new.function(a =11, b =5, c =3)
```

When we execute the above code, it produces the following result –

```
[1] 26  
[1] 58
```

Calling a Function with Default Argument

We can define the value of the arguments in the function definition and call the function without supplying any argument to get the default result. But we can also call such functions by supplying new values of the argument and get non default result.

```
# Create a function with arguments.  
new.function<-function(a =3, b =6){  
  result <- a * b  
  print(result)  
}  
  
# Call the function without giving any argument.  
new.function()
```

```
# Call the function with giving new values of the argument.
```

```
new.function(9,5)
```

When we execute the above code, it produces the following result –

```
[1] 18  
[1] 45
```

Lazy Evaluation of Function

Arguments to functions are evaluated lazily, which means so they are evaluated only when needed by the function body.

```
# Create a function with arguments.
```

```
new.function<-function(a, b){  
  print(a^2)  
  print(a)  
  print(b)  
}
```

```
# Evaluate the function without supplying one of the arguments.
```

```
new.function(6)
```

When we execute the above code, it produces the following result –

```
[1] 36  
[1] 6  
Error in print(b) : argument "b" is missing, with no default  
R - STRINGS
```

Any value written within a pair of single quote or double quotes in R is treated as a string. Internally R stores every string within double quotes, even when you create them with single quote.

Rules Applied in String Construction

- The quotes at the beginning and end of a string should be both double quotes or both single quote. They cannot be mixed.
- Double quotes can be inserted into a string starting and ending with single quote.
- Single quote can be inserted into a string starting and ending with double quotes.
- Double quotes cannot be inserted into a string starting and ending with double quotes.
- Single quote cannot be inserted into a string starting and ending with single quote.

Examples of Valid Strings

Following examples clarify the rules about creating a string in R.

a <-'Start and end with single quote'

```
print(a)
```

b <-"Start and end with double quotes"

```
print(b)
```

c <-"single quote ' in between double quotes"

```
print(c)
```

d <-'Double quotes " in between single quote'

```
print(d)
```

When the above code is run we get the following output –

```
[1] "Start and end with single quote"  
[1] "Start and end with double quotes"  
[1] "single quote ' in between double quote"  
[1] "Double quote \" in between single quote"
```

Examples of Invalid Strings

e <-'Mixed quotes'

```
print(e)
```

```
f <- 'Single quote ' inside single quote'
```

```
print(f)
```

```
g <-"Double quotes " inside double quotes"
```

```
print(g)
```

When we run the script it fails giving below results.

```
Error: unexpected symbol in:  
"print(e)  
f <- 'Single'"  
Execution halted
```

String Manipulation

Concatenating Strings - paste function

Many strings in R are combined using the **paste** function. It can take any number of arguments to be combined together.

Syntax

The basic syntax for paste function is –

```
paste(..., sep = "", collapse = NULL)
```

Following is the description of the parameters used –

- ... represents any number of arguments to be combined.
- **sep** represents any separator between the arguments. It is optional.
- **collapse** is used to eliminate the space in between two strings. But not the space within two words of one string.

Example

```
a <-"Hello"  
b <-'How'  
c <-"are you? "  
print(paste(a,b,c))
```

```
print(paste(a,b,c, sep ="-"))

print(paste(a,b,c, sep = "", collapse = ""))
```

When we execute the above code, it produces the following result –

```
[1] "Hello How are you? "
[1] "Hello-How-are you? "
[1] "HelloHoware you? "
```

Formatting numbers & strings - format function

Numbers and strings can be formatted to a specific style using **format** function.

Syntax

The basic syntax for format function is –

```
format(x, digits, nsmall, scientific, width, justify = c("left", "right", "centre", "none"))
```

Following is the description of the parameters used –

- **x** is the vector input.
- **digits** is the total number of digits displayed.
- **nsmall** is the minimum number of digits to the right of the decimal point.
- **scientific** is set to TRUE to display scientific notation.
- **width** indicates the minimum width to be displayed by padding blanks in the beginning.
- **justify** is the display of the string to left, right or center.

Example

```
# Total number of digits displayed. Last digit rounded off.
```

```
result <- format(23.123456789, digits =9)
print(result)
```

```
# Display numbers in scientific notation.
```

```
result <- format(c(6,13.14521), scientific = TRUE)
```

```
print(result)

# The minimum number of digits to the right of the decimal point.

result <- format(23.47, nsmall =5)
print(result)

# Format treats everything as a string.

result <- format(6)
print(result)

# Numbers are padded with blank in the beginning for width.

result <- format(13.7, width =6)
print(result)

# Left justify strings.

result <- format("Hello", width =8, justify ="l")
print(result)

# Justify string with center.

result <- format("Hello", width =8, justify ="c")
print(result)
```

When we execute the above code, it produces the following result –

```
[1] "23.1234568"
[1] "6.000000e+00" "1.314521e+01"
[1] "23.47000"
[1] "6"
[1] " 13.7"
[1] "Hello  "
[1] "Hello  "
```

Counting number of characters in a string - nchar function

This function counts the number of characters including spaces in a string.

Syntax

The basic syntax for nchar function is –

```
nchar(x)
```

Following is the description of the parameters used –

- **x** is the vector input.

Example

```
result <- nchar("Count the number of characters")
print(result)
```

When we execute the above code, it produces the following result –

```
[1] 30
```

Changing the case - toupper & tolower functions

These functions change the case of characters of a string.

Syntax

The basic syntax for toupper & tolower function is –

```
toupper(x)
tolower(x)
```

Following is the description of the parameters used –

- **x** is the vector input.

Example

```
# Changing to Upper case.
result <- toupper("Changing To Upper")
print(result)

# Changing to lower case.
```

```
result <- tolower("Changing To Lower")
print(result)
```

When we execute the above code, it produces the following result –

```
[1] "CHANGING TO UPPER"
[1] "changing to lower"
```

Extracting parts of a string - substring function

This function extracts parts of a String.

Syntax

The basic syntax for substring function is –

```
substring(x,first,last)
```

Following is the description of the parameters used –

- **x** is the character vector input.
- **first** is the position of the first character to be extracted.
- **last** is the position of the last character to be extracted.

Example

```
# Extract characters from 5th to 7th position.
result <- substring("Extract",5,7)
print(result)
```

When we execute the above code, it produces the following result –

```
[1] "act"
```

R - VECTORS

Vectors are the most basic R data objects and there are six types of atomic vectors. They are logical, integer, double, complex, character and raw.

Vector Creation

Single Element Vector

Even when you write just one value in R, it becomes a vector of length 1 and belongs to one of the above vector types.

```
# Atomic vector of type character.  
print("abc");  
  
# Atomic vector of type double.  
print(12.5)  
  
# Atomic vector of type integer.  
print(63L)  
  
# Atomic vector of type logical.  
print(TRUE)  
  
# Atomic vector of type complex.  
print(2+3i)  
  
# Atomic vector of type raw.  
print(charToRaw('hello'))
```

When we execute the above code, it produces the following result –

```
[1] "abc"  
[1] 12.5  
[1] 63  
[1] TRUE  
[1] 2+3i  
[1] 68 65 6c 6c 6f
```

Multiple Elements Vector

Using colon operator with numeric data

```
# Creating a sequence from 5 to 13.
```

```
v <- 5:13
```

```
print(v)
```

```
# Creating a sequence from 6.6 to 12.6.
```

```
v <- 6.6:12.6
```

```
print(v)
```

```
# If the final element specified does not belong to the sequence then it is discarded.
```

```
v <- 3.8:11.4
```

```
print(v)
```

When we execute the above code, it produces the following result –

```
[1] 5 6 7 8 9 10 11 12 13  
[1] 6.6 7.6 8.6 9.6 10.6 11.6 12.6  
[1] 3.8 4.8 5.8 6.8 7.8 8.8 9.8 10.8
```

Using sequence Seq.Seq. operator

```
# Create vector with elements from 5 to 9 incrementing by 0.4.
```

```
print(seq(5,9,by=0.4))
```

When we execute the above code, it produces the following result –

```
[1] 5.0 5.4 5.8 6.2 6.6 7.0 7.4 7.8 8.2 8.6 9.0
```

Using the c function

The non-character values are coerced to character type if one of the elements is a character.

```
# The logical and numeric values are converted to characters.
```

```
s <- c('apple','red',5,TRUE)
print(s)
```

When we execute the above code, it produces the following result –

```
[1] "apple" "red"  "5"    "TRUE"
```

Accessing Vector Elements

Elements of a Vector are accessed using indexing. The **[] brackets** are used for indexing. Indexing starts with position 1. Giving a negative value in the index drops that element from result. **TRUE, FALSE** or **0** and **1** can also be used for indexing.

```
# Accessing vector elements using position.
t <- c("Sun","Mon","Tue","Wed","Thurs","Fri","Sat")
u <- t[c(2,3,6)]
print(u)

# Accessing vector elements using logical indexing.
v <- t[c(TRUE,FALSE,FALSE,TRUE,FALSE,TRUE,FALSE)]
print(v)

# Accessing vector elements using negative indexing.
x <- t[c(-2,-5)]
print(x)

# Accessing vector elements using 0/1 indexing.
y <- t[c(0,0,0,0,0,1)]
print(y)
```

When we execute the above code, it produces the following result –

```
[1] "Mon" "Tue" "Fri"
[1] "Sun" "Fri"
[1] "Sun" "Tue" "Wed" "Fri" "Sat"
```

```
[1] "Sun"
```

Vector Manipulation

Vector arithmetic

Two vectors of same length can be added, subtracted, multiplied or divided giving the result as a vector output.

```
# Create two vectors.
```

```
v1 <- c(3,8,4,5,0,11)  
v2 <- c(4,11,0,8,1,2)
```

```
# Vector addition.
```

```
add.result <- v1+v2  
print(add.result)
```

```
# Vector subtraction.
```

```
sub.result <- v1-v2  
print(sub.result)
```

```
# Vector multiplication.
```

```
multi.result <- v1*v2  
print(multi.result)
```

```
# Vector division.
```

```
divi.result <- v1/v2  
print(divi.result)
```

When we execute the above code, it produces the following result –

```
[1] 7 19 4 13 1 13  
[1] -1 -3 4 -3 -1 9  
[1] 12 88 0 40 0 22
```

```
[1] 0.7500000 0.7272727 Inf 0.6250000 0.0000000 5.5000000
```

Vector Element Recycling

If we apply arithmetic operations to two vectors of unequal length, then the elements of the shorter vector are recycled to complete the operations.

```
v1 <- c(3,8,4,5,0,11)
v2 <- c(4,11)
# V2 becomes c(4,11,4,11,4,11)

add.result <- v1+v2
print(add.result)

sub.result <- v1-v2
print(sub.result)
```

When we execute the above code, it produces the following result –

```
[1] 7 19 8 16 4 22
[1] -1 -3 0 -6 -4 0
```

Vector Element Sorting

Elements in a vector can be sorted using the **sort** function.

```
v <- c(3,8,4,5,0,11,-9,304)
# Sort the elements of the vector.
sort.result <- sort(v)
print(sort.result)

# Sort the elements in the reverse order.
revsort.result <- sort(v, decreasing = TRUE)
print(revsort.result)

# Sorting character vectors.
```

```
v <- c("Red","Blue","yellow","violet")  
sort.result <- sort(v)  
print(sort.result)  
  
# Sorting character vectors in reverse order.  
revsort.result <- sort(v, decreasing = TRUE)  
print(revsort.result)
```

When we execute the above code, it produces the following result –

```
[1] -9  0  3  4  5  8 11 304  
[1] 304 11  8  5  4  3  0 -9  
[1] "Blue" "Red" "violet" "yellow"  
[1] "yellow" "violet" "Red"  "Blue"
```

R - LISTS

Lists are the R objects which contain elements of different types like – numbers, strings, vectors and another list inside it. A list can also contain a matrix or a function as its elements. List is created using **list** function.

Creating a List

Following is an example to create a list containing strings, numbers, vectors and a logical values.

```
# Create a list containing strings, numbers, vectors and a logical  
# values.  
list_data <- list("Red","Green", c(21,32,11), TRUE, 51.23, 119.1)  
print(list_data)
```

When we execute the above code, it produces the following result –

```
[[1]]  
[1] "Red"  
  
[[2]]  
[1] "Green"
```

```
[[3]]  
[1] 21 32 11
```

```
[[4]]  
[1] TRUE
```

```
[[5]]  
[1] 51.23
```

```
[[6]]  
[1] 119.1
```

Naming List Elements

The list elements can be given names and they can be accessed using these names.

```
# Create a list containing a vector, a matrix and a list.  
list_data <- list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8), nrow =2),  
list("green",12.3))  
  
# Give names to the elements in the list.  
names(list_data)<- c("1st Quarter","A_Matrix","A Inner list")  
  
# Show the list.  
print(list_data)
```

When we execute the above code, it produces the following result –

```
$`1st_Quarter`  
[1] "Jan" "Feb" "Mar"  
  
$A_Matrix  
[1,] [,1] [,2] [,3]  
[1,] 3 5 -2  
[2,] 9 1 8  
  
$A_Inner_list  
$A_Inner_list[[1]]  
[1] "green"
```

```
$A_Inner_list[[2]]  
[1] 12.3
```

Accessing List Elements

Elements of the list can be accessed by the index of the element in the list. In case of named lists it can also be accessed using the names.

We continue to use the list in the above example –

```
# Create a list containing a vector, a matrix and a list.  
list_data <- list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8), nrow =2),  
list("green",12.3))  
  
# Give names to the elements in the list.  
names(list_data)<- c("1st Quarter","A_Matrix","A_Inner list")  
  
# Access the first element of the list.  
print(list_data[1])  
  
# Access the third element. As it is also a list, all its elements will be printed.  
print(list_data[3])  
  
# Access the list element using the name of the element.  
print(list_data$A_Matrix)
```

When we execute the above code, it produces the following result –

```
$`1st_Quarter`  
[1] "Jan" "Feb" "Mar"  
  
$A_Inner_list
```

```
$A_Inner_list[[1]]  
[1] "green"
```

```
$A_Inner_list[[2]]  
[1] 12.3
```

```
[,1] [,2] [,3]  
[1,] 3 5 -2  
[2,] 9 1 8
```

Manipulating List Elements

We can add, delete and update list elements as shown below. We can add and delete elements only at the end of a list. But we can update any element.

```
# Create a list containing a vector, a matrix and a list.
```

```
list_data <- list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8), nrow =2),  
list("green",12.3))
```

```
# Give names to the elements in the list.
```

```
names(list_data)<- c("1st Quarter","A_Matrix","A Inner list")
```

```
# Add element at the end of the list.
```

```
list_data[4]<-"New element"  
print(list_data[4])
```

```
# Remove the last element.
```

```
list_data[4]<- NULL
```

```
# Print the 4th Element.
```

```
print(list_data[4])
```

```
# Update the 3rd Element.
```

```
list_data[3]<-"updated element"  
print(list_data[3])
```

When we execute the above code, it produces the following result –

```
[[1]]  
[1] "New element"  
  
$<NA>  
NULL  
  
$`A Inner list`  
[1] "updated element"
```

Merging Lists

You can merge many lists into one list by placing all the lists inside one list function.

```
# Create two lists.  
  
list1 <- list(1,2,3)  
  
list2 <- list("Sun","Mon","Tue")  
  
# Merge the two lists.  
  
merged.list <- c(list1,list2)  
  
# Print the merged list.  
  
print(merged.list)
```

When we execute the above code, it produces the following result –

```
[[1]]  
[1] 1  
  
[[2]]  
[1] 2  
  
[[3]]  
[1] 3  
[[4]]  
[1] "Sun"  
  
[[5]]  
[1] "Mon"
```

```
[[6]]  
[1] "Tue"
```

Converting List to Vector

A list can be converted to a vector so that the elements of the vector can be used for further manipulation. All the arithmetic operations on vectors can be applied after the list is converted into vectors. To do this conversion, we use the **unlist** function. It takes the list as input and produces a vector.

```
# Create lists.  
  
list1 <- list(1:5)  
print(list1)  
  
list2 <- list(10:14)  
print(list2)  
  
# Convert the lists to vectors.  
  
v1 <- unlist(list1)  
v2 <- unlist(list2)  
  
print(v1)  
print(v2)  
  
# Now add the vectors  
  
result <- v1+v2  
print(result)
```

When we execute the above code, it produces the following result –

```
[[1]]  
[1] 1 2 3 4 5  
  
[[1]]
```

```
[1] 10 11 12 13 14
```

```
[1] 1 2 3 4 5
```

```
[1] 10 11 12 13 14
```

```
[1] 11 13 15 17 19
```

R - MATRICES

Matrices are the R objects in which the elements are arranged in a two-dimensional rectangular layout. They contain elements of the same atomic types. Though we can create a matrix containing only characters or only logical values, they are not of much use. We use matrices containing numeric elements to be used in mathematical calculations.

A Matrix is created using the **matrix** function.

Syntax

The basic syntax for creating a matrix in R is –

```
matrix(data, nrow, ncol, byrow, dimnames)
```

Following is the description of the parameters used –

- **data** is the input vector which becomes the data elements of the matrix.
- **nrow** is the number of rows to be created.
- **ncol** is the number of columns to be created.
- **byrow** is a logical clue. If TRUE then the input vector elements are arranged by row.
- **dimname** is the names assigned to the rows and columns.

Example

Create a matrix taking a vector of numbers as input.

```
# Elements are arranged sequentially by row.  
M <- matrix(c(3:14), nrow =4, byrow = TRUE)  
print(M)  
  
# Elements are arranged sequentially by column.  
N <- matrix(c(3:14), nrow =4, byrow = FALSE)  
print(N)
```

```
# Define the column and row names.  
rownames = c("row1","row2","row3","row4")  
colnames = c("col1","col2","col3")  
  
P <- matrix(c(3:14), nrow =4, byrow = TRUE, dimnames = list(rownames, colnames))  
print(P)
```

When we execute the above code, it produces the following result –

```
[,1] [,2] [,3]  
[1,] 3 4 5  
[2,] 6 7 8  
[3,] 9 10 11  
[4,] 12 13 14  
[,1] [,2] [,3]  
[1,] 3 7 11  
[2,] 4 8 12  
[3,] 5 9 13  
[4,] 6 10 14  
col1 col2 col3  
row1 3 4 5  
row2 6 7 8  
row3 9 10 11  
row4 12 13 14
```

Ameerpet / Kondapur

Accessing Elements of a Matrix

Elements of a matrix can be accessed by using the column and row index of the element. We consider the matrix P above to find the specific elements below.

```
# Define the column and row names.  
rownames = c("row1","row2","row3","row4")  
colnames = c("col1","col2","col3")  
  
# Create the matrix.  
P <- matrix(c(3:14), nrow =4, byrow = TRUE, dimnames = list(rownames, colnames))  
  
# Access the element at 3rd column and 1st row.  
print(P[1,3])
```

```
# Access the element at 2nd column and 4th row.
```

```
print(P[4,2])
```

```
# Access only the 2nd row.
```

```
print(P[2,])
```

```
# Access only the 3rd column.
```

```
print(P[,3])
```

When we execute the above code, it produces the following result –

```
[1] 5
[1] 13
col1 col2 col3
 6  7  8
row1 row2 row3 row4
 5  8 11 14
```

Matrix Computations

Various mathematical operations are performed on the matrices using the R operators. The result of the operation is also a matrix.

The dimensions number of rows and columns should be same for the matrices involved in the operation.

Matrix Addition & Subtraction

```
# Create two 2x3 matrices.
```

```
matrix1 <- matrix(c(3,9,-1,4,2,6), nrow =2)
```

```
print(matrix1)
```

```
matrix2 <- matrix(c(5,2,0,9,3,4), nrow =2)
```

```
print(matrix2)
```

```
# Add the matrices.
```

```
result <- matrix1 + matrix2
cat("Result of addition","\n")
print(result)

# Subtract the matrices
result <- matrix1 - matrix2
cat("Result of subtraction","\n")
print(result)
```

When we execute the above code, it produces the following result –

```
[,1] [,2] [,3]
[1,] 3 -1 2
[2,] 9 4 6
[,1] [,2] [,3]
[1,] 5 0 3
[2,] 2 9 4
Result of addition
[,1] [,2] [,3]
[1,] 8 -1 5
[2,] 11 13 10
Result of subtraction
[,1] [,2] [,3]
[1,] -2 -1 -1
[2,] 7 -5 2
```

Matrix Multiplication & Division

```
# Create two 2x3 matrices.
matrix1 <- matrix(c(3,9,-1,4,2,6), nrow =2)
print(matrix1)

matrix2 <- matrix(c(5,2,0,9,3,4), nrow =2)
print(matrix2)

# Multiply the matrices.
result <- matrix1 * matrix2
```

```
cat("Result of multiplication","\n")
print(result)

# Divide the matrices
result <- matrix1 / matrix2
cat("Result of division","\n")
print(result)
```

When we execute the above code, it produces the following result –

```
[,1] [,2] [,3]
[1,] 3 -1 2
[2,] 9 4 6
[,1] [,2] [,3]
[1,] 5 0 3
[2,] 2 9 4
Result of multiplication
[,1] [,2] [,3]
[1,] 15 0 6
[2,] 18 36 24
Result of division
[,1] [,2] [,3]
[1,] 0.6 -Inf 0.6666667
[2,] 4.5 0.4444444 1.5000000
```

R - ARRAYS

Arrays are the R data objects which can store data in more than two dimensions. For example – If we create an array of dimension 2,3,4,2,3,4 then it creates 4 rectangular matrices each with 2 rows and 3 columns. Arrays can store only data type.

An array is created using the **array** function. It takes vectors as input and uses the values in the **dim** parameter to create an array.

Example

The following example creates an array of two 3x3 matrices each with 3 rows and 3 columns.

```
# Create two vectors of different lengths.
vector1 <- c(5,9,3)
```

```
vector2 <- c(10,11,12,13,14,15)

# Take these vectors as input to the array.

result <- array(c(vector1,vector2),dim = c(3,3,2))

print(result)
```

When we execute the above code, it produces the following result –

```
,, 1

[,1] [,2] [,3]
[1,] 5 10 13
[2,] 9 11 14
[3,] 3 12 15

,, 2

[,1] [,2] [,3]
[1,] 5 10 13
[2,] 9 11 14
[3,] 3 12 15
```

Naming Columns and Rows

We can give names to the rows, columns and matrices in the array by using the **dimnames** parameter.

```
# Create two vectors of different lengths.

vector1 <- c(5,9,3)

vector2 <- c(10,11,12,13,14,15)

column.names <- c("COL1","COL2","COL3")

row.names <- c("ROW1","ROW2","ROW3")

matrix.names <- c("Matrix1","Matrix2")

# Take these vectors as input to the array.

result <- array(c(vector1,vector2),dim = c(3,3,2),dimnames = list(row.names,column.names,
```

```
matrix.names))  
print(result)
```

When we execute the above code, it produces the following result –

```
, , Matrix1  
  
    COL1 COL2 COL3  
ROW1  5  10  13  
ROW2  9  11  14  
ROW3  3  12  15  
  
, , Matrix2  
  
    COL1 COL2 COL3  
ROW1  5  10  13  
ROW2  9  11  14  
ROW3  3  12  15
```

Accessing Array Elements

```
# Create two vectors of different lengths.  
  
vector1 <- c(5,9,3)  
vector2 <- c(10,11,12,13,14,15)  
column.names <- c("COL1","COL2","COL3")  
row.names <- c("ROW1","ROW2","ROW3")  
matrix.names <- c("Matrix1","Matrix2")  
  
# Take these vectors as input to the array.  
result <- array(c(vector1,vector2),dim = c(3,3,2),dimnames = list(row.names,  
    column.names, matrix.names))  
  
# Print the third row of the second matrix of the array.  
print(result[3,,2])  
  
# Print the element in the 1st row and 3rd column of the 1st matrix.
```

```
print(result[1,3,1])  
  
# Print the 2nd Matrix.  
print(result[,,2])
```

When we execute the above code, it produces the following result –

```
COL1 COL2 COL3  
3 12 15  
[1] 13  
COL1 COL2 COL3  
ROW1 5 10 13  
ROW2 9 11 14  
ROW3 3 12 15
```

Manipulating Array Elements

As array is made up matrices in multiple dimensions, the operations on elements of array are carried out by accessing elements of the matrices.

```
# Create two vectors of different lengths.  
vector1 <- c(5,9,3)  
vector2 <- c(10,11,12,13,14,15)  
  
# Take these vectors as input to the array.  
array1 <- array(c(vector1,vector2),dim = c(3,3,2))  
  
# Create two vectors of different lengths.  
vector3 <- c(9,1,0)  
vector4 <- c(6,0,11,3,14,1,2,6,9)  
array2 <- array(c(vector1,vector2),dim = c(3,3,2))  
  
# create matrices from these arrays.  
matrix1 <- array1[,,2]  
matrix2 <- array2[,,2]
```

```
# Add the matrices.  
result <- matrix1+matrix2  
print(result)
```

When we execute the above code, it produces the following result –

```
[,1] [,2] [,3]  
[1,] 10 20 26  
[2,] 18 22 28  
[3,] 6 24 30
```

Calculations Across Array Elements

We can do calculations across the elements in an array using the **apply** function.

Syntax

```
apply(x, margin, fun)
```

Following is the description of the parameters used –

- **x** is an array.
- **margin** is the name of the data set used.
- **fun** is the function to be applied across the elements of the array.

Example

We use the apply function below to calculate the sum of the elements in the rows of an array across all the matrices.

```
# Create two vectors of different lengths.  
vector1 <- c(5,9,3)  
vector2 <- c(10,11,12,13,14,15)  
  
# Take these vectors as input to the array.  
new.array <- array(c(vector1, vector2), dim = c(3,3,2))  
print(new.array)
```

```
# Use apply to calculate the sum of the rows across all the matrices.
```

```
result <- apply(new.array, c(1), sum)  
print(result)
```

When we execute the above code, it produces the following result –

```
,, 1  
[.1] [.2] [.3]  
[1,] 5 10 13  
[2,] 9 11 14  
[3,] 3 12 15
```

```
,, 2  
[.1] [.2] [.3]  
[1,] 5 10 13  
[2,] 9 11 14  
[3,] 3 12 15
```

```
[1] 56 68 60
```

R - FACTORS



Factors are the data objects which are used to categorize the data and store it as levels. They can store both strings and integers. They are useful in the columns which have a limited number of unique values. Like "Male", "Female" and True, False etc. They are useful in data analysis for statistical modeling.

Factors are created using the **factor** function by taking a vector as input.

Example

```
# Create a vector as input.  
data <- c("East","West","East","North","North","East","West","West","West","East","North")  
  
print(data)  
print(is.factor(data))
```

```
# Apply the factor function.
```

```
factor_data <- factor(data)
```

```
print(factor_data)  
print(is.factor(factor_data))
```

When we execute the above code, it produces the following result –

```
[1] "East" "West" "East" "North" "North" "East" "West" "West" "West" "East" "North"  
[1] FALSE  
[1] East West East North North East West West West East North  
Levels: East North West  
[1] TRUE
```

Factors in Data Frame

On creating any data frame with a column of text data, R treats the text column as categorical data and creates factors on it.

```
# Create the vectors for data frame.  
  
height <- c(132,151,162,139,166,147,122)  
weight <- c(48,49,66,53,67,52,40)  
gender <- c("male","male","female","female","male","female","male")  
  
# Create the data frame.  
  
input_data <- data.frame(height,weight,gender)  
print(input_data)  
  
# Test if the gender column is a factor.  
print(is.factor(input_data$gender))  
  
# Print the gender column so see the levels.  
print(input_data$gender)
```

When we execute the above code, it produces the following result –

```
height weight gender  
1 132 48 male
```

```
2 151 49 male
3 162 66 female
4 139 53 female
5 166 67 male
6 147 52 female
7 122 40 male
[1] TRUE
[1] male male female female male female male
Levels: female male
```

Changing the Order of Levels

The order of the levels in a factor can be changed by applying the factor function again with new order of the levels.

```
data <- c("East","West","East","North","North","East","West",
"West","West","East","North")
# Create the factors
factor_data <- factor(data)
print(factor_data)

# Apply the factor function with required order of the level.
new_order_data <- factor(factor_data,levels = c("East","West","North"))
print(new_order_data)
```

When we execute the above code, it produces the following result –

```
[1] East West East North North East West West West East North
Levels: East North West
[1] East West East North North East West West West East North
Levels: East West North
```

Generating Factor Levels

We can generate factor levels by using the **gl** function. It takes two integers as input which indicates how many levels and how many times each level.

Syntax

QUALITY THOUGHT	*	www.facebook.com/qthought	*	www.qualitythought.in
PH: 7730997544	*	Location: Ameerpet	*	Email: info@qualitythought.in

```
gl(n, k, labels)
```

Following is the description of the parameters used –

- **n** is a integer giving the number of levels.
- **k** is a integer giving the number of replications.
- **labels** is a vector of labels for the resulting factor levels.

Example

```
v <- gl(3,4, labels = c("Tampa","Seattle","Boston"))
print(v)
```

When we execute the above code, it produces the following result –

```
Tampa Tampa Tampa Tampa Seattle Seattle Seattle Seattle Boston
[10] Boston Boston Boston
Levels: Tampa Seattle Boston
```

R - DATA FRAMES

A data frame is a table or a two-dimensional array-like structure in which each column contains values of one variable and each row contains one set of values from each column.

Following are the characteristics of a data frame.

- The column names should be non-empty.
- The row names should be unique.
- The data stored in a data frame can be of numeric, factor or character type.
- Each column should contain same number of data items.

Create Data Frame

```
# Create the data frame.
emp.data <- data.frame(
  emp_id = c (1:5),
  emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
  salary = c(623.3,515.2,611.0,729.0,843.25),
```

```
start_date =as.Date(c("2012-01-01","2013-09-23","2014-11-15","2014-05-11",
"2015-03-27")),
stringsAsFactors = FALSE
)
# Print the data frame.
print(emp.data)
```

When we execute the above code, it produces the following result –

	emp_id	emp_name	salary	start_date
1	1	Rick	623.30	2012-01-01
2	2	Dan	515.20	2013-09-23
3	3	Michelle	611.00	2014-11-15
4	4	Ryan	729.00	2014-05-11
5	5	Gary	843.25	2015-03-27

Get the Structure of the Data Frame

The structure of the data frame can be seen by using **str** function.

```
# Create the data frame.
emp.data <- data.frame(
  emp_id = c (1:5),
  emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
  salary = c(623.3,515.2,611.0,729.0,843.25),
  start_date =as.Date(c("2012-01-01","2013-09-23","2014-11-15","2014-05-11",
"2015-03-27")),
  stringsAsFactors = FALSE
)
# Get the structure of the data frame.
str(emp.data)
```

When we execute the above code, it produces the following result –

```
'data.frame': 5 obs. of 4 variables:
 $ emp_id : int 1 2 3 4 5
```

```
$ emp_name : chr "Rick" "Dan" "Michelle" "Ryan" ...
$ salary : num 623 515 611 729 843
$ start_date: Date, format: "2012-01-01" "2013-09-23" "2014-11-15" "2014-05-11" ...
```

Summary of Data in Data Frame

The statistical summary and nature of the data can be obtained by applying **summary** function.

```
# Create the data frame.
emp.data <- data.frame(
  emp_id = c(1:5),
  emp_name = c("Rick", "Dan", "Michelle", "Ryan", "Gary"),
  salary = c(623.3, 515.2, 611.0, 729.0, 843.25),
  start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
  "2015-03-27")),
  stringsAsFactors = FALSE
)
# Print the summary.
print(summary(emp.data))
```

Ameerpet / Kondapur

When we execute the above code, it produces the following result –

```
  emp_id  emp_name      salary    start_date
Min.   :1  Length:5      Min.   :515.2  Min.   :2012-01-01
1st Qu.:2  Class :character  1st Qu.:611.0  1st Qu.:2013-09-23
Median :3  Mode   :character  Median :623.3  Median :2014-05-11
Mean   :3                  Mean   :664.4  Mean   :2014-01-14
3rd Qu.:4                  3rd Qu.:729.0  3rd Qu.:2014-11-15
Max.   :5                  Max.   :843.2  Max.   :2015-03-27
```

Extract Data from Data Frame

Extract specific column from a data frame using column name.

```
# Create the data frame.  
emp.data <- data.frame(  
  emp_id = c(1:5),  
  emp_name = c("Rick", "Dan", "Michelle", "Ryan", "Gary"),  
  salary = c(623.3, 515.2, 611.0, 729.0, 843.25),  
  
  start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",  
  "2015-03-27")),  
  stringsAsFactors = FALSE  
)  
  
# Extract Specific columns.  
result <- data.frame(emp.data$emp_name, emp.data$salary)  
print(result)
```

When we execute the above code, it produces the following result –

```
emp.data.emp_name emp.data.salary  
1      Rick      623.30  
2      Dan       515.20  
3    Michelle     611.00  
4      Ryan      729.00  
5      Gary      843.25
```

Extract the first two rows and then all columns

```
# Create the data frame.  
emp.data <- data.frame(  
  emp_id = c(1:5),  
  emp_name = c("Rick", "Dan", "Michelle", "Ryan", "Gary"),  
  salary = c(623.3, 515.2, 611.0, 729.0, 843.25),  
  
  start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",  
  "2015-03-27")),  
  stringsAsFactors = FALSE
```

```
)  
# Extract first two rows.  
result <- emp.data[1:2,]  
print(result)
```

When we execute the above code, it produces the following result –

```
emp_id  emp_name  salary  start_date  
1      1      Rick    623.3  2012-01-01  
2      2      Dan     515.2  2013-09-23
```

Extract 3rd and 5th row with 2nd and 4th column

```
# Create the data frame.  
emp.data <- data.frame(  
  emp_id = c(1:5),  
  emp_name = c("Rick", "Dan", "Michelle", "Ryan", "Gary"),  
  salary = c(623.3, 515.2, 611.0, 729.0, 843.25),  
  start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",  
  "2015-03-27")),  
  stringsAsFactors = FALSE  
)  
# Extract 3rd and 5th row with 2nd and 4th column.  
result <- emp.data[c(3,5),c(2,4)]  
print(result)
```

When we execute the above code, it produces the following result –

```
emp_name start_date  
3 Michelle 2014-11-15  
5 Gary 2015-03-27
```

Expand Data Frame

A data frame can be expanded by adding columns and rows.

Add Column

Just add the column vector using a new column name.

```
# Create the data frame.  
emp.data <- data.frame(  
  emp_id = c(1:5),  
  emp_name = c("Rick", "Dan", "Michelle", "Ryan", "Gary"),  
  salary = c(623.3, 515.2, 611.0, 729.0, 843.25),  
  
  start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",  
  "2015-03-27")),  
  stringsAsFactors = FALSE  
)  
  
# Add the "dept" column.  
emp.data$dept <- c("IT", "Operations", "IT", "HR", "Finance")  
v <- emp.data  
print(v)
```

When we execute the above code, it produces the following result –

	emp_id	emp_name	salary	start_date	dept
1	1	Rick	623.30	2012-01-01	IT
2	2	Dan	515.20	2013-09-23	Operations
3	3	Michelle	611.00	2014-11-15	IT
4	4	Ryan	729.00	2014-05-11	HR
5	5	Gary	843.25	2015-03-27	Finance

Add Row

To add more rows permanently to an existing data frame, we need to bring in the new rows in the same structure as the existing data frame and use the **rbind** function.

In the example below we create a data frame with new rows and merge it with the existing data frame to create the final data frame.

```
# Create the first data frame.

emp.data <- data.frame(
  emp_id = c(1:5),
  emp_name = c("Rick", "Dan", "Michelle", "Ryan", "Gary"),
  salary = c(623.3, 515.2, 611.0, 729.0, 843.25),

  start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
  "2015-03-27")),
  dept = c("IT", "Operations", "IT", "HR", "Finance"),
  stringsAsFactors = FALSE
)

# Create the second data frame

emp.newdata <- data.frame(
  emp_id = c(6:8),
  emp_name = c("Rasmi", "Pranab", "Tusar"),
  salary = c(578.0, 722.5, 632.8),
  start_date = as.Date(c("2013-05-21", "2013-07-30", "2014-06-17")),
  dept = c("IT", "Operations", "Finance"),
  stringsAsFactors = FALSE
)

# Bind the two data frames.

emp.finaldata <- rbind(emp.data, emp.newdata)
print(emp.finaldata)
```

When we execute the above code, it produces the following result –

	emp_id	emp_name	salary	start_date	dept
1	1	Rick	623.30	2012-01-01	IT
2	2	Dan	515.20	2013-09-23	Operations
3	3	Michelle	611.00	2014-11-15	IT
4	4	Ryan	729.00	2014-05-11	HR
5	5	Gary	843.25	2015-03-27	Finance

6	6	Rasmi	578.00	2013-05-21	IT
7	7	Pranab	722.50	2013-07-30	Operations
8	8	Tusar	632.80	2014-06-17	Fianance

R - PACKAGES

R packages are a collection of R functions, compiled code and sample data. They are stored under a directory called "**library**" in the R environment. By default, R installs a set of packages during installation. More packages are added later, when they are needed for some specific purpose. When we start the R console, only the default packages are available by default. Other packages which are already installed have to be loaded explicitly to be used by the R program that is going to use them.

All the packages available in R language are listed at R Packages.

Below is a list of commands to be used to check, verify and use the R packages.

Check Available R Packages

Get library locations containing R packages

```
.libPaths()
```

When we execute the above code, it produces the following result. It may vary depending on the local settings of your pc.

```
[2] "C:/Program Files/R/R-3.2.2/library"
```

Get the list of all the packages installed

```
library()
```

When we execute the above code, it produces the following result. It may vary depending on the local settings of your pc.

Packages in library 'C:/Program Files/R/R-3.2.2/library':

base	The R Base Package
boot	Bootstrap Functions (Originally by Angelo Canty for S)
class	Functions for Classification
cluster	"Finding Groups in Data": Cluster Analysis Extended Rousseeuw et al.
codetools	Code Analysis Tools for R

compiler	The R Compiler Package
datasets	The R Datasets Package
foreign	Read Data Stored by 'Minitab', 'S', 'SAS', 'SPSS', 'Stata', 'Systat', 'Weka', 'dBase', ...
graphics	The R Graphics Package
grDevices	The R Graphics Devices and Support for Colours and Fonts
grid	The Grid Graphics Package
KernSmooth & Jones (1995)	Functions for Kernel Smoothing Supporting Wand
lattice	Trellis Graphics for R
MASS	Support Functions and Datasets for Venables and Ripley's MASS
Matrix	Sparse and Dense Matrix Classes and Methods
methods	Formal Methods and Classes
mgcv	Mixed GAM Computation Vehicle with GCV/AIC/REML
nlme	Smoothness Estimation
nnet	Linear and Nonlinear Mixed Effects Models
parallel	Feed-Forward Neural Networks and Multinomial
rpart	Log-Linear Models
spatial	Support for Parallel computation in R
spines	Recursive Partitioning and Regression Trees
stats	Functions for Kriging and Point Pattern
stats4	Analysis
survival	Regression Spline Functions and Classes
tcltk	The R Stats Package
tools	Statistical Functions using S4 Classes
utils	Survival Analysis
	Tcl/Tk Interface
	Tools for Package Development
	The R Utils Package

Get all packages currently loaded in the R environment

search()

When we execute the above code, it produces the following result. It may vary depending on the local settings of your pc.

```
[1] ".GlobalEnv"      "package:stats"    "package:graphics"  
[4] "package:grDevices" "package:utils"     "package:datasets"  
[7] "package:methods"   "Autoloads"      "package:base"
```

Install a New Package

There are two ways to add new R packages. One is installing directly from the CRAN directory and another is downloading the package to your local system and installing it manually.

Install directly from CRAN

The following command gets the packages directly from CRAN webpage and installs the package in the R environment. You may be prompted to choose a nearest mirror. Choose the one appropriate to your location.

```
install.packages("Package Name")  
  
# Install the package named "XML".  
install.packages("XML")
```

Install package manually

Go to the link [R Packages](#) to download the package needed. Save the package as a **.zip** file in a suitable location in the local system.

Now you can run the following command to install this package in the R environment.

```
install.packages(file_name_with_path, repos = NULL, type ="source")  
  
# Install the package named "XML"  
install.packages("E:/XML_3.98-1.3.zip", repos = NULL, type ="source")
```

Huderahad

Load Package to Library

Before a package can be used in the code, it must be loaded to the current R environment. You also need to load a package that is already installed previously but not available in the current environment.

A package is loaded using the following command –

```
library("package Name", lib.loc = "path to library")  
  
# Load the package named "XML"  
install.packages("E:/XML_3.98-1.3.zip", repos = NULL, type = "source")
```

R - DATA RESHAPING

Data Reshaping in R is about changing the way data is organized into rows and columns. Most of the time data processing in R is done by taking the input data as a data frame. It is easy to extract data from the rows and columns of a data frame but there are situations when we need the data frame in a format that is different from format in which we received it. R has many functions to split, merge and change the rows to columns and vice-versa in a data frame.

Joining Columns and Rows in a Data Frame

We can join multiple vectors to create a data frame using the **cbind** function. Also we can merge two data frames using **rbind** function.

```
# Create vector objects.  
city <- c("Tampa","Seattle","Hartford","Denver")  
state <- c("FL","WA","CT","CO")  
zipcode <- c(33602,98104,06161,80294)  
  
# Combine above three vectors into one data frame.  
addresses <- cbind(city,state,zipcode)  
  
# Print a header.  
cat("# # # # The First data frame\n")  
  
# Print the data frame.  
print(addresses)  
  
# Create another data frame with similar columns  
new.address <- data.frame(  
  city = c("Lowry","Charlotte"),  
  state = c("CO","FL"),  
  zipcode = c("80230","33949"),  
  stringsAsFactors = FALSE  
)  
# Print a header.
```

```
cat("# # # The Second data frame\n")

# Print the data frame.
print(new.address)

# Combine rows from both the data frames.
all.addresses <- rbind(addresses,new.address)

# Print a header.
cat("# # # The combined data frame\n")

# Print the result.
print(all.addresses)
```

When we execute the above code, it produces the following result –

```
# # # The First data frame
  city    state zipcode
[1,] "Tampa"  "FL"  "33602"
[2,] "Seattle" "WA"  "98104"
[3,] "Hartford" "CT"  "6161"
[4,] "Denver"   "CO"  "80294"

# # # The Second data frame
  city    state zipcode
1  Lowry   CO    80230
2  Charlotte FL    33949

# # # The combined data frame
  city    state zipcode
1  Tampa   FL    33602
2  Seattle  WA    98104
3  Hartford CT    6161
4  Denver   CO    80294
5  Lowry   CO    80230
6  Charlotte FL    33949
```

Merging Data Frames

We can merge two data frames by using the **merge** function. The data frames must have same column names on which the merging happens.

In the example below, we consider the data sets about Diabetes in Pima Indian Women available in the library names "MASS". we merge the two data sets based on the values of blood pressure "bp" and body mass index "bmi". On choosing these two columns for merging, the records where values of these two variables match in both data sets are combined together to form a single data frame.

```
library(MASS)
merged.Pima<- merge(x =Pima.te, y =Pima.tr,
by.x = c("bp","bmi"),
by.y = c("bp","bmi")
)
print(merged.Pima)
nrow(merged.Pima)
```

When we execute the above code, it produces the following result –

	bp	bmi	npreg.x	glu.x	skin.x	ped.x	age.x	type.x	npreg.y	glu.y	skin.y	ped.y
1	60	33.8	1	117	23	0.466	27	No	2	125	20	0.088
2	64	29.7	2	75	24	0.370	33	No	2	100	23	0.368
3	64	31.2	5	189	33	0.583	29	Yes	3	158	13	0.295
4	64	33.2	4	117	27	0.230	24	No	1	96	27	0.289
5	66	38.1	3	115	39	0.150	28	No	1	114	36	0.289
6	68	38.5	2	100	25	0.324	26	No	7	129	49	0.439
7	70	27.4	1	116	28	0.204	21	No	0	124	20	0.254
8	70	33.1	4	91	32	0.446	22	No	9	123	44	0.374
9	70	35.4	9	124	33	0.282	34	No	6	134	23	0.542
10	72	25.6	1	157	21	0.123	24	No	4	99	17	0.294
11	72	37.7	5	95	33	0.370	27	No	6	103	32	0.324
12	74	25.9	9	134	33	0.460	81	No	8	126	38	0.162
13	74	25.9	1	95	21	0.673	36	No	8	126	38	0.162
14	78	27.6	5	88	30	0.258	37	No	6	125	31	0.565
15	78	27.6	10	122	31	0.512	45	No	6	125	31	0.565
16	78	39.4	2	112	50	0.175	24	No	4	112	40	0.236
17	88	34.5	1	117	24	0.403	40	Yes	4	127	11	0.598
					age.y	type.y						
1	31				No							
2	21				No							
3	24				No							
4	21				No							
5	21				No							

```
6 43 Yes
7 36 Yes
8 40 No
9 29 Yes
10 28 No
11 55 No
12 39 No
13 39 No
14 49 Yes
15 49 Yes
16 38 No
17 28 No
[1] 17
```

Melting and Casting

One of the most interesting aspects of R programming is about changing the shape of the data in multiple steps to get a desired shape. The functions used to do this are called **melt** and **cast**.

We consider the dataset called ships present in the library called "MASS".

```
library(MASS)
print(ships)
```

When we execute the above code, it produces the following result –

```
type year period service incidents
1 A 60 60 127 0
2 A 60 75 63 0
3 A 65 60 1095 3
4 A 65 75 1095 4
5 A 70 60 1512 6
.....
.....
8 A 75 75 2244 11
9 B 60 60 44882 39
10 B 60 75 17176 29
11 B 65 60 28609 58
.....
.....
17 C 60 60 1179 1
18 C 60 75 552 1
19 C 65 60 781 0
.....
```

.....

Melt the Data

Now we melt the data to organize it, converting all columns other than type and year into multiple rows.

```
molten.ships <- melt(ships, id = c("type","year"))
print(molten.ships)
```

When we execute the above code, it produces the following result –

```
  type year variable value
1   A  60   period   60
2   A  60   period   75
3   A  65   period   60
4   A  65   period   75
.....
.....
9   B  60   period   60
10  B  60   period   75
11  B  65   period   60
12  B  65   period   75
13  B  70   period   60
.....
.....
41  A  60   service  127
42  A  60   service  63
43  A  65   service 1095
.....
.....
70  D  70   service 1208
71  D  75   service   0
72  D  75   service 2051
73  E  60   service   45
74  E  60   service   0
75  E  65   service  789
.....
.....
101 C  70   incidents  6
102 C  70   incidents  2
103 C  75   incidents  0
104 C  75   incidents  1
105 D  60   incidents  0
106 D  60   incidents  0
```

.....
.....

Cast the Molten Data

We can cast the molten data into a new form where the aggregate of each type of ship for each year is created. It is done using the **cast** function.

```
recasted.ship <- cast(molten.ships, type+year~variable,sum)  
print(recasted.ship)
```

When we execute the above code, it produces the following result –

	type	year	period	service	incidents
1	A	60	135	190	0
2	A	65	135	2190	7
3	A	70	135	4865	24
4	A	75	135	2244	11
5	B	60	135	62058	68
6	B	65	135	48979	111
7	B	70	135	20163	56
8	B	75	135	7117	18
9	C	60	135	1731	2
10	C	65	135	1457	1
11	C	70	135	2731	8
12	C	75	135	274	1
13	D	60	135	356	0
14	D	65	135	480	0
15	D	70	135	1557	13
16	D	75	135	2051	4
17	E	60	135	45	0
18	E	65	135	1226	14
19	E	70	135	3318	17
20	E	75	135	542	1

R - CSV FILES

In R, we can read data from files stored outside the R environment. We can also write data into files which will be stored and accessed by the operating system. R can read and write into various file formats like csv, excel, xml etc.

In this chapter we will learn to read data from a csv file and then write data into a csv file. The file should be present in current working directory so that R can read it. Of course we can also set our own directory and read files from there.

Getting and Setting the Working Directory

You can check which directory the R workspace is pointing to using the **getwd** function. You can also set a new working directory using **setwd** function.

```
# Get and print current working directory.
```

```
print(getwd())
```

```
# Set current working directory.
```

```
setwd("/web/com")
```

```
# Get and print current working directory.
```

```
print(getwd())
```

When we execute the above code, it produces the following result –

```
[1] "/web/com/1441086124_2016"  
[1] "/web/com"
```

This result depends on your OS and your current directory where you are working.

Input as CSV File

The csv file is a text file in which the values in the columns are separated by a comma. Let's consider the following data present in the file named **input.csv**.

You can create this file using windows notepad by copying and pasting this data. Save the file as **input.csv** using the save As All files*.*.* option in notepad.

```
id,name,salary,start_date,dept  
1,Rick,623.3,2012-01-01,IT  
2,Dan,515.2,2013-09-23,Operations  
3,Michelle,611,2014-11-15,IT  
4,Ryan,729,2014-05-11,HR  
5,Gary,843.25,2015-03-27,Finance  
6,Nina,578,2013-05-21,IT  
7,Simon,632.8,2013-07-30,Operations
```

8,Guru,722.5,2014-06-17,Finance

Reading a CSV File

Following is a simple example of **read.csv** function to read a CSV file available in your current working directory –

```
data <- read.csv("input.csv")
print(data)
```

When we execute the above code, it produces the following result –

	id	name	salary	start_date	dept
1	1	Rick	623.30	2012-01-01	IT
2	2	Dan	515.20	2013-09-23	Operations
3	3	Michelle	611.00	2014-11-15	IT
4	4	Ryan	729.00	2014-05-11	HR
5	NA	Gary	843.25	2015-03-27	Finance
6	6	Nina	578.00	2013-05-21	IT
7	7	Simon	632.80	2013-07-30	Operations
8	8	Guru	722.50	2014-06-17	Finance

Analyzing the CSV File

By default the **read.csv** function gives the output as a data frame. This can be easily checked as follows. Also we can check the number of columns and rows.

```
data <- read.csv("input.csv")
print(is.data.frame(data))
print(ncol(data))
print(nrow(data))
```

When we execute the above code, it produces the following result –

```
[1] TRUE
[1] 5
[1] 8
```

Once we read data in a data frame, we can apply all the functions applicable to data frames as explained in subsequent section.

Get the maximum salary

```
# Create a data frame.  
data <- read.csv("input.csv")  
# Get the max salary from data frame.  
sal <- max(data$salary)  
print(sal)
```

When we execute the above code, it produces the following result –

```
[1] 843.25
```

Get the details of the person with max salary

We can fetch rows meeting specific filter criteria similar to a SQL where clause.

```
# Create a data frame.  
data <- read.csv("input.csv")  
  
# Get the max salary from data frame.  
sal <- max(data$salary)  
  
# Get the person detail having max salary.  
retval <- subset(data, salary == max(salary))  
print(retval)
```

When we execute the above code, it produces the following result –

```
  id  name salary start_date  dept  
5  NA  Gary  843.25 2015-03-27  Finance
```

Get all the people working in IT department

```
# Create a data frame.  
data <- read.csv("input.csv")
```

```
retval <- subset( data, dept == "IT")
print(retval)
```

When we execute the above code, it produces the following result –

```
  id  name  salary start_date dept
1  1  Rick  623.3  2012-01-01  IT
3  3 Michelle 611.0  2014-11-15  IT
6  6  Nina  578.0  2013-05-21  IT
```

Get the persons in IT department whose salary is greater than 600

```
# Create a data frame.
data <- read.csv("input.csv")

info <- subset(data, salary > 600 & dept == "IT")
print(info)
```

When we execute the above code, it produces the following result –

```
  id  name  salary start_date dept
1  1  Rick  623.3  2012-01-01  IT
3  3 Michelle 611.0  2014-11-15  IT
```

Get the people who joined on or after 2014

```
# Create a data frame.
data <- read.csv("input.csv")

retval <- subset(data, as.Date(start_date) > as.Date("2014-01-01"))
print(retval)
```

When we execute the above code, it produces the following result –

```
  id  name  salary start_date dept
3  3 Michelle 611.00  2014-11-15  IT
```

4	4	Ryan	729.00	2014-05-11	HR
5	NA	Gary	843.25	2015-03-27	Finance
8	8	Guru	722.50	2014-06-17	Finance

Writing into a CSV File

R can create csv file from existing data frame. The **write.csv** function is used to create the csv file. This file gets created in the working directory.

```
# Create a data frame.  
data <- read.csv("input.csv")  
retval <- subset(data,as.Date(start_date)>as.Date("2014-01-01"))  
  
# Write filtered data into a new file.  
write.csv(retval,"output.csv")  
newdata <- read.csv("output.csv")  
print(newdata)
```

When we execute the above code, it produces the following result –

X	id	name	salary	start_date	dept
1	3	Michelle	611.00	2014-11-15	IT
2	4	Ryan	729.00	2014-05-11	HR
3	NA	Gary	843.25	2015-03-27	Finance
4	8	Guru	722.50	2014-06-17	Finance

Here the column X comes from the data set newper. This can be dropped using additional parameters while writing the file.

```
# Create a data frame.  
data <- read.csv("input.csv")  
retval <- subset(data,as.Date(start_date)>as.Date("2014-01-01"))  
  
# Write filtered data into a new file.  
write.csv(retval,"output.csv", row.names = FALSE)
```

```
newdata <- read.csv("output.csv")
print(newdata)
```

When we execute the above code, it produces the following result –

```
  id  name  salary start_date  dept
1  3  Michelle  611.00 2014-11-15  IT
2  4    Ryan  729.00 2014-05-11  HR
3  NA   Gary  843.25 2015-03-27 Finance
4  8    Guru  722.50 2014-06-17 Finance
```

R - EXCEL FILE

Microsoft Excel is the most widely used spreadsheet program which stores data in the .xls or .xlsx format. R can read directly from these files using some excel specific packages. Few such packages are - XLConnect, xlsx, gdata etc. We will be using xlsx package. R can also write into excel file using this package.

Install xlsx Package

You can use the following command in the R console to install the "xlsx" package. It may ask to install some additional packages on which this package is dependent. Follow the same command with required package name to install the additional packages.

```
install.packages("xlsx")
```

Verify and Load the "xlsx" Package

Use the following command to verify and load the "xlsx" package.

```
# Verify the package is installed.
any(grepl("xlsx",installed.packages()))

# Load the library into R workspace.
library("xlsx")
```

When the script is run we get the following output.

```
[1] TRUE
Loading required package: rJava
```

Loading required package: methods
Loading required package: xlsxjars

Input as xlsx File

Open Microsoft excel. Copy and paste the following data in the work sheet named as sheet1.

id	name	salary	start_date	dept
1	Rick	623.3	1/1/2012	IT
2	Dan	515.2	9/23/2013	Operations
3	Michelle	611	11/15/2014	IT
4	Ryan	729	5/11/2014	HR
5	Gary	43.25	3/27/2015	Finance
6	Nina	578	5/21/2013	IT
7	Simon	632.8	7/30/2013	Operations
8	Guru	722.5	6/17/2014	Finance

Also copy and paste the following data to another worksheet and rename this worksheet to "city".

name	city
Rick	Seattle
Dan	Tampa
Michelle	Chicago
Ryan	Seattle
Gary	Houston
Nina	Boston
Simon	Mumbai
Guru	Dallas

Save the Excel file as "input.xlsx". You should save it in the current working directory of the R workspace.

Reading the Excel File

The input.xlsx is read by using the **read.xlsx** function as shown below. The result is stored as a data frame in the R environment.

```
# Read the first worksheet in the file input.xlsx.  
data <- read.xlsx("input.xlsx", sheetIndex = 1)  
print(data)
```

When we execute the above code, it produces the following result –

```
id, name, salary, start_date, dept
1 1 Rick 623.30 2012-01-01 IT
2 2 Dan 515.20 2013-09-23 Operations
3 3 Michelle 611.00 2014-11-15 IT
4 4 Ryan 729.00 2014-05-11 HR
5 NA Gary 843.25 2015-03-27 Finance
6 6 Nina 578.00 2013-05-21 IT
7 7 Simon 632.80 2013-07-30 Operations
8 8 Guru 722.50 2014-06-17 Finance
```

R - BINARY FILES

A binary file is a file that contains information stored only in form of bits and bytes. 0's and 1's and 0's and 1's. They are not human readable as the bytes in it translate to characters and symbols which contain many other non-printable characters. Attempting to read a binary file using any text editor will show characters like Ø and ð.

The binary file has to be read by specific programs to be useable. For example, the binary file of a Microsoft Word program can be read to a human readable form only by the Word program. Which indicates that, besides the human readable text, there is a lot more information like formatting of characters and page numbers etc., which are also stored along with alphanumeric characters. And finally a binary file is a continuous sequence of bytes. The line break we see in a text file is a character joining first line to the next.

Sometimes, the data generated by other programs are required to be processed by R as a binary file. Also R is required to create binary files which can be shared with other programs.

R has two functions **WriteBin** and **readBin** to create and read binary files.

Syntax

```
writeBin(object, con)
readBin(con, what, n )
```

Following is the description of the parameters used –

- **con** is the connection object to read or write the binary file.
- **object** is the binary file which to be written.
- **what** is the mode like character, integer etc. representing the bytes to be read.
- **n** is the number of bytes to read from the binary file.

Example

We consider the R inbuilt data "mtcars". First we create a csv file from it and convert it to a binary file and store it as a OS file. Next we read this binary file created into R.

Writing the Binary File

We read the data frame "mtcars" as a csv file and then write it as a binary file to the OS.

```
# Read the "mtcars" data frame as a csv file and store only the columns
"cyl","am"and"gear".
write.table(mtcars, file ="mtcars.csv",row.names = FALSE, na ="",
col.names = TRUE, sep =",")  
  
# Store 5 records from the csv file as a new data frame.
new.mtcars <- read.table("mtcars.csv",sep =",",header = TRUE,nrows =5)  
  
# Create a connection object to write the binary file using mode "wb".
write.filename = file("/web/com/binmtcars.dat","wb")
# Write the column names of the data frame to the connection object.
writeBin(colnames(new.mtcars), write.filename)
# Write the records in each of the column to the file.
writeBin(c(new.mtcars$cyl,new.mtcars$am,new.mtcars$gear), write.filename)  
  
# Close the file for writing so that it can be read by other program.
close(write.filename)
```

Reading the Binary File

The binary file created above stores all the data as continuous bytes. So we will read it by choosing appropriate values of column names as well as the column values.

```
# Create a connection object to read the file in binary mode using "rb".
read.filename <- file("/web/com/binmtcars.dat","rb")  
  
# First read the column names. n = 3 as we have 3 columns.
```

```
column.names <- readBin(read.filename, character(), n = 3)
```

```
# Next read the column values. n = 18 as we have 3 column names and 15 values.
```

```
read.filename <- file("/web/com/binmtcars.dat", "rb")
```

```
bindata <- readBin(read.filename, integer(), n = 18)
```

```
# Print the data.
```

```
print(bindata)
```

```
# Read the values from 4th byte to 8th byte which represents "cyl".
```

```
cyldata = bindata[4:8]
```

```
print(cyldata)
```

```
# Read the values from 9th byte to 13th byte which represents "am".
```

```
amdata = bindata[9:13]
```

```
print(amdata)
```

```
# Read the values from 14th byte to 18th byte which represents "gear".
```

```
geardata = bindata[14:18]
```

```
print(geardata)
```

```
# Combine all the read values to a data frame.
```

```
finaldata = cbind(cyldata, amdata, geardata)
```

```
colnames(finaldata) = column.names
```

```
print(finaldata)
```

When we execute the above code, it produces the following result and chart –

```
[1] 7108963 1728081249 7496037 6 6 4
[7] 6 8 1 1 1 0
[13] 0 4 4 4 3 3
[1] 6 6 4 6 8
```

```
[1] 1 1 1 0 0  
[1] 4 4 4 3 3  
cyl am gear  
[1] 6 1 4  
[2] 6 1 4  
[3] 4 1 4  
[4] 6 0 3  
[5] 8 0 3
```

As we can see, we got the original data back by reading the binary file in R.

R - XML FILES

XML is a file format which shares both the file format and the data on the World Wide Web, intranets, and elsewhere using standard ASCII text. It stands for Extensible Markup Language XML. Similar to HTML it contains markup tags. But unlike HTML where the markup tag describes structure of the page, in XML the markup tags describe the meaning of the data contained into the file.

You can read a XML file in R using the "XML" package. This package can be installed using following command.

```
install.packages("XML")
```

Input Data

Create a XML file by copying the below data into a text editor like notepad. Save the file with a **.xml** extension and choosing the file type as **all files*.*.*.***.

```
<RECORDS>  
<EMPLOYEE>  
<ID>1</ID>  
<NAME>Rick</NAME>  
<SALARY>623.3</SALARY>  
<STARTDATE>1/1/2012</STARTDATE>  
<DEPT>IT</DEPT>  
</EMPLOYEE>
```

```
<EMPLOYEE>
<ID>2</ID>
<NAME>Dan</NAME>
<SALARY>515.2</SALARY>
<STARTDATE>9/23/2013</STARTDATE>
<DEPT>Operations</DEPT>
</EMPLOYEE>
```

```
<EMPLOYEE>
<ID>3</ID>
<NAME>Michelle</NAME>
<SALARY>611</SALARY>
<STARTDATE>11/15/2014</STARTDATE>
<DEPT>IT</DEPT>
</EMPLOYEE>
```

```
<EMPLOYEE>
<ID>4</ID>
<NAME>Ryan</NAME>
<SALARY>729</SALARY>
<STARTDATE>5/11/2014</STARTDATE>
<DEPT>HR</DEPT>
</EMPLOYEE>
```

```
<EMPLOYEE>
<ID>5</ID>
<NAME>Gary</NAME>
<SALARY>843.25</SALARY>
<STARTDATE>3/27/2015</STARTDATE>
<DEPT>Finance</DEPT>
```

```
</EMPLOYEE>

<EMPLOYEE>
<ID>6</ID>
<NAME>Nina</NAME>
<SALARY>578</SALARY>
<STARTDATE>5/21/2013</STARTDATE>
<DEPT>IT</DEPT>
</EMPLOYEE>

<EMPLOYEE>
<ID>7</ID>
<NAME>Simon</NAME>
<SALARY>632.8</SALARY>
<STARTDATE>7/30/2013</STARTDATE>
<DEPT>Operations</DEPT>
</EMPLOYEE>

<EMPLOYEE>
<ID>8</ID>
<NAME>Guru</NAME>
<SALARY>722.5</SALARY>
<STARTDATE>6/17/2014</STARTDATE>
<DEPT>Finance</DEPT>
</EMPLOYEE>

</RECORDS>
```

Reading XML File

The xml file is read by R using the function **xmlParse**. It is stored as a list in R.

```
# Load the package required to read XML files.  
library("XML")  
  
# Also load the other required package.  
library("methods")  
  
# Give the input file name to the function.  
result <- xmlParse(file ="input.xml")  
  
# Print the result.  
print(result)
```

When we execute the above code, it produces the following result –

```
1  
Rick  
623.3  
1/1/2012  
IT  
  
2  
Dan  
515.2  
9/23/2013  
Operations  
  
3  
Michelle  
611  
11/15/2014  
IT  
  
4  
Ryan  
729  
5/11/2014  
HR  
  
5  
Gary  
843.25
```

3/27/2015

Finance

6

Nina

578

5/21/2013

IT

7

Simon

632.8

7/30/2013

Operations

8

Guru

722.5

6/17/2014

Finance

Get Number of Nodes Present in XML File

```
# Load the packages required to read XML files.
```

```
library("XML")
```

```
library("methods")
```

```
# Give the input file name to the function.
```

```
result <- xmlParse(file ="input.xml")
```

```
# Extract the root node form the xml file.
```

```
rootnode <- xmlRoot(result)
```

```
# Find number of nodes in the root.
```

```
rootsize <- xmlSize(rootnode)
```

```
# Print the result.
```

```
print(rootsize)
```

When we execute the above code, it produces the following result –

```
output  
[1] 8
```

Details of the First Node

Let's look at the first record of the parsed file. It will give us an idea of the various elements present in the top level node.

```
# Load the packages required to read XML files.  
library("XML")  
library("methods")  
  
# Give the input file name to the function.  
result <- xmlParse(file ="input.xml")  
# Extract the root node form the xml file.  
rootnode <- xmlRoot(result)  
  
# Print the result.  
print(rootnode[1])
```

When we execute the above code, it produces the following result –

```
$EMPLOYEE  
1  
Rick  
623.3  
1/1/2012  
IT  
  
attr("class")  
[1] "XMLInternalNodeList" "XMLNodeList"
```

Get Different Elements of a Node

```
# Load the packages required to read XML files.  
library("XML")  
library("methods")  
  
# Give the input file name to the function.  
result <- xmlParse(file ="input.xml")  
  
# Extract the root node form the xml file.  
rootnode <- xmlRoot(result)  
  
# Get the first element of the first node.  
print(rootnode[[1]][[1]])  
  
# Get the fifth element of the first node.  
print(rootnode[[1]][[5]])  
# Get the second element of the third node.  
print(rootnode[[3]][[2]])
```

When we execute the above code, it produces the following result –

```
1  
IT  
Michelle
```

XML to Data Frame

To handle the data effectively in large files we read the data in the xml file as a data frame. Then process the data frame for data analysis.

```
# Load the packages required to read XML files.  
library("XML")  
library("methods")  
  
# Convert the input xml file to a data frame.
```

```
xmldataframe <- xmlToDataFrame("input.xml")  
print(xmldataframe)
```

When we execute the above code, it produces the following result –

	ID	NAME	SALARY	STARTDATE	DEPT
1	1	Rick	623.30	2012-01-01	IT
2	2	Dan	515.20	2013-09-23	Operations
3	3	Michelle	611.00	2014-11-15	IT
4	4	Ryan	729.00	2014-05-11	HR
5	NA	Gary	843.25	2015-03-27	Finance
6	6	Nina	578.00	2013-05-21	IT
7	7	Simon	632.80	2013-07-30	Operations
8	8	Guru	722.50	2014-06-17	Finance

As the data is now available as a dataframe we can use data frame related function to read and manipulate the file.

R - JSON FILES

JSON file stores data as text in human-readable format. Json stands for JavaScript Object Notation. R can read JSON files using the rjson package.

Install rjson Package

In the R console, you can issue the following command to install the rjson package.

```
install.packages("rjson")
```

Input Data

Create a JSON file by copying the below data into a text editor like notepad. Save the file with a **.json** extension and choosing the file type as **all files*.*.***.

```
{  
  "ID": ["1", "2", "3", "4", "5", "6", "7", "8"],  
  "Name": ["Rick", "Dan", "Michelle", "Ryan", "Gary", "Nina", "Simon", "Guru"],  
  "Salary": ["623.3", "515.2", "611", "729", "843.25", "578", "632.8", "722.5"],  
  "Dept": ["IT", "Operations", "IT", "HR", "Finance", "IT", "Operations", "Finance"]}
```

```
"StartDate":["1/1/2012","9/23/2013","11/15/2014","5/11/2014","3/27/2015","5/21/2013",
"7/30/2013","6/17/2014"],
"Dept":["IT","Operations","IT","HR","Finance","IT","Operations","Finance"]
}
```

Read the JSON File

The JSON file is read by R using the function from **JSON**. It is stored as a list in R.

```
# Load the package required to read JSON files.
library("rjson")

# Give the input file name to the function.
result <- fromJSON(file ="input.json")

# Print the result.
print(result)
```

When we execute the above code, it produces the following result –

```
$ID
[1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"

$name
[1] "Rick"  "Dan"   "Michelle" "Ryan"  "Gary"  "Nina"  "Simon"  "Guru"

$Salary
[1] "623.3" "515.2" "611"   "729"  "843.25" "578"  "632.8"  "722.5"

$StartDate
[1] "1/1/2012" "9/23/2013" "11/15/2014" "5/11/2014" "3/27/2015" "5/21/2013"
"7/30/2013" "6/17/2014"

$Dept
[1] "IT"      "Operations" "IT"      "HR"      "Finance"  "IT"
"Operations" "Finance"
```

Convert JSON to a Data Frame

We can convert the extracted data above to a R data frame for further analysis using the **as.data.frame** function.

```
# Load the package required to read JSON files.  
library("rjson")  
  
# Give the input file name to the function.  
result <- fromJSON(file ="input.json")  
  
# Convert JSON file to a data frame.  
json_data_frame <- as.data.frame(result)  
  
print(json_data_frame)
```

When we execute the above code, it produces the following result –

		id	name	salary	start_date	dept
1	1	Rick	623.30	2012-01-01	IT	
2	2	Dan	515.20	2013-09-23	Operations	
3	3	Michelle	611.00	2014-11-15	IT	
4	4	Ryan	729.00	2014-05-11	HR	
5	NA	Gary	843.25	2015-03-27	Finance	
6	6	Nina	578.00	2013-05-21	IT	
7	7	Simon	632.80	2013-07-30	Operations	
8	8	Guru	722.50	2014-06-17	Finance	

R - WEB DATA

Many websites provide data for consumption by its users. For example the World Health Organization WHO provides reports on health and medical information in the form of CSV, txt and XML files. Using R programs, we can programmatically extract specific data from such websites. Some packages in R which are used to scrap data from the web are – "RCurl", "XML", and "stringr". They are used to connect to the URL's, identify required links for the files and download them to the local environment.

Install R Packages

The following packages are required for processing the URL's and links to the files. If they are not available in your R Environment, you can install them using following commands.

```
install.packages("RCurl")  
install.packages("XML")
```

```
install.packages("stringr")
install.packages("plyr")
```

Input Data

We will visit the URL [weather data](#) and download the CSV files using R for the year 2015.

Example

We will use the function **getHTMLLinks** to gather the URLs of the files. Then we will use the function **download.file** to save the files to the local system. As we will be applying the same code again and again for multiple files, we will create a function to be called multiple times. The filenames are passed as parameters in form of a R list object to this function.

```
# Read the URL.
url <- "http://www.geos.ed.ac.uk/~weather/jcmb_ws/"

# Gather the html links present in the webpage.
links <- getHTMLLinks(url)

# Identify only the links which point to the JCMB 2015 files.
filenames <- links[str_detect(links, "JCMB_2015")]

# Store the file names as a list.
filenames_list <- as.list(filenames)

# Create a function to download the files by passing the URL and filename list.
downloadcsv <- function(mainurl,filename){
  filedetails <- str_c(mainurl,filename)
  download.file(filedetails,filename)
}

# Now apply the l_ply function and save the files into the current R working directory.
```

```
l_ply(filenames, downloadcsv, mainurl  
="http://www.geos.ed.ac.uk/~weather/jcmb_ws/")
```

Verify the File Download

After running the above code, you can locate the following files in the current R working directory.

```
"JCMB_2015.csv" "JCMB_2015_Apr.csv" "JCMB_2015_Feb.csv" "JCMB_2015_Jan.csv"  
"JCMB_2015_Mar.csv"
```

R - DATABASES

The data in Relational database systems are stored in a normalized format. So, to carry out statistical computing we will need very advanced and complex Sql queries. But R can connect easily to many relational databases like MySql, Oracle, Sql server etc. and fetch records from them as a data frame. Once the data is available in the R environment, it becomes a normal R data set and can be manipulated or analyzed using all the powerful packages and functions.

In this tutorial we will be using MySql as our reference database for connecting to R.

RMySQL Package

R has a built-in package named "RMySQL" which provides native connectivity between with MySql database. You can install this package in the R environment using the following command.

```
install.packages("RMySQL")
```

Connecting R to MySql

Once the package is installed we create a connection object in R to connect to the database. It takes the username, password, database name and host name as input.

```
# Create a connection Object to MySQL database.  
  
# We will connect to the sampel database named "sakila" that comes with MySql  
installation.  
  
mysqlconnection = dbConnect(MySQL(), user ='root', password ="", dbname ='sakila',  
host ='localhost')  
  
# List the tables available in this database.
```

```
dbListTables(mysqlconnection)
```

When we execute the above code, it produces the following result –

```
[1] "actor"           "actor_info"  
[3] "address"        "category"  
[5] "city"            "country"  
[7] "customer"        "customer_list"  
[9] "film"            "film_actor"  
[11] "film_category"  "film_list"  
[13] "film_text"       "inventory"  
[15] "language"        "nicer_but_slower_film_list"  
[17] "payment"         "rental"  
[19] "sales_by_film_category"  "sales_by_store"  
[21] "staff"           "staff_list"  
[23] "store"
```

Querying the Tables

We can query the database tables in MySql using the function **dbSendQuery**. The query gets executed in MySql and the result set is returned using the R **fetch** function. Finally it is stored as a data frame in R.

```
# Query the "actor" tables to get all the rows.  
result = dbSendQuery(mysqlconnection,"select * from actor")  
  
# Store the result in a R data frame object. n = 5 is used to fetch first 5 rows.  
data.frame = fetch(result, n =5)  
print(data.fame)
```

When we execute the above code, it produces the following result –

	actor_id	first_name	last_name	last_update
1	1	PENELOPE	GUINNESS	2006-02-15 04:34:33
2	2	NICK	WAHLBERG	2006-02-15 04:34:33
3	3	ED	CHASE	2006-02-15 04:34:33
4	4	JENNIFER	DAVIS	2006-02-15 04:34:33
5	5	JOHNNY	LOLLOBRIGIDA	2006-02-15 04:34:33

Query with Filter Clause

We can pass any valid select query to get the result.

```
result = dbSendQuery(mysqlconnection,"select * from actor where last_name = 'TORN'"')  
  
# Fetch all the records(with n = -1) and store it as a data frame.  
data.frame = fetch(result, n =-1)  
print(data)
```

When we execute the above code, it produces the following result –

	actor_id	first_name	last_name	last_update
1	18	DAN	TORN	2006-02-15 04:34:33
2	94	KENNETH	TORN	2006-02-15 04:34:33
3	102	WALTER	TORN	2006-02-15 04:34:33

Updating Rows in the Tables

We can update the rows in a Mysql table by passing the update query to the dbSendQuery function.

```
dbSendQuery(mysqlconnection,"update mtcars set disp = 168.5 where hp = 110")
```

After executing the above code we can see the table updated in the MySql Environment.

Inserting Data into the Tables

```
dbSendQuery(mysqlconnection,  
"insert into mtcars(row_names, mpg, cyl, disp, hp, drat, wt, qsec, vs, am, gear, carb)  
values('New Mazda RX4 Wag', 21, 6, 168.5, 110, 3.9, 2.875, 17.02, 0, 1, 4, 4)"  
)
```

After executing the above code we can see the row inserted into the table in the MySql Environment.

Creating Tables in MySql

We can create tables in the MySql using the function **dbWriteTable**. It overwrites the table if it already exists and takes a data frame as input.

```
# Create the connection object to the database where we want to create the table.  
mysqlconnection = dbConnect(MySQL(), user ='root', password ='', dbname ='sakila',  
host ='localhost')  
  
# Use the R data frame "mtcars" to create the table in MySql.  
# All the rows of mtcars are taken inot MySql.  
dbWriteTable(mysqlconnection,"mtcars", mtcars[,], overwrite = TRUE)
```

After executing the above code we can see the table created in the MySql Environment.

Dropping Tables in MySql

We can drop the tables in MySql database passing the drop table statement into the dbSendQuery in the same way we used it for querying data from tables.

```
dbSendQuery(mysqlconnection,'drop table if exists mtcars')
```

After executing the above code we can see the table is dropped in the MySql Environment.

R - PIE CHARTS

R Programming language has numerous libraries to create charts and graphs. A pie-chart is a representation of values as slices of a circle with different colors. The slices are labeled and the numbers corresponding to each slice is also represented in the chart.

In R the pie chart is created using the **pie** function which takes positive numbers as a vector input. The additional parameters are used to control labels, color, title etc.

Syntax

The basic syntax for creating a pie-chart using the R is –

```
pie(x, labels, radius, main, col, clockwise)
```

Following is the description of the parameters used –

- **x** is a vector containing the numeric values used in the pie chart.
- **labels** is used to give description to the slices.

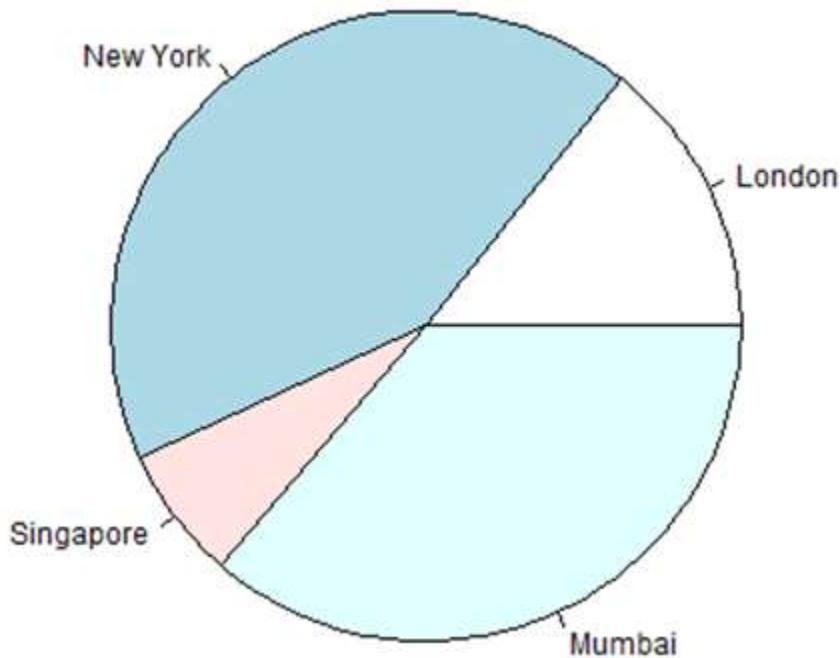
- **radius** indicates the radius of the circle of the pie chart. value between -1 and +1 value between -1 and +1.
- **main** indicates the title of the chart.
- **col** indicates the color palette.
- **clockwise** is a logical value indicating if the slices are drawn clockwise or anti clockwise.

Example

A very simple pie-chart is created using just the input vector and labels. The below script will create and save the pie chart in the current R working directory.

```
# Create data for the graph.  
x <- c(21,62,10,53)  
labels <- c("London","New York","Singapore","Mumbai")  
  
# Give the chart file a name.  
png(file ="city.jpg")  
  
# Plot the chart.  
pie(x,labels)  
  
# Save the file.  
dev.off()
```

When we execute the above code, it produces the following result –



Pie Chart Title and Colors

We can expand the features of the chart by adding more parameters to the function. We will use parameter **main** to add a title to the chart and another parameter is **col** which will make use of rainbow colour pallet while drawing the chart. The length of the pallet should be same as the number of values we have for the chart. Hence we use **lengthxx**.

Example

The below script will create and save the pie chart in the current R working directory.

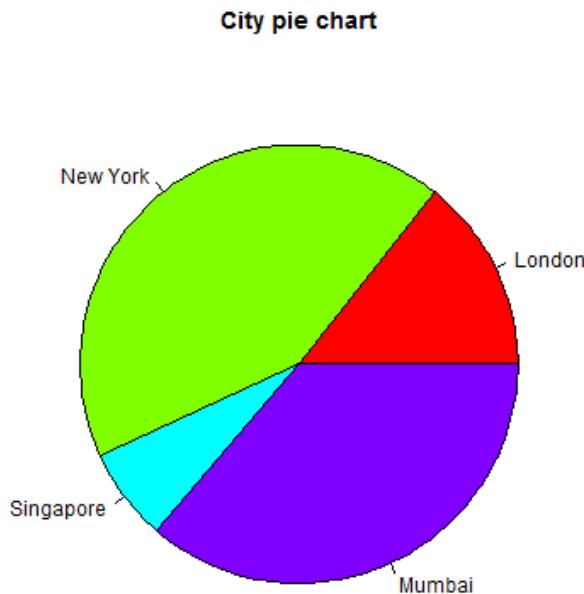
```
# Create data for the graph.
x <- c(21,62,10,53)
labels <- c("London","New York","Singapore","Mumbai")
# Give the chart file a name.
png(file ="city_title_colours.jpg")

# Plot the chart with title and rainbow color pallet.
pie(x, labels, main ="City pie chart", col = rainbow(length(x)))
```

```
# Save the file.
```

```
dev.off()
```

When we execute the above code, it produces the following result –



Slice Percentages and Chart Legend

We can add slice percentage and a chart legend by creating additional chart variables.

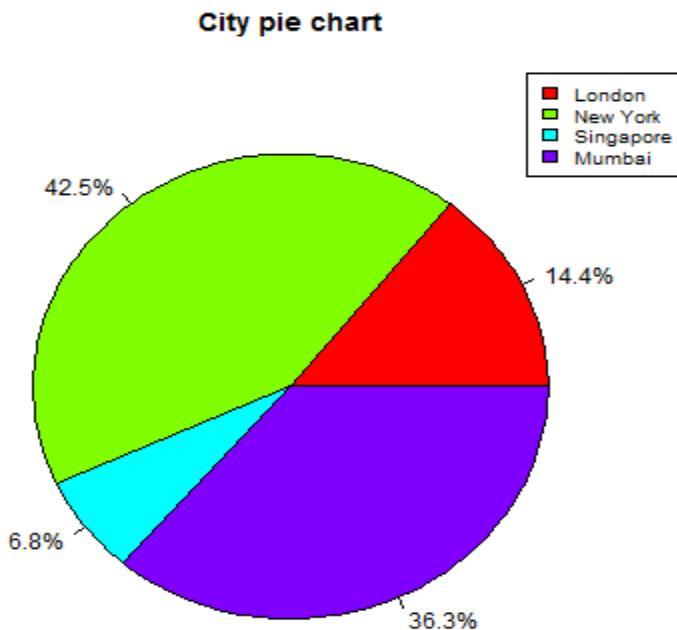
```
# Create data for the graph.  
x <- c(21,62,10,53)  
labels <- c("London","New York","Singapore","Mumbai")  
piepercent<- round(100*x/sum(x),1)  
  
# Give the chart file a name.  
png(file ="city_percentage_legends.jpg")  
  
# Plot the chart.  
pie(x, labels = piepercent, main ="City pie chart",col = rainbow(length(x)))  
legend("topright", c("London","New York","Singapore","Mumbai"), cex =0.8,
```

```
fill = rainbow(length(x))
```

```
# Save the file.
```

```
dev.off()
```

When we execute the above code, it produces the following result –



3D Pie Chart

A pie chart with 3 dimensions can be drawn using additional packages. The package **plotrix** has a function called **pie3D** that is used for this.

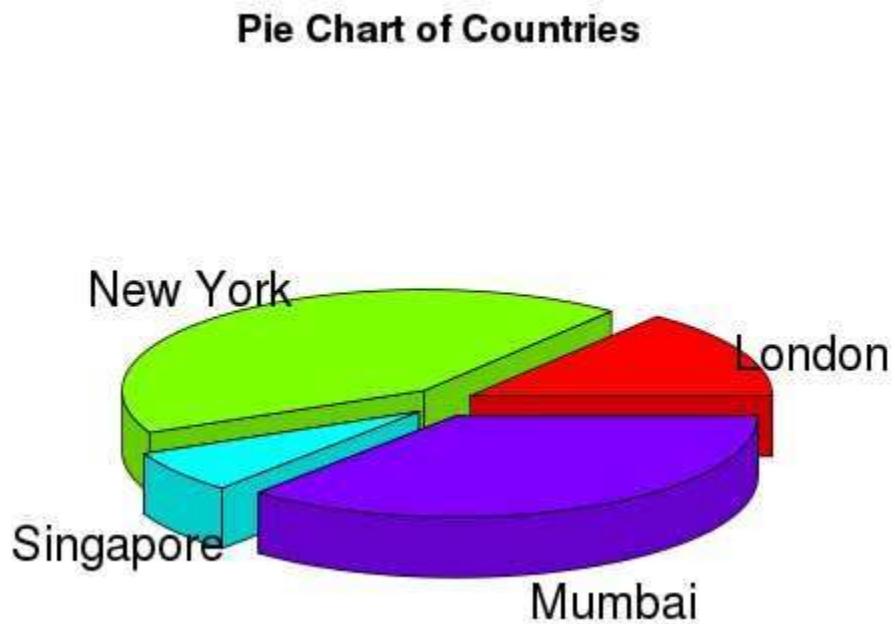
```
# Get the library.
library(plotrix)

# Create data for the graph.
x <- c(21,62,10,53)
lbl <- c("London","New York","Singapore","Mumbai")

# Give the chart file a name.
png(file ="3d_pie_chart.jpg")
```

```
# Plot the chart.  
pie3D(x,labels = lbl,explode =0.1, main ="Pie Chart of Countries ")  
  
# Save the file.  
dev.off()
```

When we execute the above code, it produces the following result –



R - BAR CHARTS

A bar chart represents data in rectangular bars with length of the bar proportional to the value of the variable. R uses the function **barplot** to create bar charts. R can draw both vertical and Horizontal bars in the bar chart. In bar chart each of the bars can be given different colors.

Syntax

The basic syntax to create a bar-chart in R is –

```
barplot(H,xlab,ylab,main, names.arg,col)
```

Following is the description of the parameters used –

- **H** is a vector or matrix containing numeric values used in bar chart.
- **xlab** is the label for x axis.
- **ylab** is the label for y axis.
- **main** is the title of the bar chart.
- **names.arg** is a vector of names appearing under each bar.
- **col** is used to give colors to the bars in the graph.

Example

A simple bar chart is created using just the input vector and the name of each bar.

The below script will create and save the bar chart in the current R working directory.

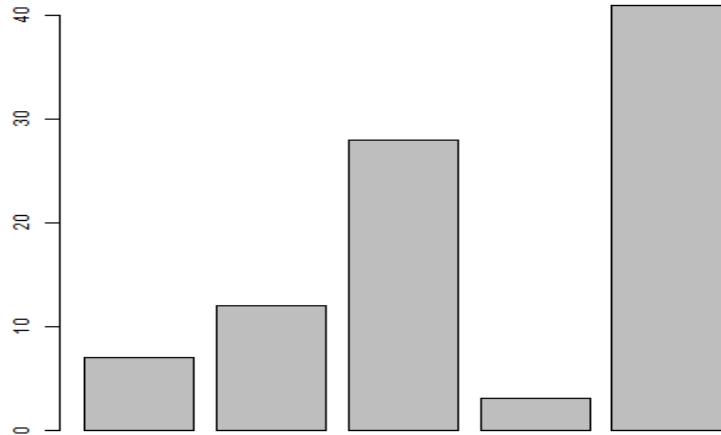
```
# Create the data for the chart
H <- c(7,12,28,3,41)

# Give the chart file a name
png(file ="barchart.png")

# Plot the bar chart
barplot(H)

# Save the file
dev.off()
```

When we execute above code, it produces following result –



Bar Chart Labels, Title and Colors

The features of the bar chart can be expanded by adding more parameters. The **main** parameter is used to add **title**. The **col** parameter is used to add colors to the bars. The **args.name** is a vector having same number of values as the input vector to describe the meaning of each bar.

Example

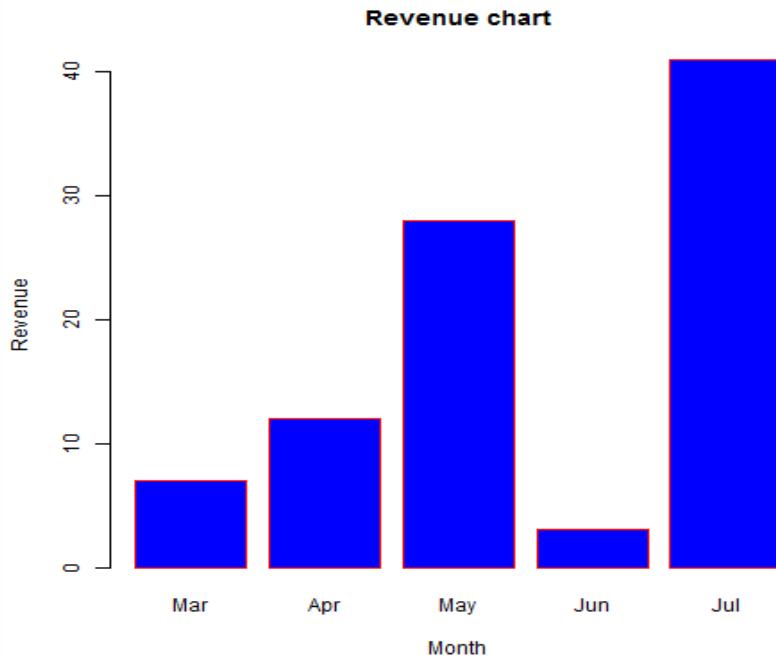
The below script will create and save the bar chart in the current R working directory.

```
# Create the data for the chart
H <- c(7,12,28,3,41)
M <- c("Mar","Apr","May","Jun","Jul")

# Give the chart file a name
png(file ="barchart_months_revenue.png")

# Plot the bar chart
barplot(H,names.arg=M,xlab="Month",ylab="Revenue",col="blue",
main="Revenue chart",border="red")
# Save the file
dev.off()
```

When we execute above code, it produces following result –



Group Bar Chart and Stacked Bar Chart

We can create bar chart with groups of bars and stacks in each bar by using a matrix as input values.

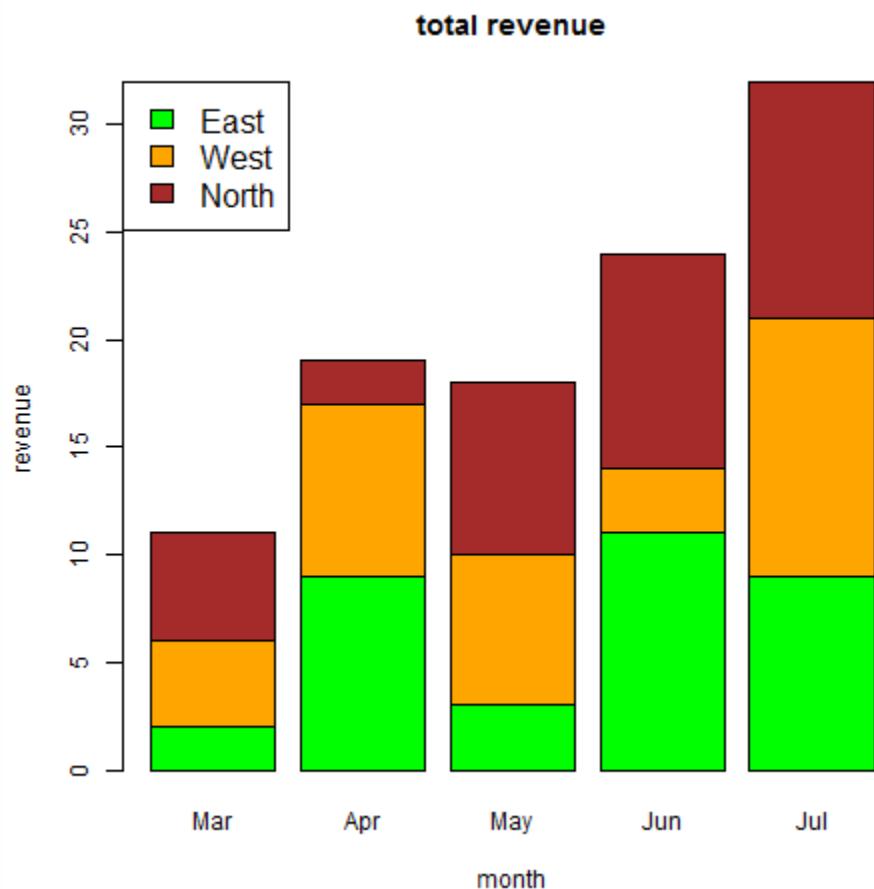
More than two variables are represented as a matrix which is used to create the group bar chart and stacked bar chart.

```
# Create the input vectors.  
colors = c("green","orange","brown")  
months <- c("Mar","Apr","May","Jun","Jul")  
regions <- c("East","West","North")  
  
# Create the matrix of the values.  
Values<- matrix(c(2,9,3,11,9,4,8,7,3,12,5,2,8,10,11), nrow =3, ncol =5, byrow = TRUE)  
# Give the chart file a name  
png(file ="barchart_stacked.png")  
  
# Create the bar chart
```

```
barplot(values, main ="total revenue", names.arg = months, xlab ="month", ylab ="revenue", col = colors)

# Add the legend to the chart
legend("topleft", regions, cex =1.3, fill = colors)

# Save the file
dev.off()
```



R - BOXPLOTS

Boxplots are a measure of how well distributed is the data in a data set. It divides the data set into three quartiles. This graph represents the minimum, maximum, median, first quartile and third quartile in the data set. It is also useful in comparing the distribution of data across data sets by drawing boxplots for each of them.

Boxplots are created in R by using the **boxplot** function.

Syntax

The basic syntax to create a boxplot in R is –

```
boxplot(x, data, notch, varwidth, names, main)
```

Following is the description of the parameters used –

- **x** is a vector or a formula.
- **data** is the data frame.
- **notch** is a logical value. Set as TRUE to draw a notch.
- **varwidth** is a logical value. Set as true to draw width of the box proportionate to the sample size.
- **names** are the group labels which will be printed under each boxplot.
- **main** is used to give a title to the graph.

Example

We use the data set "mtcars" available in the R environment to create a basic boxplot. Let's look at the columns "mpg" and "cyl" in mtcars.

```
input <- mtcars[,c('mpg','cyl')]  
print(head(input))
```

When we execute above code, it produces following result –

```
  mpg cyl  
Mazda RX4    21.0 6  
Mazda RX4 Wag 21.0 6  
Datsun 710   22.8 4  
Hornet 4 Drive 21.4 6  
Hornet Sportabout 18.7 8  
Valiant     18.1 6
```

Creating the Boxplot

The below script will create a boxplot graph for the relation between mpg milespergallon and cyl numberofcylinders.

```
# Give the chart file a name.
```

```
png(file ="boxplot.png")

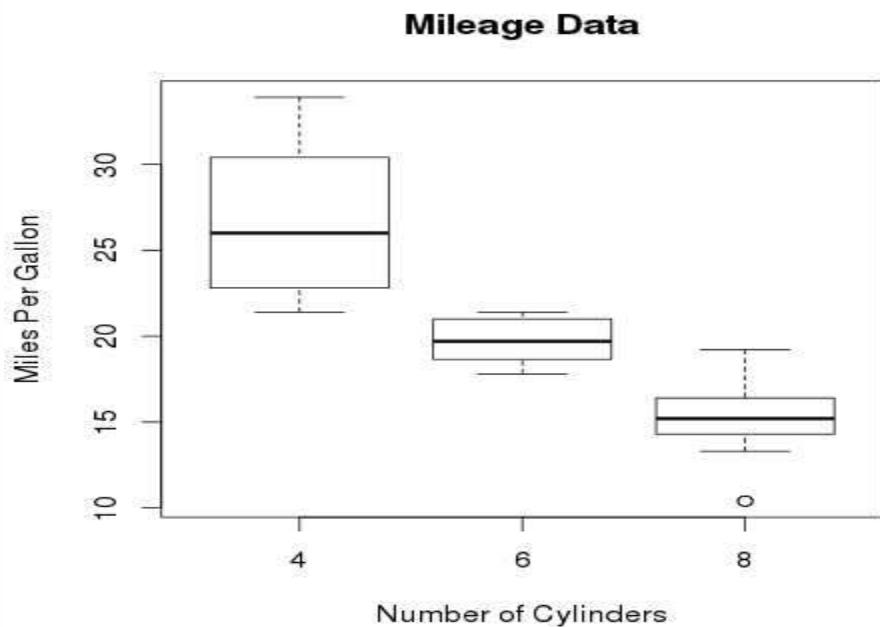
# Plot the chart.

boxplot(mpg ~ cyl, data = mtcars, xlab ="Number of Cylinders",
        ylab ="Miles Per Gallon", main ="Mileage Data")

# Save the file.

dev.off()
```

When we execute the above code, it produces the following result –



Boxplot with Notch

We can draw boxplot with notch to find out how the medians of different data groups match with each other.

The below script will create a boxplot graph with notch for each of the data group.

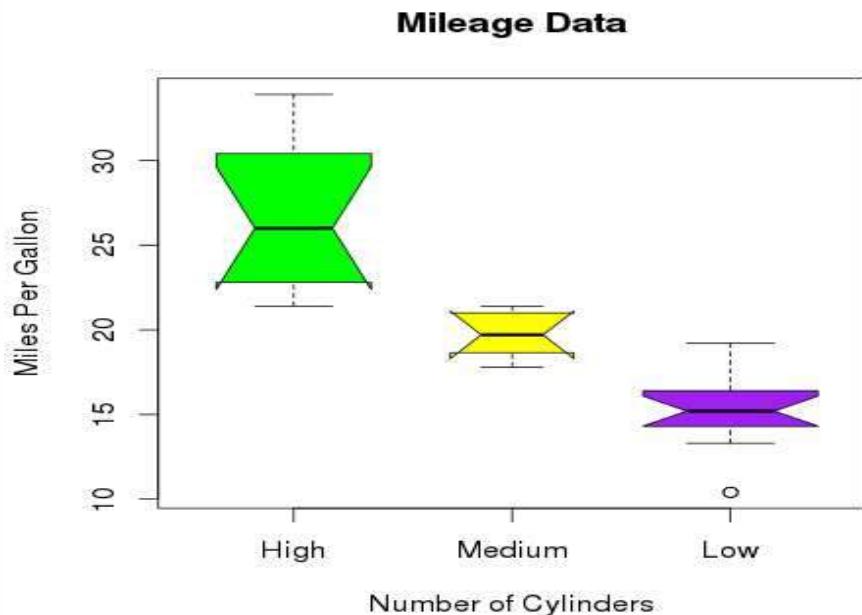
```
# Give the chart file a name.

png(file ="boxplot_with_notch.png")
```

```
# Plot the chart.
```

```
boxplot(mpg ~ cyl, data = mtcars,  
       xlab = "Number of Cylinders",  
       ylab = "Miles Per Gallon",  
       main = "Mileage Data",  
       notch = TRUE,  
       varwidth = TRUE,  
       col = c("green","yellow","purple"),  
       names = c("High","Medium","Low"))  
}  
  
# Save the file.  
dev.off()
```

When we execute the above code, it produces the following result –



R - HISTOGRAMS

A histogram represents the frequencies of values of a variable bucketed into ranges. Histogram is similar to bar chart but the difference is it groups the values into continuous ranges. Each bar in histogram represents the height of the number of values present in that range.

R creates histogram using **hist** function. This function takes a vector as an input and uses some more parameters to plot histograms.

Syntax

The basic syntax for creating a histogram using R is –

```
hist(v,main,xlab,xlim,ylim,breaks,col,border)
```

Following is the description of the parameters used –

- **v** is a vector containing numeric values used in histogram.
- **main** indicates title of the chart.
- **col** is used to set color of the bars.
- **border** is used to set border color of each bar.
- **xlab** is used to give description of x-axis.
- **xlim** is used to specify the range of values on the x-axis.
- **ylim** is used to specify the range of values on the y-axis.
- **breaks** is used to mention the width of each bar.

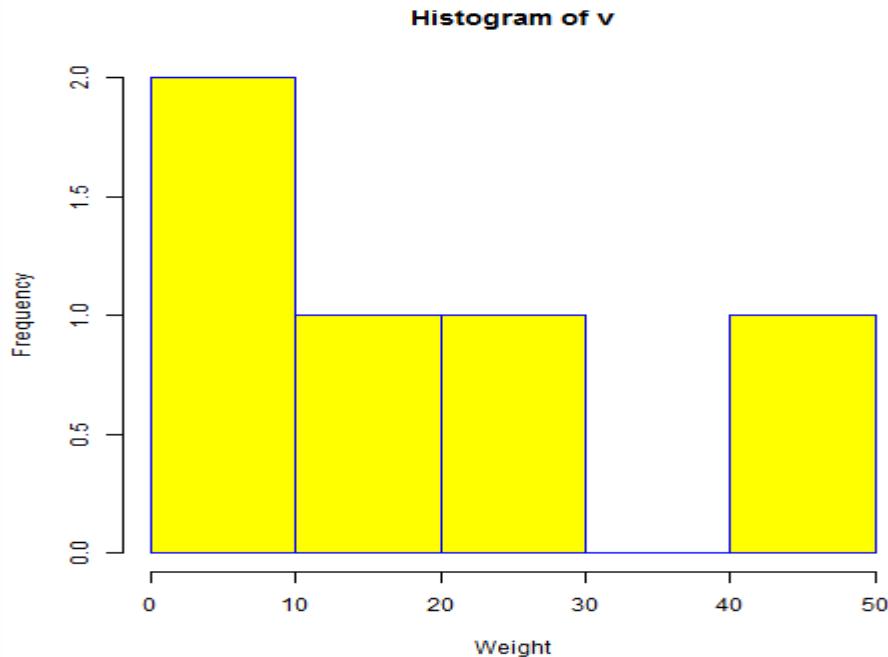
Example

A simple histogram is created using input vector, label, col and border parameters.

The script given below will create and save the histogram in the current R working directory.

```
# Create data for the graph.  
v <- c(9,13,21,8,36,22,12,41,31,33,19)  
  
# Give the chart file a name.  
png(file ="histogram.png")  
  
# Create the histogram.  
hist(v,xlab ="Weight",col ="yellow",border ="blue")  
  
# Save the file.  
dev.off()
```

When we execute the above code, it produces the following result –



Range of X and Y values

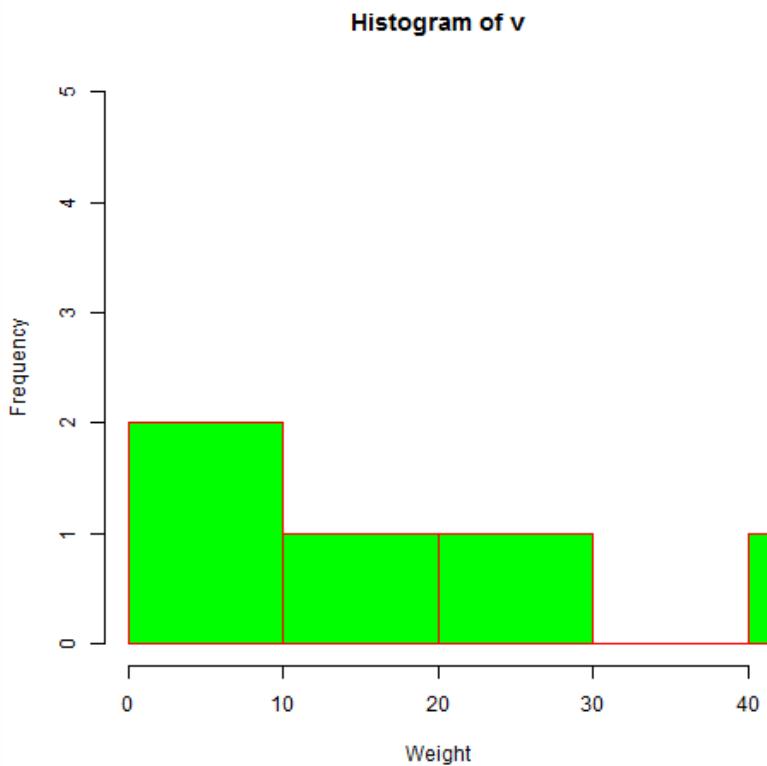
To specify the range of values allowed in X axis and Y axis, we can use the `xlim` and `ylim` parameters.

The width of each of the bar can be decided by using `breaks`.

```
# Create data for the graph.  
v <- c(9,13,21,8,36,22,12,41,31,33,19)  
  
# Give the chart file a name.  
png(file ="histogram_lim_breaks.png")  
# Create the histogram.  
hist(v,xlab ="Weight",col ="green",border ="red", xlim = c(0,40), ylim = c(0,5),  
breaks =5)  
  
# Save the file.
```

```
dev.off()
```

When we execute the above code, it produces the following result –



R - LINE GRAPHS

A line chart is a graph that connects a series of points by drawing line segments between them. These points are ordered in one of their coordinate usually the x-coordinate usually the x-coordinate value. Line charts are usually used in identifying the trends in data.

The **plot** function in R is used to create the line graph.

Syntax

The basic syntax to create a line chart in R is –

```
plot(v,type,col,xlab,ylab)
```

Following is the description of the parameters used –

- **v** is a vector containing the numeric values.

- **type** takes the value "p" to draw only the points, "l" to draw only the lines and "o" to draw both points and lines.
- **xlab** is the label for x axis.
- **ylab** is the label for y axis.
- **main** is the Title of the chart.
- **col** is used to give colors to both the points and lines.

Example

A simple line chart is created using the input vector and the type parameter as "O". The below script will create and save a line chart in the current R working directory.

```
# Create the data for the chart.
```

```
v <- c(7,12,28,3,41)
```

```
# Give the chart file a name.
```

```
png(file ="line_chart.jpg")
```

```
# Plot the bar chart.
```

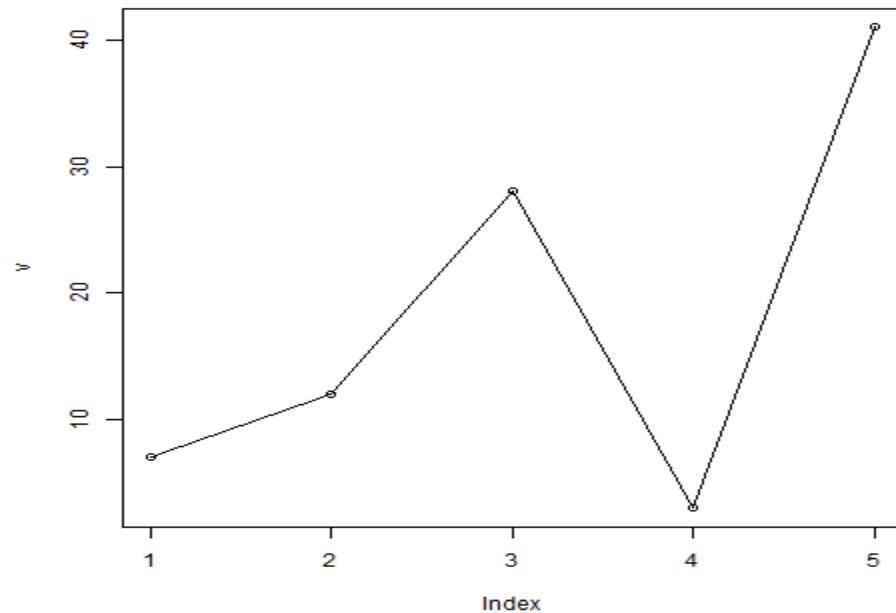
```
plot(v,type ="o")
```

```
# Save the file.
```

```
dev.off()
```

Hvderabad

When we execute the above code, it produces the following result –



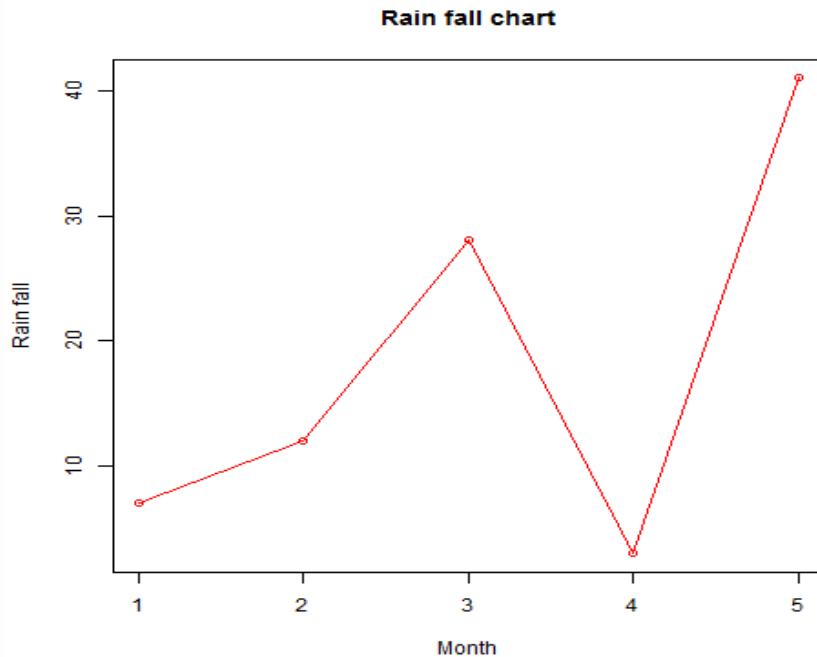
Line Chart Title, Color and Labels

The features of the line chart can be expanded by using additional parameters. We add color to the points and lines, give a title to the chart and add labels to the axes.

Example

```
9963799240 / 7730997544 /  
# Create the data for the chart.  
v <- c(7,12,28,3,41)  
  
# Give the chart file a name.  
png(file ="line_chart_label_colored.jpg")  
  
# Plot the bar chart.  
plot(v,type ="o", col ="red", xlab ="Month", ylab ="Rain fall",  
     main ="Rain fall chart")  
  
# Save the file.  
dev.off()
```

When we execute the above code, it produces the following result –



Multiple Lines in a Line Chart

More than one line can be drawn on the same chart by using the **lines** function.

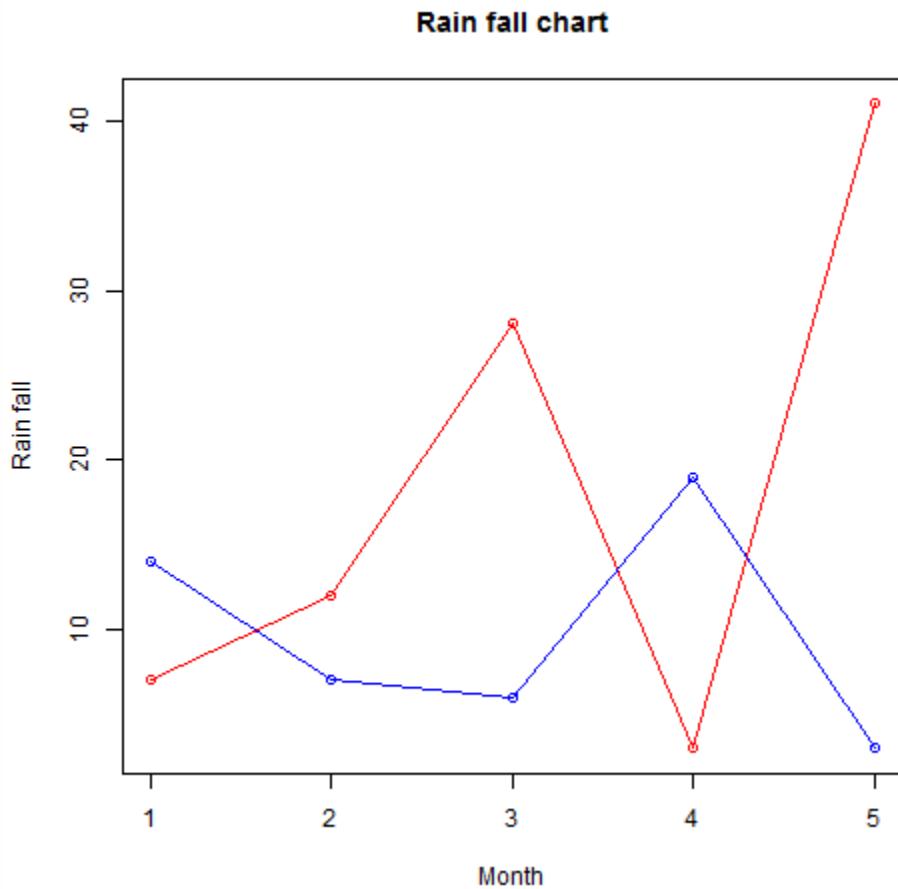
After the first line is plotted, the **lines** function can use an additional vector as input to draw the second line in the chart,

```
# Create the data for the chart.  
v <- c(7,12,28,3,41)  
t <- c(14,7,6,19,3)  
  
# Give the chart file a name.  
png(file ="line_chart_2_lines.jpg")  
  
# Plot the bar chart.  
plot(v,type ="o",col ="red", xlab ="Month", ylab ="Rain fall",  
     main ="Rain fall chart")  
lines(t, type ="o", col ="blue")
```

```
# Save the file.
```

```
dev.off()
```

When we execute the above code, it produces the following result –



R - SCATTERPLOTS

Scatterplots show many points plotted in the Cartesian plane. Each point represents the values of two variables. One variable is chosen in the horizontal axis and another in the vertical axis.

The simple scatterplot is created using the **plot** function.

Syntax

The basic syntax for creating scatterplot in R is –

```
plot(x, y, main, xlab, ylab, xlim, ylim, axes)
```

Following is the description of the parameters used –

- **x** is the data set whose values are the horizontal coordinates.
- **y** is the data set whose values are the vertical coordinates.
- **main** is the title of the graph.
- **xlab** is the label in the horizontal axis.
- **ylab** is the label in the vertical axis.
- **xlim** is the limits of the values of x used for plotting.
- **ylim** is the limits of the values of y used for plotting.
- **axes** indicates whether both axes should be drawn on the plot.

Example

We use the data set "**mtcars**" available in the R environment to create a basic scatterplot. Let's use the columns "wt" and "mpg" in mtcars.

```
input <- mtcars[,c('wt','mpg')]  
print(head(input))
```

When we execute the above code, it produces the following result –

	wt	mpg
Mazda RX4	2.620	21.0
Mazda RX4 Wag	2.875	21.0
Datsun 710	2.320	22.8
Hornet 4 Drive	3.215	21.4
Hornet Sportabout	3.440	18.7
Valiant	3.460	18.1

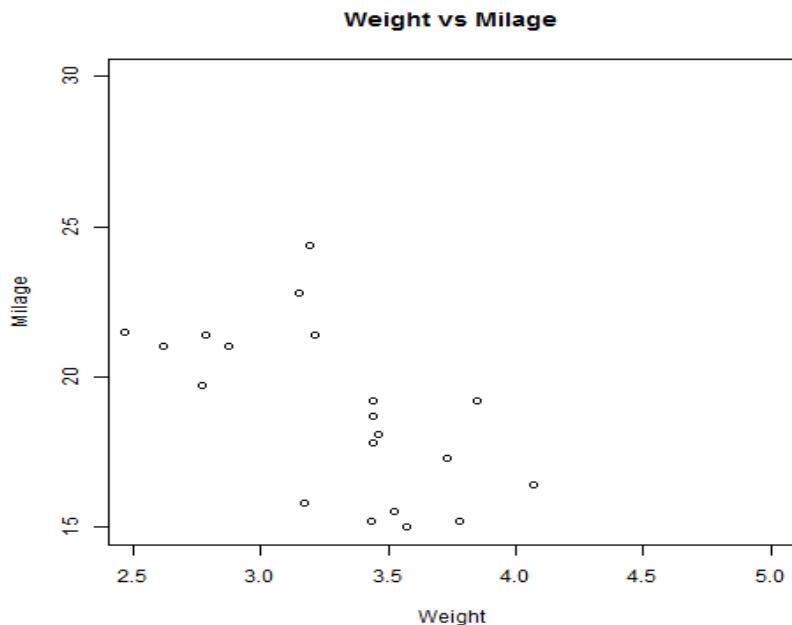
Creating the Scatterplot

The below script will create a scatterplot graph for the relation between wtweight and mpgmilespergallonmilespergallon.

```
# Get the input values.  
input <- mtcars[,c('wt','mpg')]  
# Give the chart file a name.  
png(file ="scatterplot.png")  
# Plot the chart for cars with weight between 2.5 to 5 and mileage between 15 and 30.  
plot(x = input$wt,y = input$mpg,  
      xlab ="Weight",
```

```
ylab = "Milage",  
xlim = c(2.5,5),  
ylim = c(15,30),  
main ="Weight vs Milage"  
}  
  
# Save the file.  
dev.off()
```

When we execute the above code, it produces the following result –



Scatterplot Matrices

When we have more than two variables and we want to find the correlation between one variable versus the remaining ones we use scatterplot matrix. We use **pairs** function to create matrices of scatterplots.

Syntax

The basic syntax for creating scatterplot matrices in R is –

```
pairs(formula, data)
```

Following is the description of the parameters used –

- **formula** represents the series of variables used in pairs.
- **data** represents the data set from which the variables will be taken.

Example

Each variable is paired up with each of the remaining variable. A scatterplot is plotted for each pair.

```
# Give the chart file a name.  
png(file ="scatterplot_matrices.png")  
# Plot the matrices between 4 variables giving 12 plots.  
# One variable with 3 others and total 4 variables.  
pairs(~wt+mpg+disp+cyl,data = mtcars,  
      main ="Scatterplot Matrix")  
# Save the file.  
dev.off()
```

When the above code is executed we get the following output.

