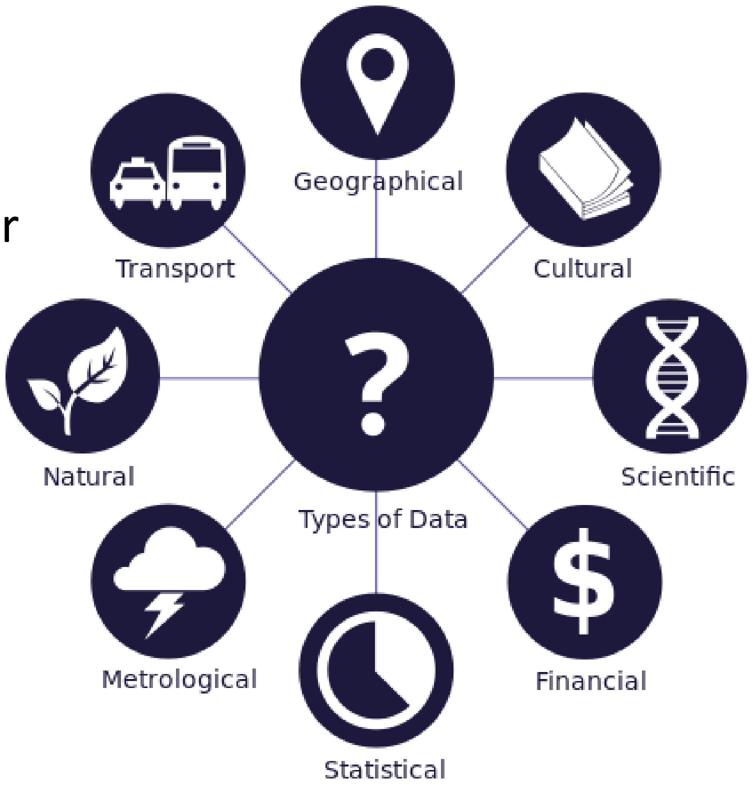
Data Analytics & M/c Learning using R programming Part-1

By
Lokesh Singh
@SCTPL

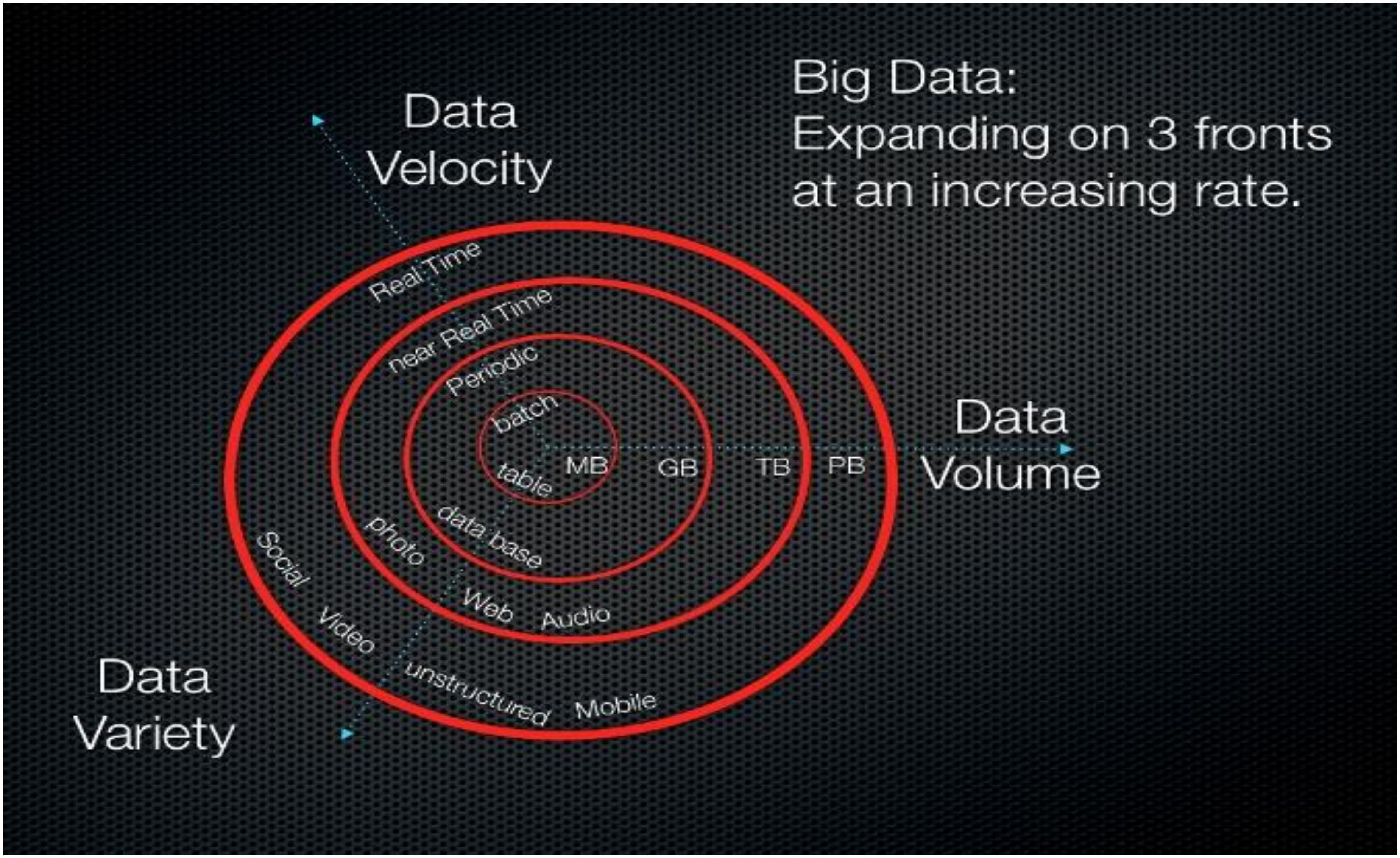
what is data

Data is a set of values of qualitative or quantitative variables.
An example of qualitative data would be an anthropologist's handwritten notes about his or her interviews with indigenous people.

Pieces of data are individual pieces of information. While the concept of data is commonly associated with scientific research, data is collected by a huge range of organizations and institutions, including businesses (e.g., sales data, revenue, profits, stock price), governments (e.g., crime rates, unemployment rates, literacy rates) and non-governmental organizations (e.g., censuses of the number of homeless people by non-profit organizations).



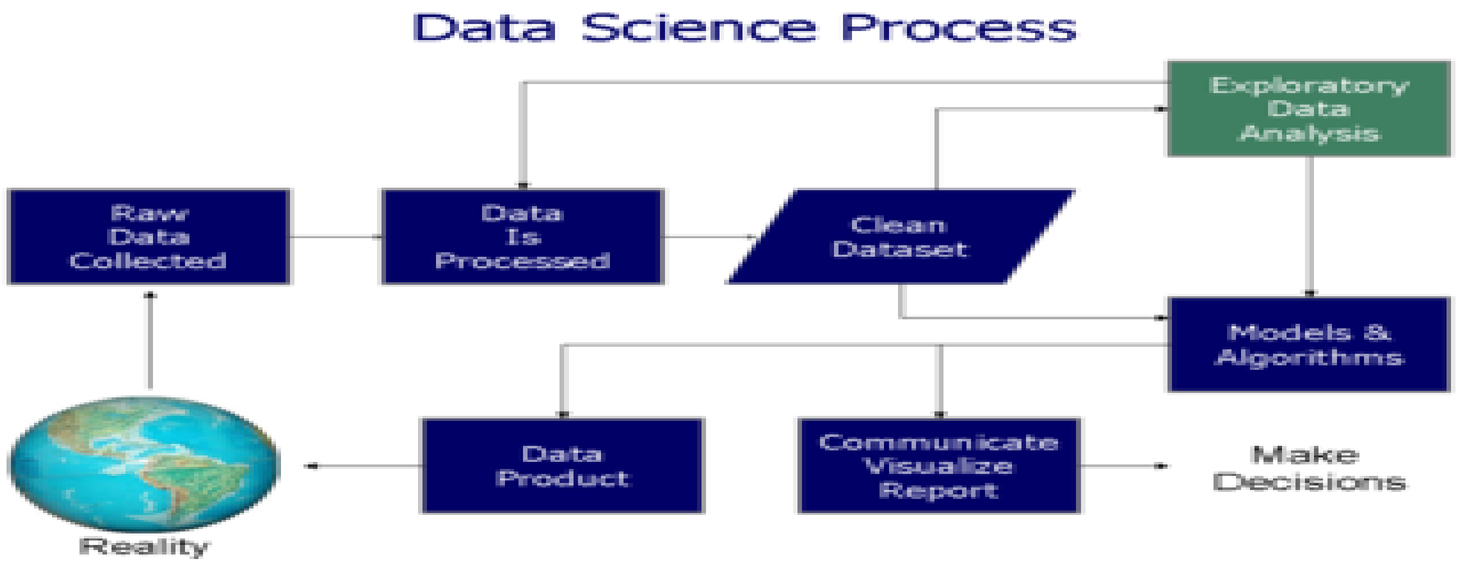
Data Types


Big Data:
Expanding on 3 fronts at an increasing rate.
Data Velocity
Real Time
near Real Time
Periodic
batch
Data Volume
MB
GB
TB
PB
Data Variety
table
data base
photo
Web
Audio
Social
Video
unstructured
Mobile

What is Data analysis

Data analysis is a primary component of data mining and Business Intelligence (BI) and is key to gaining the insight that drives business decisions.
Organizations and enterprises analyze data from a multitude of sources using Big Data management solutions and customer experience management solutions that utilize data analysis to transform data into actionable insights.

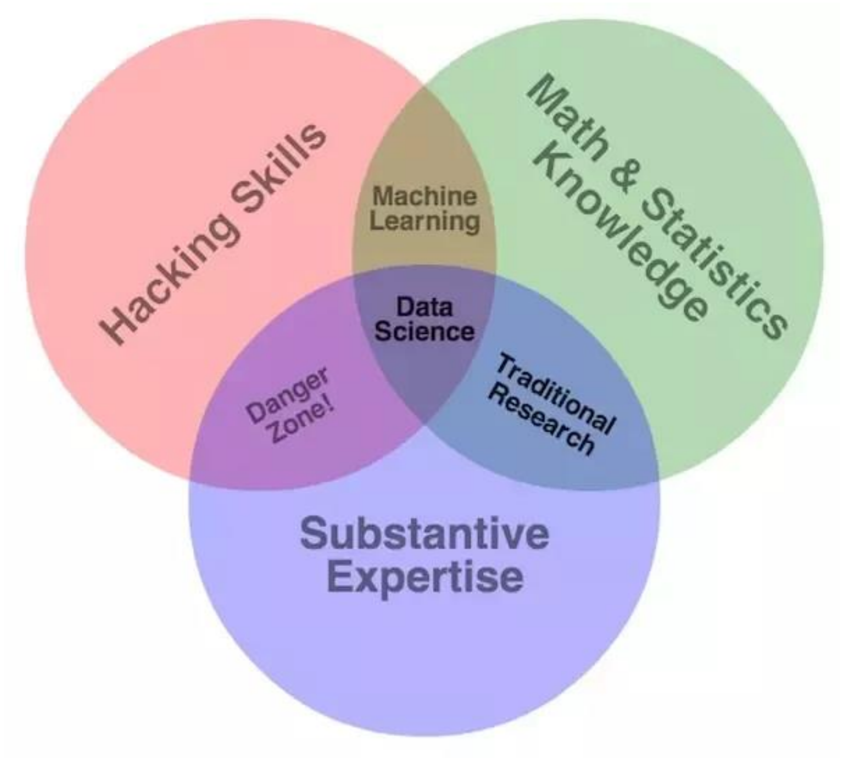
Data analysis, also known as **analysis of data** or **data analytics**, is a process of inspecting, cleansing, transforming, and modeling data with the goal of discovering useful information, suggesting conclusions, and supporting decision-making. Data analysis has multiple facets and approaches, encompassing diverse techniques under a variety of names, in different business, science, and social science domains.



What is Data Science

Data science is a "concept to unify statistics, data analysis and their related methods" in order to "understand and analyze actual phenomena" with data.[3] It employs techniques and theories drawn from many fields within the broad areas of mathematics, statistics, information science, and computer science, in particular from the subdomains of machine learning, classification, cluster analysis, data mining, databases, and visualization.

Data science, also known as **data-driven science**, is an interdisciplinary field about scientific methods, processes, and systems to extract knowledge or insights from data in various forms, either structured or unstructured , similar to data mining.



What is difference between data analysis and data science

Data Scientist – Rock Star of IT

A Data Scientist is a professional who understands data from a business point of view. He is in charge of making predictions to help businesses take accurate decisions.
Data scientists come with a solid foundation of computer applications, modeling, statistics and math. What sets them apart is their brilliance in business coupled with great communication skills, to deal with both business and IT leaders.
They are efficient in picking the right problems, which will add value to the organization after resolving it.
Harvard Business Review has named 'Data scientist' as the "sexiest job of the 21st century. Up-skill with Data Science now to take advantage of the career opportunities that come your way.

Data Analysts – No Cool Tag Yet!

Data Analysts also plays a major role in Data Science. They perform a variety of tasks related to collecting, organizing data and obtaining statistical information out of them.
They are also responsible to present the data in the form of charts, graphs and tables and use the same to build relational databases for organizations.

Qualification Required for Data Scientists and Data Analysts

Data Scientist

- They should be familiar with database systems. Example: MySQL, Hive etc.
- Better to also be familiar with Java, Python, MapReduce job developments.
- Should have clear understanding of various analytical functions – median, rank etc and how to use them on data sets.
- Perfection in mathematics, statistics, correlation, data mining and predictive analysis better predictions for business decisions.
- Knowing 'R' is like a feather on a Data Scientist's cap.
- Deep statistical insights and machine learning – Mahout, Bayesian,Clustering etc

Data Analysts

- Familiarity with data warehousing and business intelligence concepts.
- In-depth exposure of SQL and analytics
- Strong understanding of Hadoop based analytics (HBase, Hive, MapReduce jobs, Impada, Casscading etc)
- Data storing and retrieving skills and tools
- Perfect with the tools and components of data architecture
- Familiar with various ETL tools - for transforming different sources of data into analytics data stores. Should also be able to make some critical business features real time..
- Proficiency in decision making

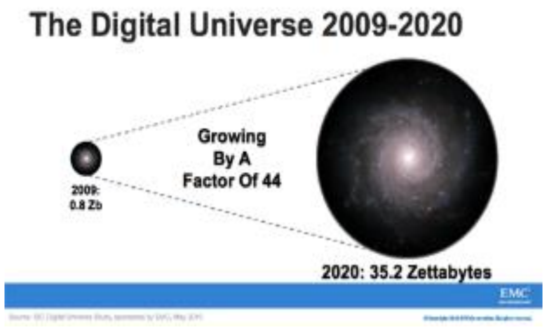
Big Data Definition

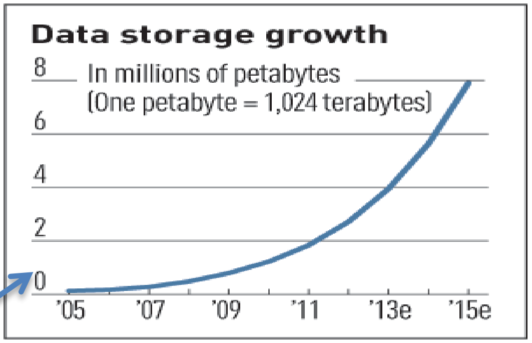
- No single standard definition…

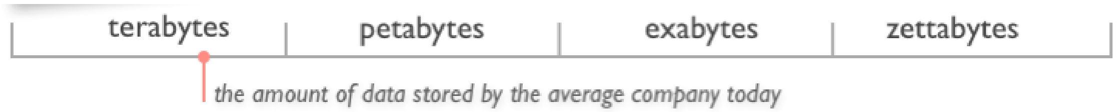
"***Big Data***" is data whose scale, diversity, and complexity require new architecture, techniques, algorithms, and analytics to manage it and extract value and hidden knowledge from it…

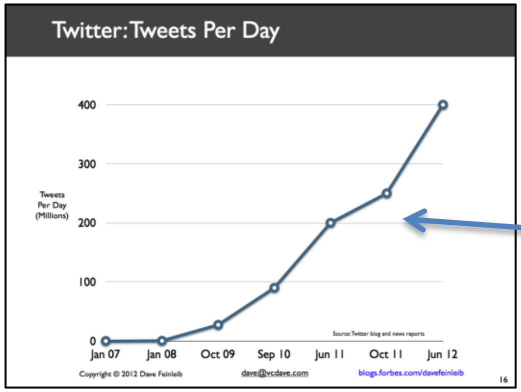
Characteristics of Big Data: 1-Scale (Volume)

- **Data Volume**
  - 44x increase from 2009
  - From 0.8 zettabytes to 35zb
- Data volume is increasing exponentially





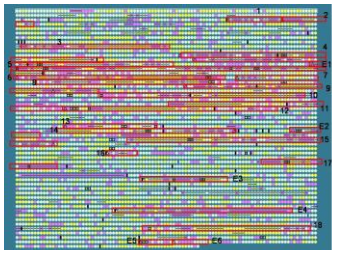
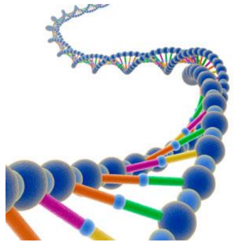
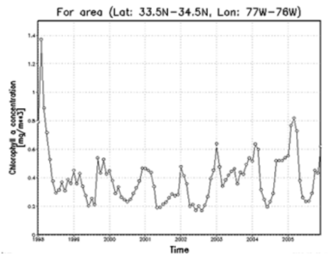


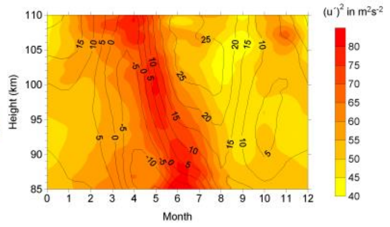


Exponential increase in collected/generated data

Characteristics of Big Data: 2-Complexity (Variety)

- Various formats, types, and structures
- Text, numerical, images, audio, video, sequences, time series, social media data, multi-dim arrays, etc…
- Static data vs. streaming data
- A single application can be generating/collecting many types of data

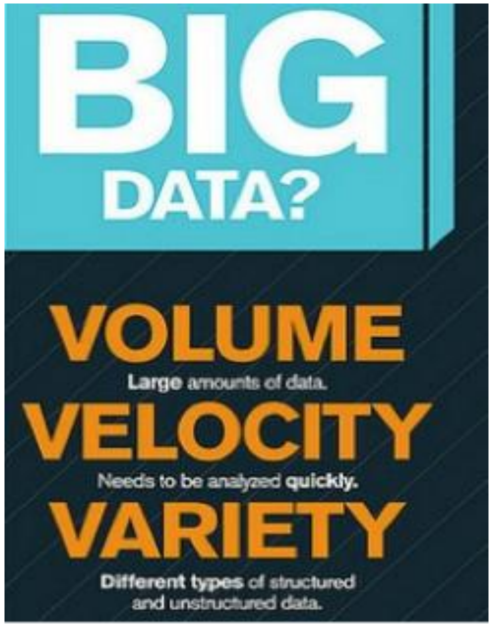


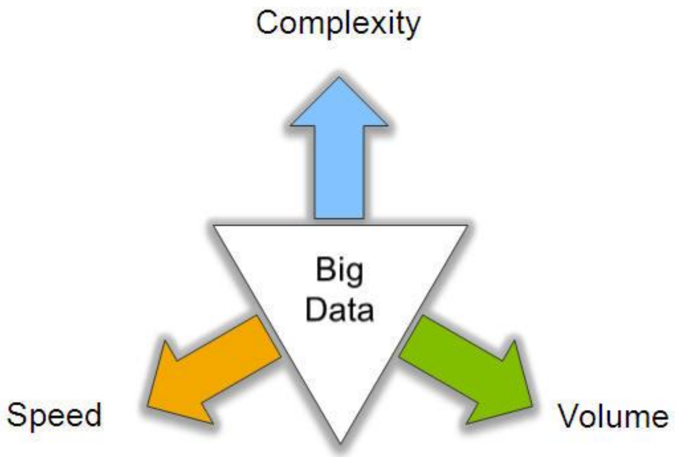
To extract knowledge➔ all these types of data need to linked together

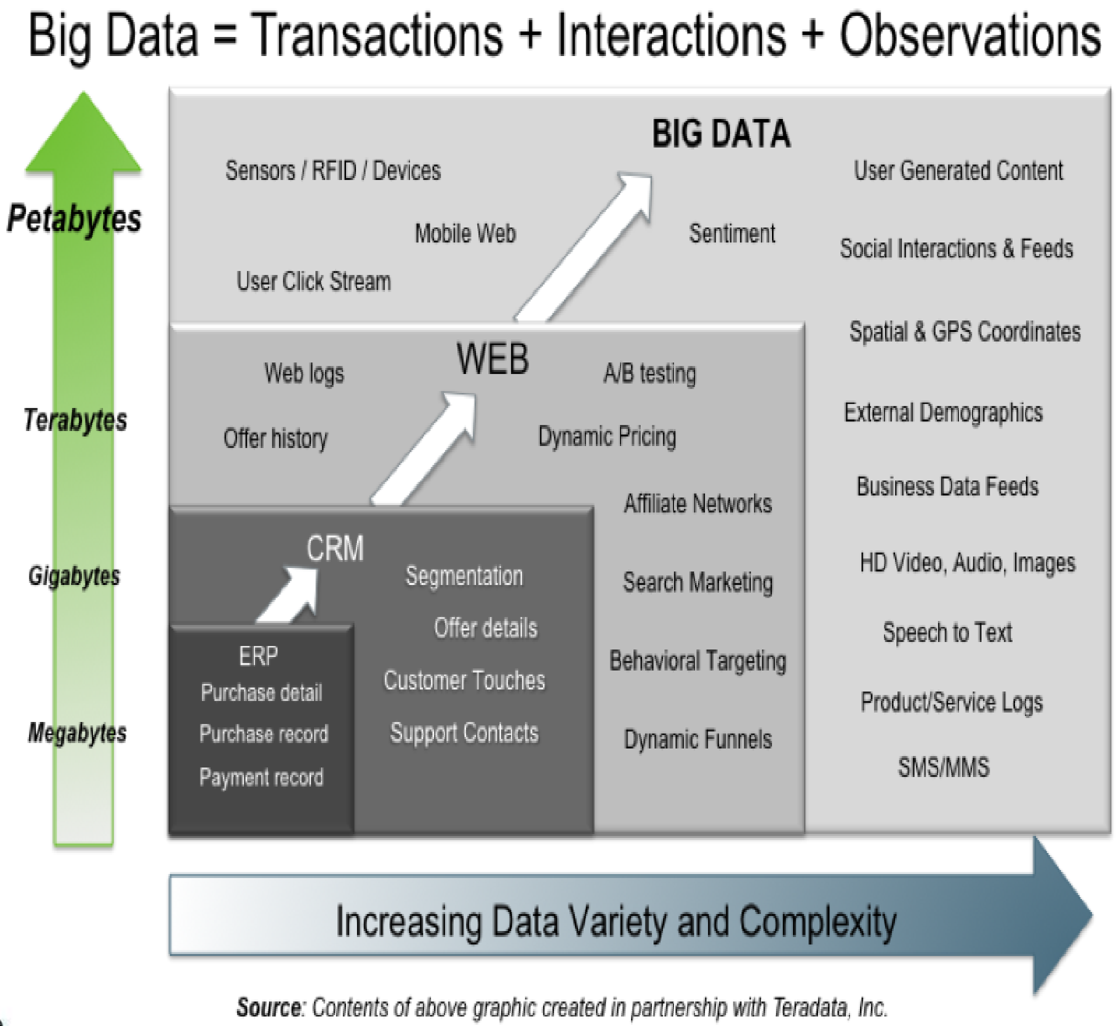
Characteristics of Big Data: 3-Speed (Velocity)

- Data is begin generated fast and need to be processed fast
- Online Data Analytics
- Late decisions ➔ missing opportunities
- **Examples**
  - **E-Promotions:** Based on your current location, your purchase history, what you like ➔ send promotions right now from store next to you.
  - **Healthcare monitoring:** sensors monitoring your activities and body ➔ any abnormal measurements require immediate reaction.

Big Data: 3V's

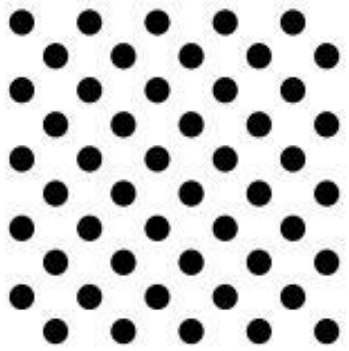

BIG DATA?
VOLUME
Large amounts of data.
VELOCITY
Needs to be analyzed quickly.
VARIETY
Different types of structured and unstructured data.


Complexity
Big Data
Speed
Volume


Big Data = Transactions + Interactions + Observations
Petabytes
Terabytes
Gigabytes
Megabytes
BIG DATA
Sensors / RFID / Devices
Mobile Web
User Click Stream
Sentiment
User Generated Content
Social Interactions & Feeds
Spatial & GPS Coordinates
External Demographics
Business Data Feeds
HD Video, Audio, Images
Speech to Text
Product/Service Logs
SMS/MMS
WEB
Web logs
Offer history
A/B testing
Dynamic Pricing
Affiliate Networks
Search Marketing
Behavioral Targeting
Dynamic Funnels
CRM
Segmentation
Offer details
Customer Touches
Support Contacts
ERP
Purchase detail
Purchase record
Payment record
Increasing Data Variety and Complexity
Source: Contents of above graphic created in partnership with Teradata, Inc.

Some Make it 4V's

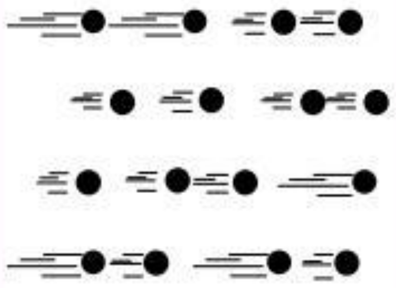
Volume



Data at Rest

Terabytes to exabytes of existing data to process

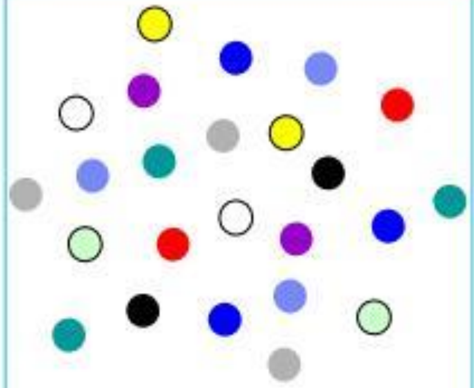
Velocity



Data in Motion

Streaming data, milliseconds to seconds to respond

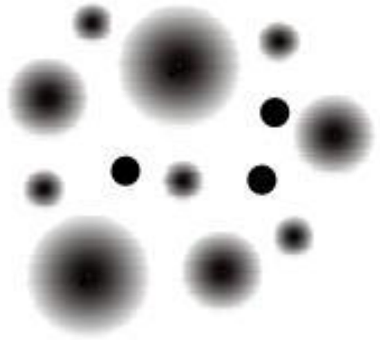
Variety



Data in Many Forms

Structured, unstructured, text, multimedia

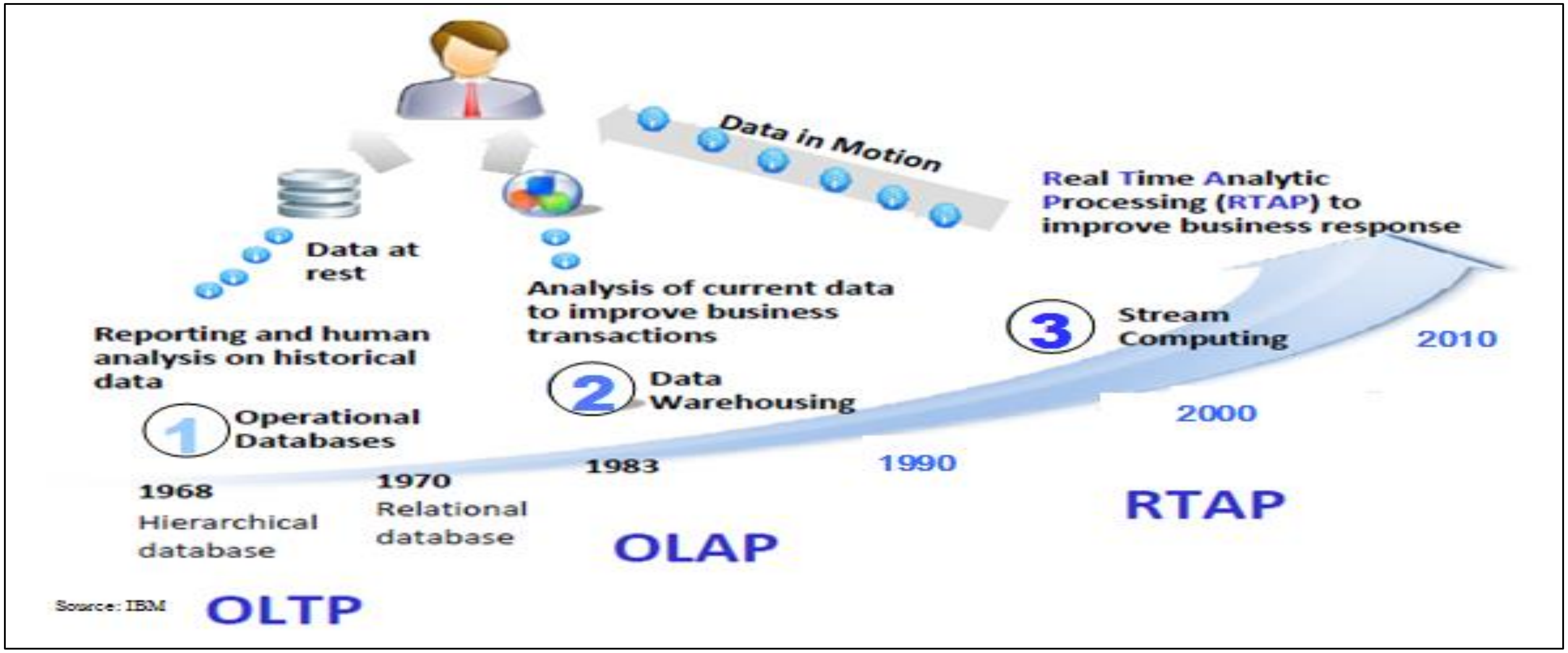
Veracity*



Data in Doubt

Uncertainty due to data inconsistency & incompleteness, ambiguities, latency, deception, model approximations

Harnessing Big Data


Data in Motion
Real Time Analytic Processing (RTAP) to improve business response
Data at rest
Analysis of current data to improve business transactions
Reporting and human analysis on historical data
3 Stream Computing
2010
2 Data Warehousing
2000
1 Operational Databases
1983
1990
1968 Hierarchical database
1970 Relational database
OLAP
RTAP
Source: IBM
OLTP

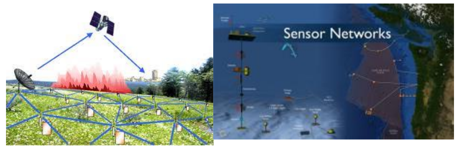
- **OLTP:** Online Transaction Processing (DBMSs)
- **OLAP:** Online Analytical Processing (Data Warehousing)
- **RTAP:** Real-Time Analytics Processing (Big Data Architecture & technology)

Who's Generating Big Data

Social media and networks
(all of us are generating data)

Scientific instruments
(collecting all sorts of data)

Mobile devices
(tracking all objects all the time)



Sensor technology and networks
(measuring all kinds of data)

- The progress and innovation is no longer hindered by the ability to collect data
- But, by the ability to manage, analyze, summarize, visualize, and discover knowledge from the collected data in a timely manner and in a scalable fashion

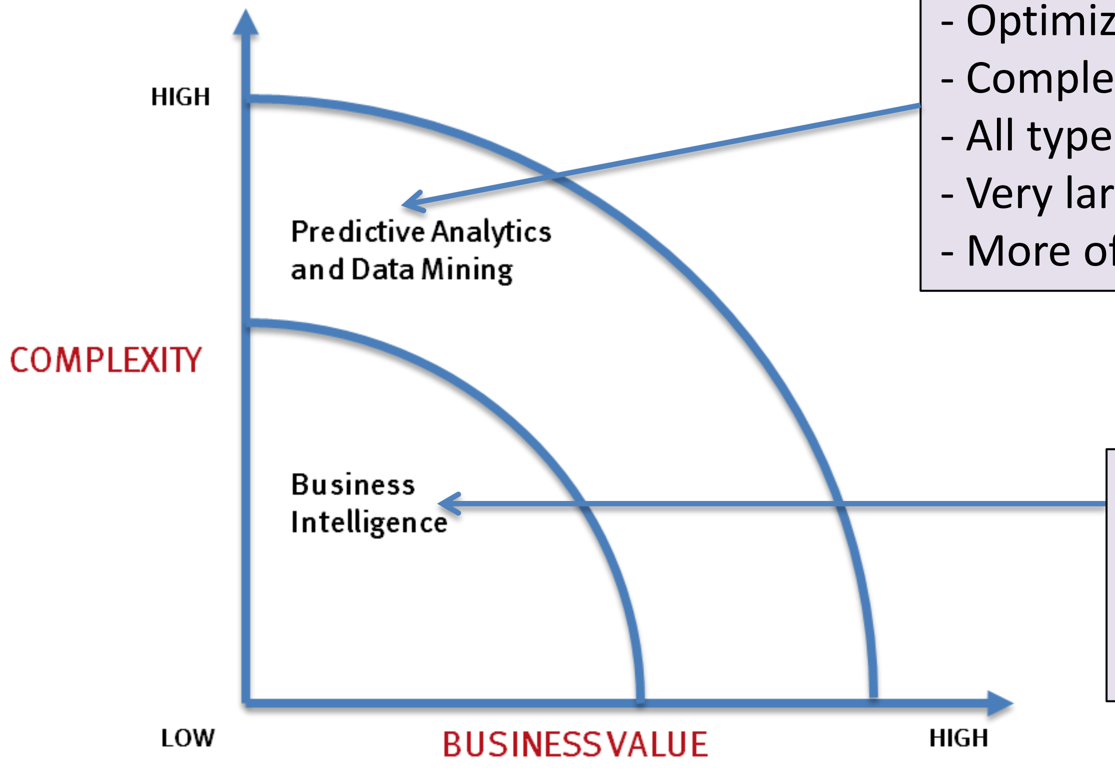
The Model Has Changed…

- **The Model of Generating/Consuming Data has Changed**

Old Model: Few companies are generating data, all others are consuming data

New Model: all of us are generating data, and all of us are consuming data

What's driving Big Data

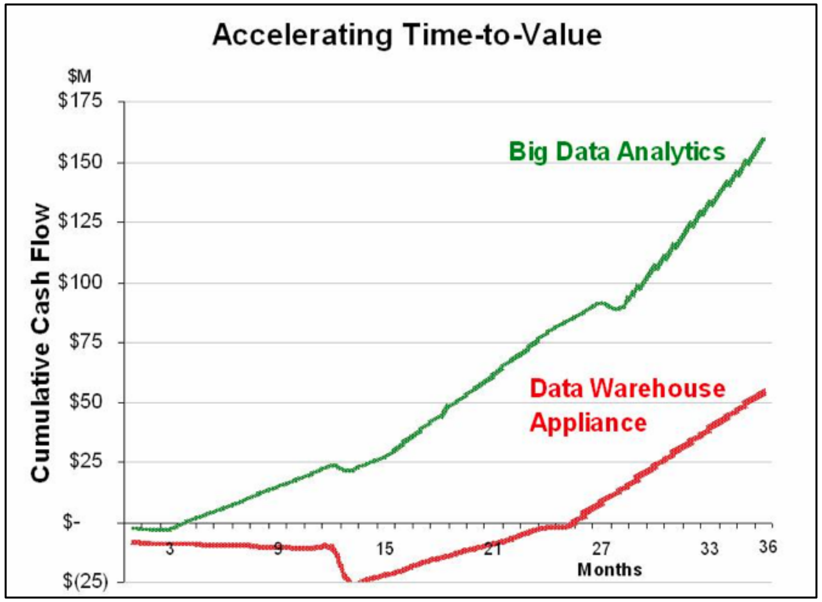


- Optimizations and predictive analytics
- Complex statistical analysis
- All types of data, and many sources
- Very large datasets
- More of a real-time

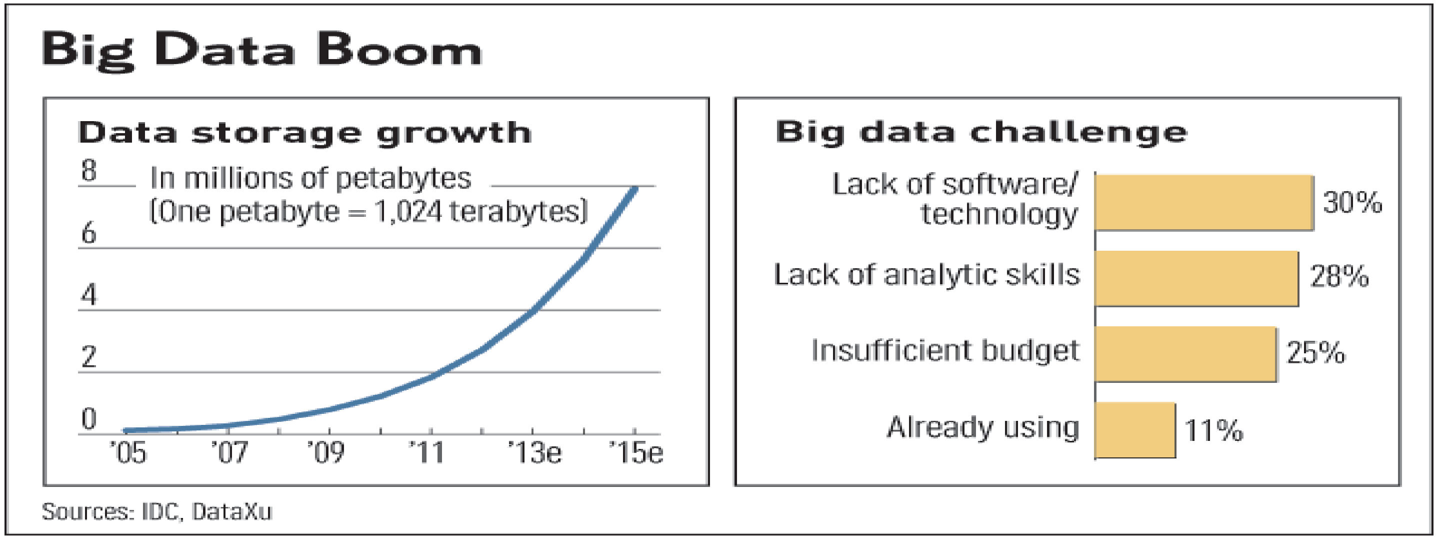
- Ad-hoc querying and reporting
- Data mining techniques
- Structured data, typical sources
- Small to mid-size datasets

Value of Big Data Analytics

- Big data is more real-time in nature than traditional DW applications
- Traditional DW architectures (e.g. Exadata, Teradata) are not well-suited for big data apps
- Shared nothing, massively parallel processing, scale out architectures are well-suited for big data apps



Challenges in Handling Big Data



- **The Bottleneck is in technology**
  - New architecture, algorithms, techniques are needed
- **Also in technical skills**
  - Experts in using the new technology and dealing with big data

What Technology Do We Have For Big Data ??

Big Data Landscape

Vertical Apps

PREDICTIVE POLICING
bloomreach GET FOUND.
MYRRIX

Log Data Apps

splunk>
loggly
sumologic

Ad/Media Apps

rocketfuel
collective[i]
bluefin LABS
Recorded Future
LuckySort
Media Science
TURN
DataXu Data. Insight. Action.

Data As A Service

kaggle
factual.
knoema beta
GNIP
DATASIFT
Windows Azure Marketplace
INRIX
LexisNexis
SPACE CURVE
LOCATE Everything Location

Business Intelligence

ORACLE | Hyperion
SAP
Business Objects
RJMetrics
Microsoft | Business Intelligence
IBM
COGNOS
birst
MicroStrategy
Autonomy
bime
DOMO
QlikView
Chart.io
GoodData

Analytics and Visualization

tableau
Palantir
OPERA
metaLayer
METAMARKETS
dataspora
centrifuge
TERADATA ASTER
sas
TIBCO
KARMASPHERE
panopticon Real-Time Visual Data Analysis
pentaho
Datameer
ClearStory DATA
CIRRO
platfora
alteryx
visual.ly
AYATA

Analytics Infrastructure

Hortonworks
VERTICA An HP Company
MAPR TECHNOLOGIES
cloudera
INFOBRIGHT
PARACCEL
EMC²
GREENPLUM.
NETEZZA
kognitio
DATASTAX
EXASOL
calpont

Operational Infrastructure

COUCHBASE
10gen | the MongoDB company
TERADATA
HADAPT
TERRACOTTA
VoltDB
MarkLogic
INFORMATICA

Infrastructure As A Service

amazon web services
Windows Azure
infochimps
Google BigQuery

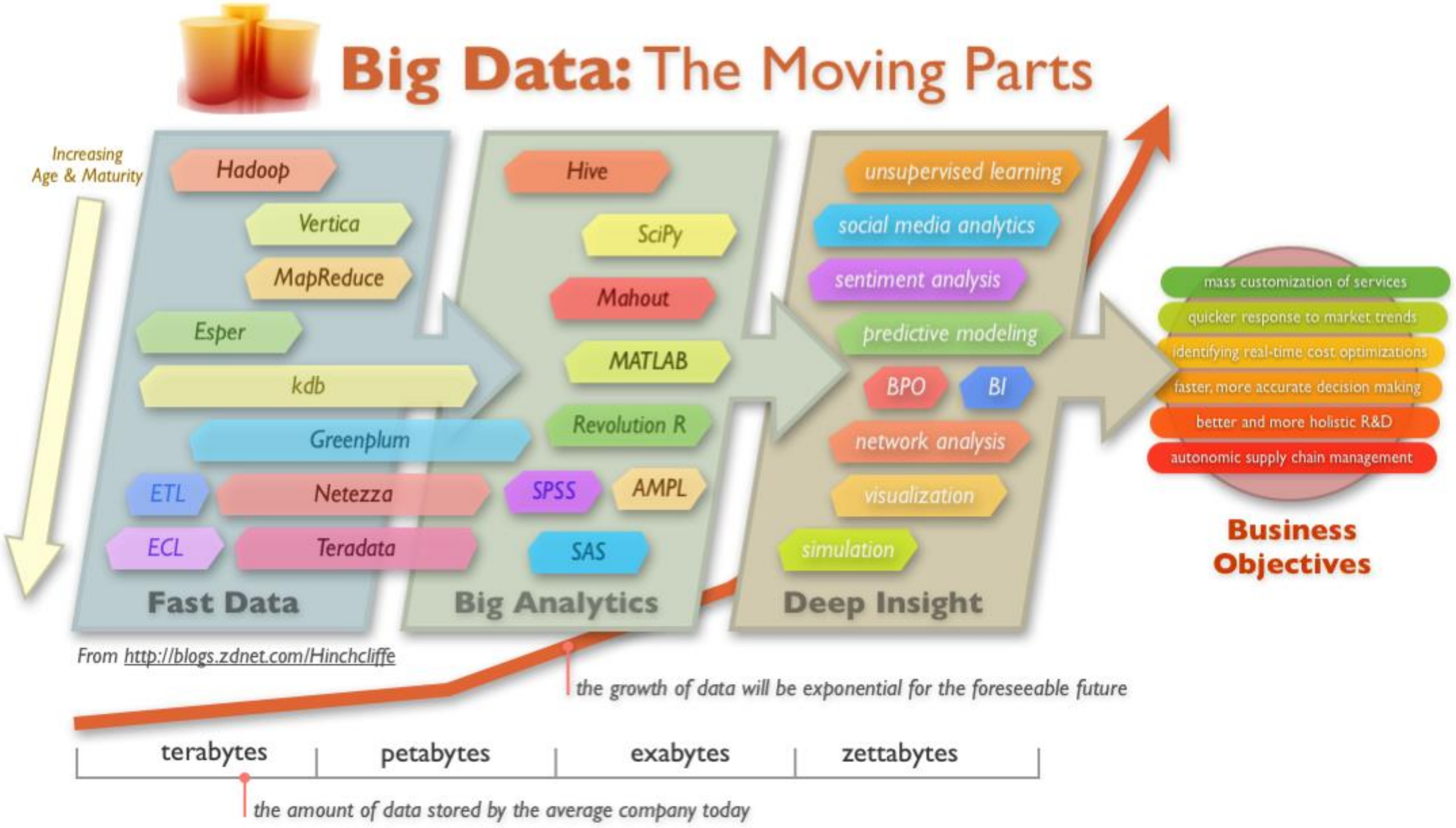
Structured Databases

ORACLE
MySQL
Microsoft SQL Server
PostgreSQL
IBM DB2.
SYBASE
memsql

Technologies

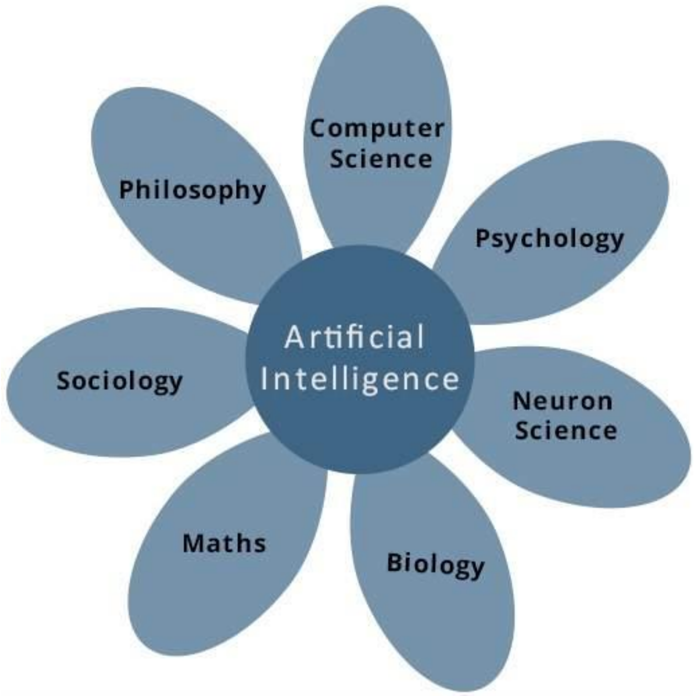
hadoop
hadoop Map Reduce
mahout
APACHE HBASE
Cassandra

Big Data Technology



Big Data: The Moving Parts
Increasing Age & Maturity
Hadoop
Vertica
MapReduce
Esper
kdb
Greenplum
ETL
Netezza
ECL
Teradata
Fast Data
Hive
SciPy
Mahout
MATLAB
Revolution R
SPSS
AMPL
SAS
Big Analytics
unsupervised learning
social media analytics
sentiment analysis
predictive modeling
BPO
BI
network analysis
visualization
simulation
Deep Insight
mass customization of services
quicker response to market trends
identifying real-time cost optimizations
faster, more accurate decision making
better and more holistic R&D
autonomic supply chain management
Business Objectives
From http://blogs.zdnet.com/Hinchcliffe
the growth of data will be exponential for the foreseeable future
terabytes
petabytes
exabytes
zettabytes
the amount of data stored by the average company today

What is Artificial Intelligence?



According to the father of Artificial Intelligence, John McCarthy, it is *"The science and engineering of making intelligent machines, especially intelligent computer programs".*
Artificial Intelligence is a way of **making a computer, a computer-controlled robot, or a software think intelligently**, in the similar manner the intelligent humans think.
AI is accomplished by studying how human brain thinks, and how humans learn, decide, and work while trying to solve a problem, and then using the outcomes of this study as a basis of developing intelligent software and systems.

Goals of AI

•**To Create Expert Systems** – The systems which exhibit intelligent behavior, learn, demonstrate, explain, and advice its users.

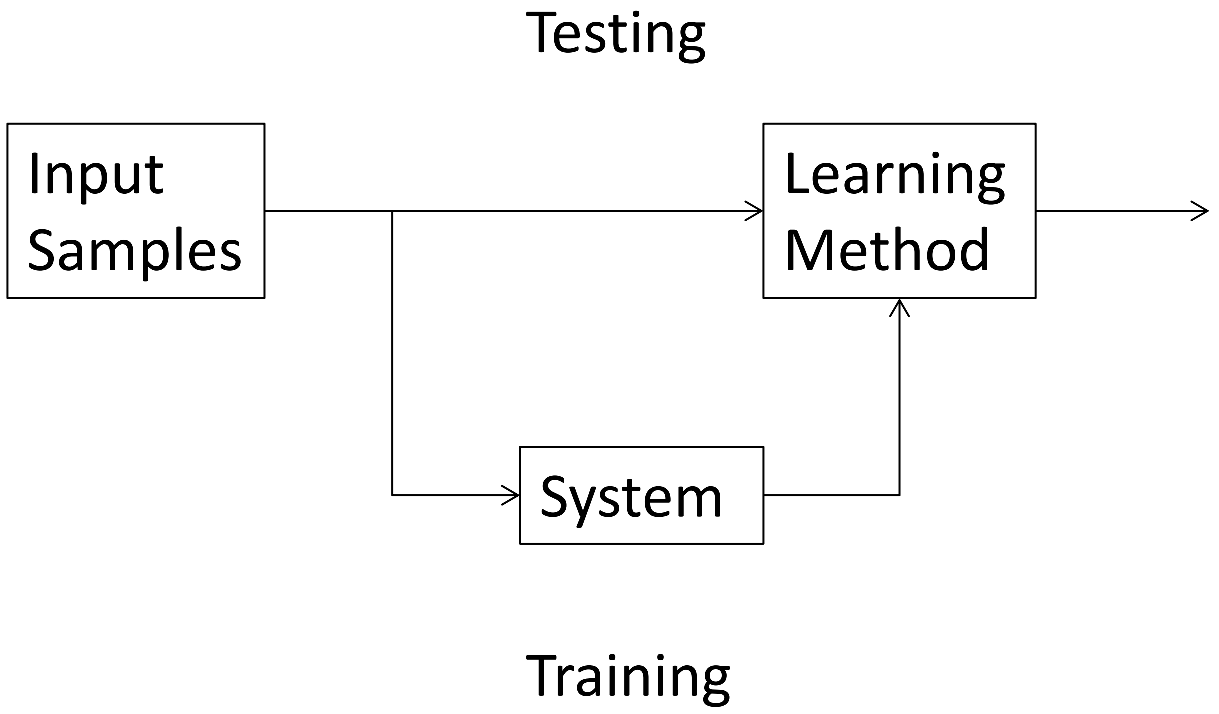
•**To Implement Human Intelligence in Machines** – Creating systems that understand, think, learn, and behave like humans.

What is machine learning?

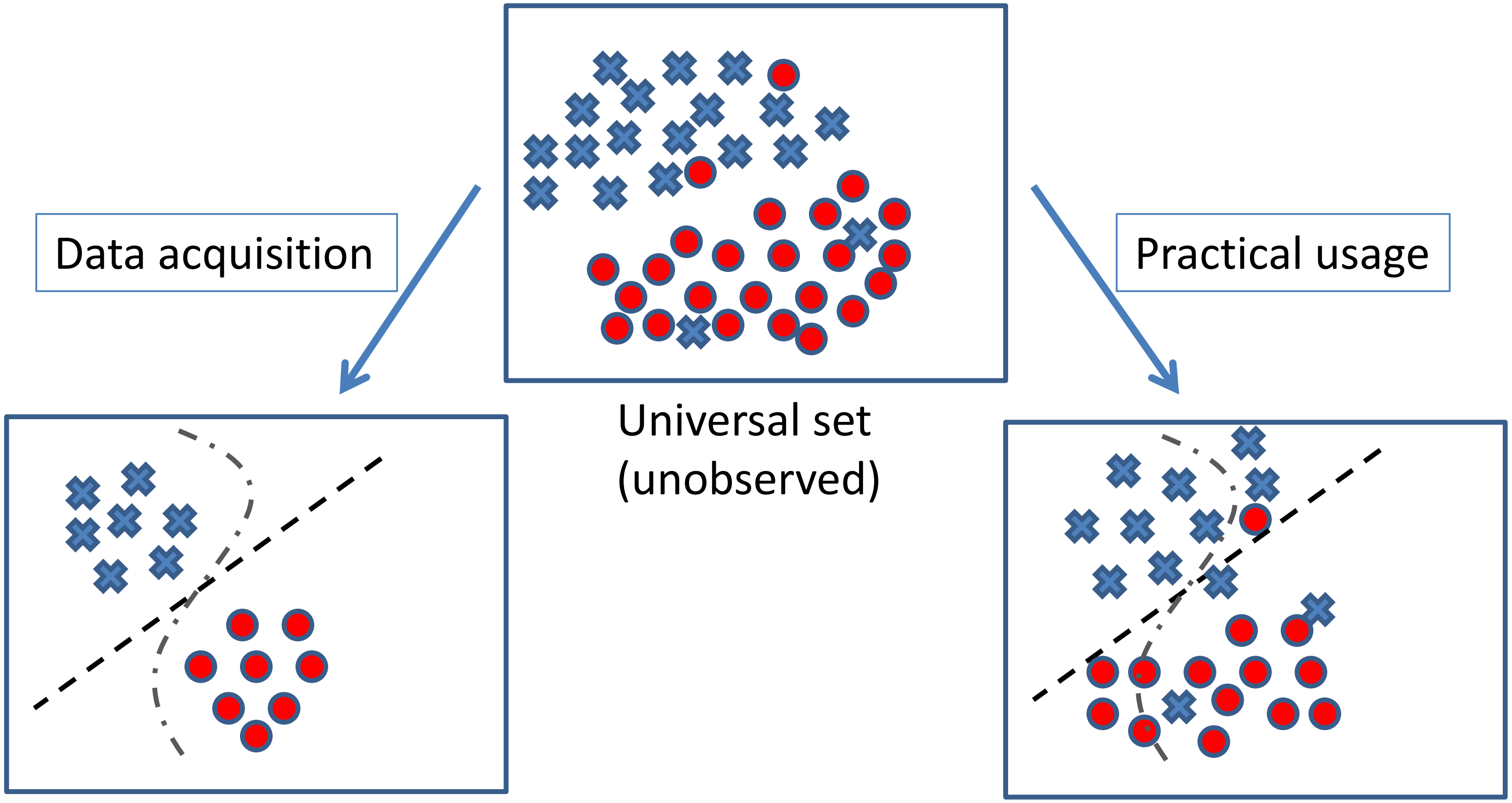
- A branch of **artificial intelligence**, concerned with the design and development of algorithms that allow computers to evolve behaviors based on empirical data.

- As intelligence requires knowledge, it is necessary for the computers to acquire knowledge.

Learning system model



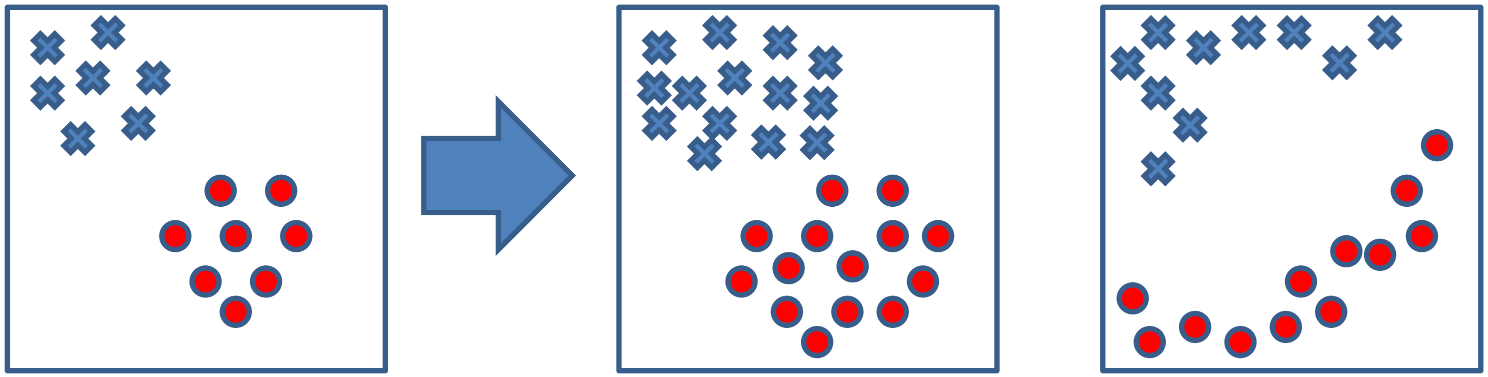
Training and testing



Training and testing

- Training is the process of making the system able to learn.

- No free lunch rule:
  - Training set and testing set come from the same distribution
  - Need to make some assumptions or bias



Performance

- There are several factors affecting the performance:
  - **Types of training** provided
  - The form and extent of any initial **background knowledge**
  - The **type of feedback** provided
  - The **learning algorithms** used

- Two important factors:
  - Modeling
  - Optimization

Algorithms

- The success of machine learning system also depends on the algorithms.

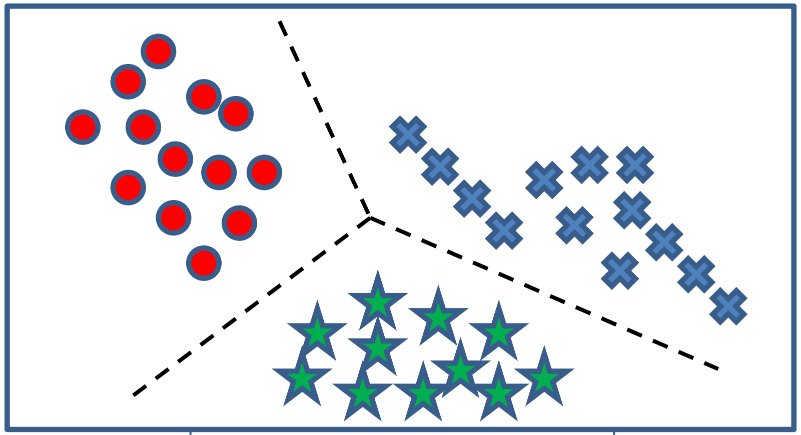
- The algorithms control the search to find and build the knowledge structures.

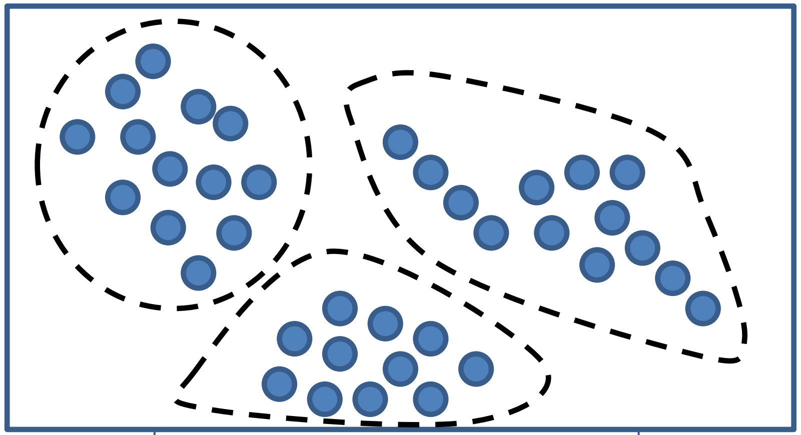
- The learning algorithms should extract useful information from training examples.

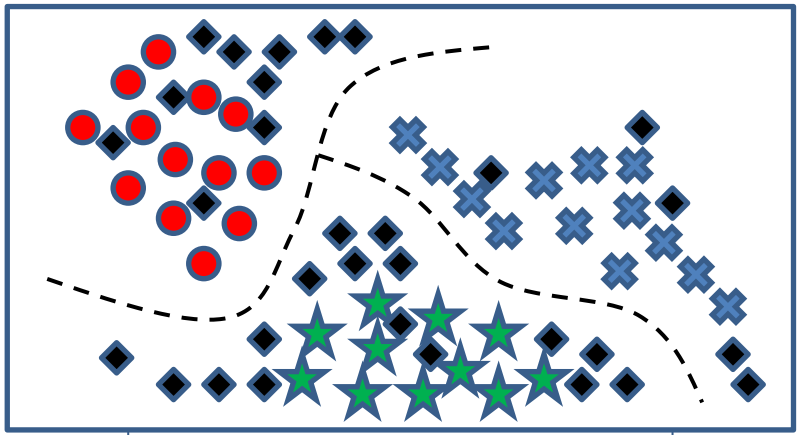
Algorithms

- **Supervised learning**
  - Prediction
  - Classification (discrete labels), Regression (real values)
- **Unsupervised learning**
  - Clustering
  - Probability distribution estimation
  - Finding association (in features)
  - Dimension reduction
- **Semi-supervised learning**
- **Reinforcement learning**
  - Decision making (robot, chess machine)

Algorithms

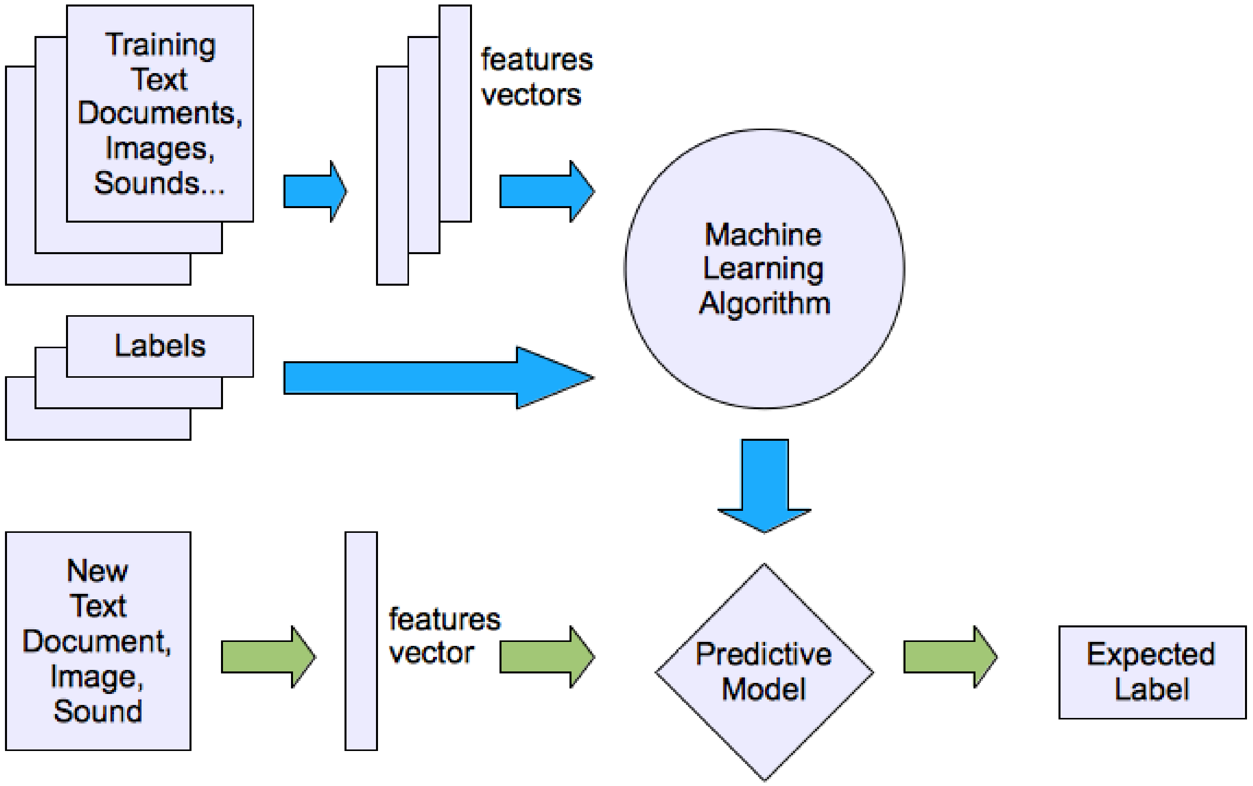

Supervised learning


Unsupervised learning


Semi-supervised learning

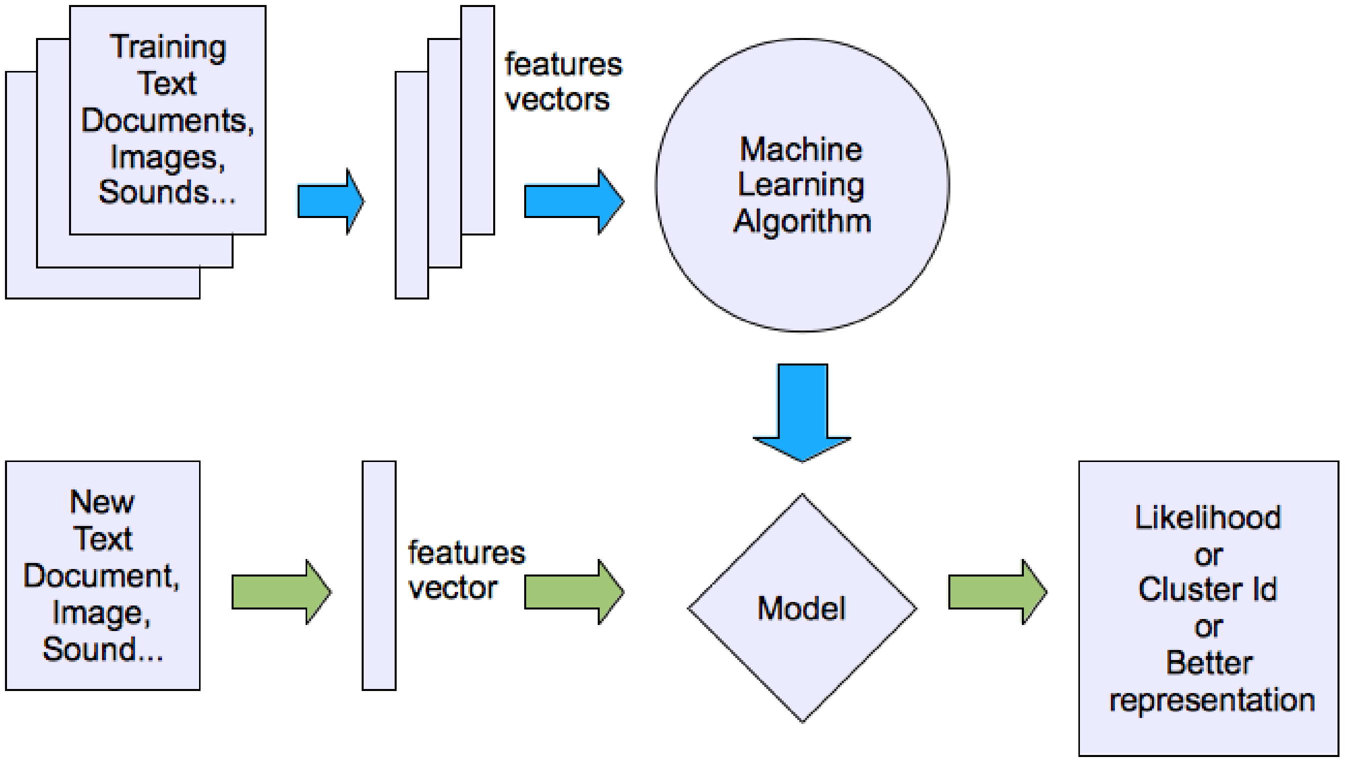
Machine learning structure

- Supervised learning

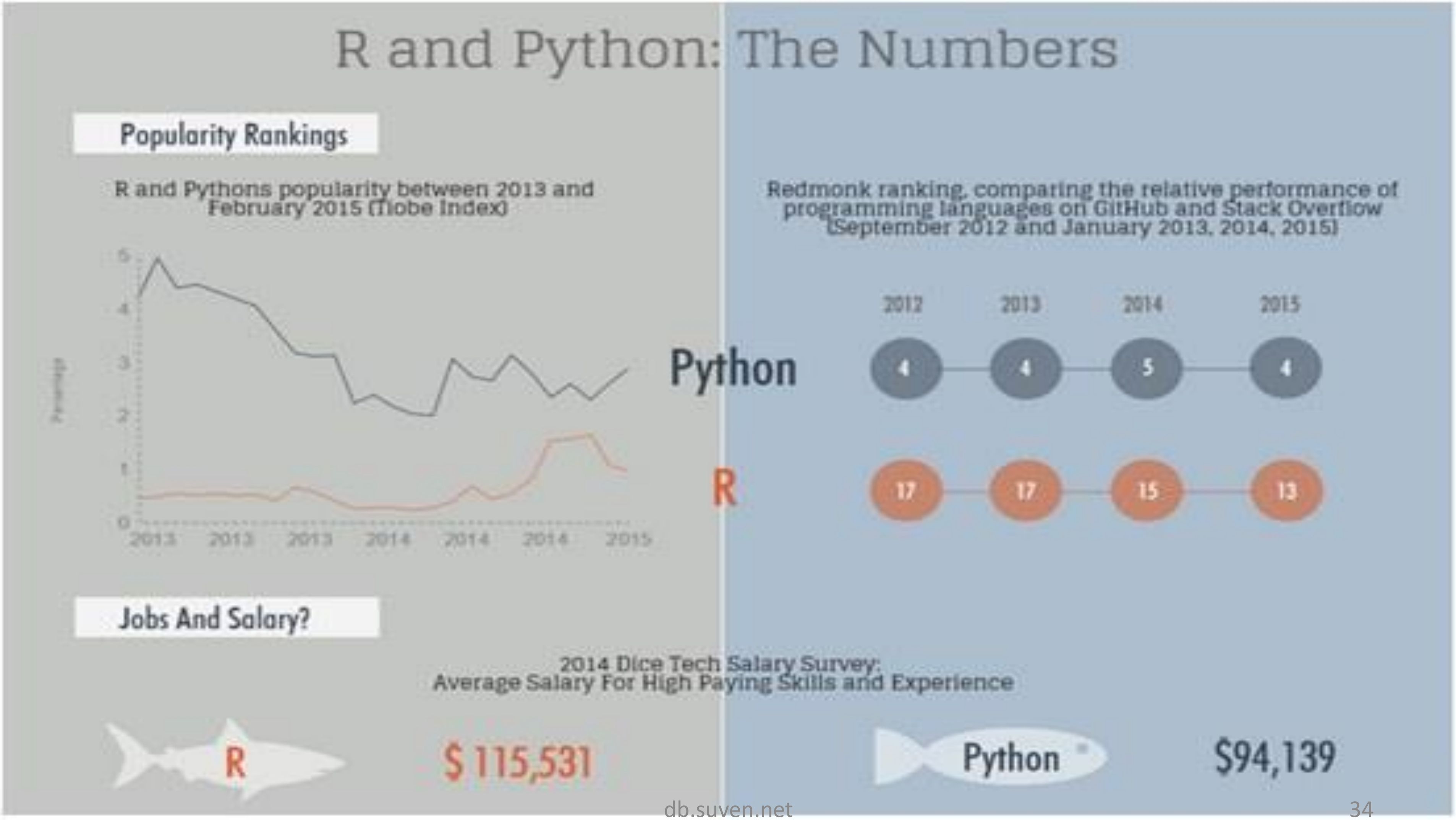

Training Text Documents, Images, Sounds...
features vectors
Labels
Machine Learning Algorithm
New Text Document, Image, Sound
features vector
Predictive Model
Expected Label

Machine learning structure

- Unsupervised learning


Training Text Documents, Images, Sounds...
features vectors
Machine Learning Algorithm
New Text Document, Image, Sound...
features vector
Model
Likelihood or Cluster Id or Better representation

R And Python



R and Python: The Numbers

Popularity Rankings

R and Pythons popularity between 2013 and February 2015 (Tiobe Index)

Redmonk ranking, comparing the relative performance of programming languages on GitHub and Stack Overflow (September 2012 and January 2013, 2014, 2015)

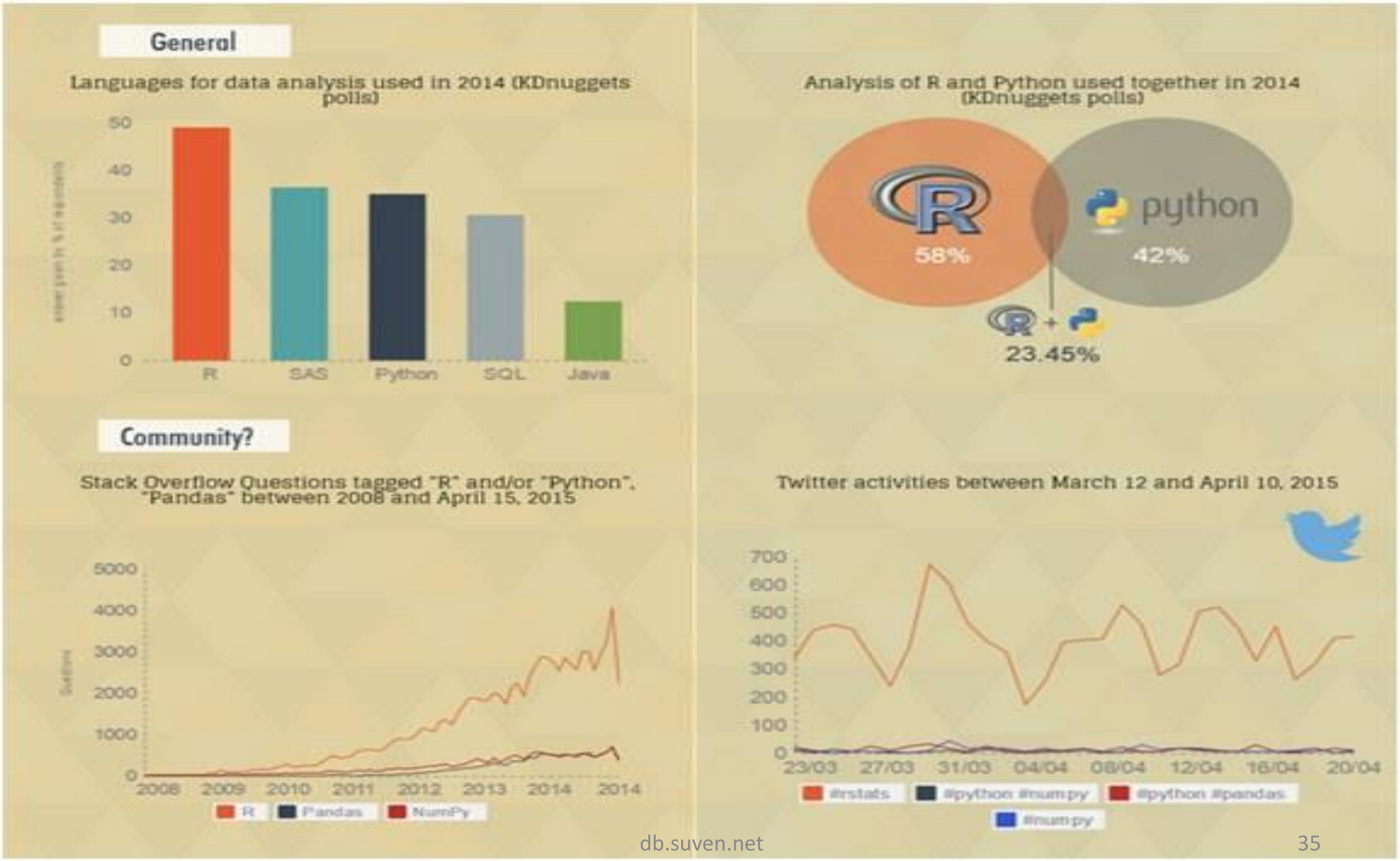
	2012	2013	2014	2015
Python	4	4	5	4
R	17	17	15	13

Jobs And Salary?

2014 Dice Tech Salary Survey:
Average Salary For High Paying Skills and Experience

R	$ 115,531
Python	$94,139

R And Python


General
Languages for data analysis used in 2014 (KDnuggets polls)
50
40
30
20
10
0
R
SAS
Python
SQL
Java
Analysis of R and Python used together in 2014 (KDnuggets polls)
python
58%
42%
23.45%
Community?
Stack Overflow Questions tagged "R" and/or "Python", "Pandas" between 2008 and April 15, 2015
5000
4000
3000
2000
1000
0
2008
2009
2010
2011
2012
2013
2014
2014
R
Pandas
NumPy
Twitter activities between March 12 and April 10, 2015
700
600
500
400
300
200
100
0
23/03
27/03
31/03
04/04
08/04
12/04
16/04
20/04
#rstats
#python #numpy
#python #pandas
#numpy

R Programming

By

Lokesh Singh

@SCTPL

Introduction

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.

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.

This programming language was named **R**, based on the first letter of first name of the two R authors (Robert Gentleman and Ross Ihaka), and partly a play on the name of the Bell Labs Language **S**.

```
# Print Hello World.
print("Hello World")

# Add two numbers.
 print(45+ 34.56)
```

OverView

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 "R Core Team") 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.

DataType

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

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.
color <- c(‘white',‘pink',“blue")
 print(color)
# Get the class of the vector.
 print(class(color))
```

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.

- Logical
- Numeric
- Integer
- Complex
- Character
- Raw

List

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(24,25,35),21.35,cos)
 # Print the list.
print(list1)
```

List

A matrix is a two-dimensional rectangular data set. It can be created using a vector input to the matrix function.

```
# Create a matrix.
matrix = matrix( c(‘b',‘c',‘d',‘e',‘f',‘g'), nrow = 2, ncol = 3, byrow = TRUE)
print(matrix)
```

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(‘red',‘pink'),dim = c(3,3,2))
print(a)
```

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.
colors <-
c('green','green','yellow','red','red','
red','green')
# Create a factor object.
factor_apple <-factor(colors)
# Print the factor.
print(factor_apple)
print(nlevels(factor_apple))
```

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(“Female", “Female",“Male"),
height = c(153, 172.5, 170),
weight = c(90,92, 70),
Age = c(24,83,62) )

 print(BMI)
```

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 Robjects. A valid variable name consists of letters, numbers and the dot or underline 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 , var.name	valid	Can start with a dot(.) but the dot(.)should not be followed by a number.
.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(2,4,6,8)
 # Assignment using leftward operator.
var.2 <- c(“study",“SCTPL")
# 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")
```

Note – The vector c(TRUE,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 SCTPL"
cat("The class of var_x is ",class(var_x),"\n")
var_x <- 34.56
cat(" Now the class of var_x is ",class(var_x),"\n")
 var_x <- 273L
cat(" Next the class of var_x becomes ",class(var_x),"\n")
```

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())
```

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"))
```

The variables starting with **dot(.)** are hidden, they can be listed using "all.names = TRUE" argument to ls() function.

```
print(ls(all.name = TRUE))
```

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())
```

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"))
```

The variables starting with **dot(.)** are hidden, they can be listed using "all.names = TRUE" argument to ls() function.

```
print(ls(all.name = TRUE))
```

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)
```

All the variables can be deleted by using the **rm()** and **ls()** function together.

```
rm(list = ls())
print(ls())
```

Types of 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.

We have the following types of operators in R programming −

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Assignment Operators
- Miscellaneous Operators

Arithmetic Operators

Operator	Description	Example

+	Adds two vectors	v <- c(2,5.5,6) t <- c(8, 3, 4)
		print(v+t)

−	Subtracts second vector from the first	v <- c(2,5.5,6) t <- c(8, 3, 4)
		print(v-t)

*	Multiplies both vectors	v <- c(2,5.5,6) t <- c(8, 3, 4)
		print(v*t)

/	Divide the first vector with the second	v <- c(2,5.5,6) t <- c(8, 3, 4)
		print(v/t)

%%	Give the remainder of the first vector with the second	v <- c(2,5.5,6) t <- c(8, 3, 4)
		print(v%%t)

%/%	The result of division of first vector with second (quotient)	v <- c(2,5.5,6) t <- c(8, 3, 4)
		print(v%/%t)

^	The first vector raised to the exponent of second vector	v <- c(2,5.5,6)
		t <- c(8, 3, 4)
		print(v^t)

Relational Operators

Operator	Description	Example
>	Checks if each element of the first vector is greater than the corresponding element of the second vector.	v <- c(2,5.5,6,9) t <- c(8,2.5,14,9) print(v>t)
<	Checks if each element of the first vector is less than the corresponding element of the second vector.	v <- c(2,5.5,6,9) t <- c(8,2.5,14,9) print(v < t)
==	Checks if each element of the first vector is equal to the corresponding element of the second vector.	v <- c(2,5.5,6,9) t <- c(8,2.5,14,9) print(v == t)
<=	Checks if each element of the first vector is less than or equal to the corresponding element of the second vector.	v <- c(2,5.5,6,9) t <- c(8,2.5,14,9) print(v<=t)
>=	Checks if each element of the first vector is greater than or equal to the corresponding element of the second vector.	v <- c(2,5.5,6,9) t <- c(8,2.5,14,9) print(v>=t)
!=	Checks if each element of the first vector is unequal to the corresponding element of the second vector.	v <- c(2,5.5,6,9) t <- c(8,2.5,14,9) print(v!=t)

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

&	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.	`v <- c(3,1,TRUE,2+3i)`
		`t<-c(4,1,FALSE,2+3i) print(v&t)`

|	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.	`v <- c(3,0,TRUE,2+2i)`
		`t <- c(4,0,FALSE,2+3i)`
		`print(v|t)`

!	It is called Logical NOT operator. Takes each element of the vector and gives the opposite logical value.	`v <- c(3,0,TRUE,2+2i) print(!v)`

Logical Operators Continue….

The logical operator && and || considers only the first element of the vectors and give a vector of single element as output.

Operator	Description	Example
&&	Called Logical AND operator. Takes first element of both the vectors and gives the TRUE only if both are TRUE.	v <- c(3,0,TRUE,2+2i) t <- c(1,3,TRUE,2+3i) print(v&&t)it produces the following result – [1] TRUE
||	Called Logical OR operator. Takes first element of both the vectors and gives the TRUE if one of them is TRUE.	v <- c(0,0,TRUE,2+2i) t <- c(0,3,TRUE,2+3i) print(v||t)it produces the following result – [1] FALSE

Assignment Operators

Operator	Description	Example

<−
or
=
or
<<−
	Called Left Assignment	`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)`

->
or
->>
	Called Right Assignment	`c(3,1,TRUE,2+3i)->v1`
		`c(3,1,TRUE,2+3i)->>v2`
		`print(v1)`
		`print(v2)`

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.	`v <- 2:8`
		`print(v)`

%in%	This operator is used to identify if an element belongs to a vector.	`v1 <- 8`
		`v2 <- 12`
		`t <- 1:10 print(v1 %in% t)`
		`print(v2 %in% t)`

%*%	This operator is used to multiply a matrix with its transpose.	`M = matrix( c(2,6,5,1,10,4), nrow = 2,ncol = 3,byrow = TRUE)`
		`t = M %*% t(M)`
		`print(t)`

Decision Making Syntax

If Statement

```
if(boolean_expression)
{
// statement(s) will execute if the boolean
expression is true.
 }
```

If Statement Example

```
x <- 30L
if(is.integer(x)) {
   print("X is an Integer")
}
```

If...Else Statement

```
if(boolean_expression) {
   // statement(s) will execute if the
boolean expression is true.
} else {
   // statement(s) will execute if the
boolean expression is false.
}
```

If...Else Statement Example

```
x <- c("what","is","truth")

if("Truth" %in% x) {
   print("Truth is found")
} else {
   print("Truth is not found")
}
```

Decision Making Syntax Continue..

if...else if...else Statement

```
if(boolean_expression 1) {
  // Executes when the boolean expression 1 is true.
} else if( boolean_expression 2) {
  // Executes when the boolean expression 2 is true.
} else if( boolean_expression 3) {
  // Executes when the boolean expression 3 is true.
} else {
  // executes when none of the above condition is true.
}
```

if...else if...else Statement

```
x <- c("what","is","truth")

if("Truth" %in% x) {
   print("Truth is found the first time")
} else if ("truth" %in% x) {
   print("truth is found the second time")
} else {
   print("No truth found")
}
```

Switch Statement

```
switch(expression, case1, case2,
case3....)
```

Switch Statement Example

```
x <- switch(
   3,
   "first",
   "second",
   "third",
   "fourth"
)
print(x)
```

Loops

Repeat Loop

```
repeat {
  commands
  if(condition) {
    break
  }
}
```

Repeat Loop Example

```
v <- c("Hello","loop")
cnt <- 2

repeat {
  print(v)
  cnt <- cnt+1

  if(cnt > 5) {
    break
  }
}
```

While Loop

```
while (test_expression) {
  statement
}
```

While Loop Example

```
v <- c("Hello","while loop")
cnt <- 2

while (cnt < 7) {
  print(v)
  cnt = cnt + 1
}
```

Loops Continue…

For Loop

```
for (value in vector) {
   statements
}
```

For Loop Example

```
v <- LETTERS[1:4]
for ( i in v) {
   print(i)
}
```

Loop Control Statements

break statement

```
break
```

Break Example

```
v <- c("Hello","loop")
cnt <- 2

repeat {
   print(v)
   cnt <- cnt + 1

   if(cnt > 5) {
      break
   }
}
```

Next statement

```
next
```

Next Example

```
v <- LETTERS[1:6]
for ( i in v) {

   if (i == "D") {
      next
   }
   print(i)
}
```

Function

Function Defination

```
function_name <- function(arg_1, arg_2, ...) {
   Function body
}
```

Build in Function

```
function_name <- function(arg_1, arg_2, ...) {
   Function body
} # 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))
```

NOTE:-For more built in function check short-refcard.pdf given on SCTPL site .

User-defined 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)
  }
}
```

User-defined 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)
```

Function Continue…

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()
```

Calling a Function with Argument Values (by position and by name)

```
# 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)
```

Calling a Function with Default Argument

```
# 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)
```

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 can not 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 can not be inserted into a string starting and ending with double quotes.
•Single quote can not be inserted into a string starting and ending with single quote.

Examples of Valid Strings

```
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)
```

String Continue…

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.

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)
```

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.

```
# 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)
```

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'))
```

Vectors Continue…

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)
```

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.

Using sequence (Seq.) operator

```
# Create vector with elements from 5 to
9 incrementing by 0.4.
print(seq(5, 9, by = 0.4))
```

Using the c() function

```
# The logical and numeric values are
converted to characters.
s <- c('apple','red',5,TRUE)
print(s)
```

```
# Accessing vector elements using position.
t <-
c("Sun","Mon","Tue","Wed","Thurs","Fri","Sa
t")
u <- t[c(2,3,6)]
print(u)
# Accessing vector elements using logical
indexing.
v <-
t[c(TRUE,FALSE,FALSE,FALSE,FALSE,TRUE,FALS
E)]
print(v)
# Accessing vector elements using 0/1
indexing.
y <- t[c(0,0,0,0,0,0,1)]
print(y)
```

List

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.
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)
```

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)
```

List Continue….

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.

```
# 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 thrid 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)
```

List Continue….

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])
```

List Continue….

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)
```

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)
```

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.

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

```
# 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)
```

Matrices Continue…

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])
```

Array

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) 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.

```
# 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)
```

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)
```

Array Continue…

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])
```

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)
```

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.

```
# 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))
```

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","femal
e","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)
```

Factors Continue…

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)
```

Factors Continue…

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: 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.

```
v <- gl(3, 4, labels = c("Tampa", "Seattle","Boston"))
print(v)
```

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 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)
```

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)
```

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))
```

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$emp_name,emp.data$sala
ry)
print(result)
```

```
# 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", "20
23", "2014-11-15", "2014-05-11",
      "2015-03-27")), stringsAsFactors = FALS
)
# Extract first two rows.
result <- emp.data[1:2,]
print(result)
# Extract 3rd and 5th row with 2nd and 4th
column.
result <- emp.data[c(3,5),c(2,4)]
print(result)
```

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" coulmn.
emp.data$dept <- c("IT","Operations","IT","HR","Finance")
v <- emp.data
print(v)
```

Data Frames Continue..

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","Fianance"),
  stringsAsFactors = FALSE
)
# Bind the two data frames.
emp.finaldata <- rbind(emp.data,emp.newdata)
```

Packages

R packages are a collection of R functions, complied 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
https://cran.r-project.org/web/packages/available_packages_by_name.html

```
install.packages("Package Name")

# Install the package named "XML".
 install.packages("XML")
```

Check Available R Packages

Get library locations containing R packages

```
.libPaths()
```

Get the list of all the packages installed

```
library()
```

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.

Data Reshaping Continue…

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 form 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)
```

R Data Interfaces

CSV

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.

The file should be present in current working directory so that R can read it.

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())
```

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
 ,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)
```

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))
```

CSV Continue…

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)
```

Get the details of the person with max salary

```
# Create a data frame.
data <- read.csv("input.csv")

# Get the max salary from data frame.
sal <- max(data$salary)
print(sal)
```

Get all the people working in IT department

```
# Create a data frame.
data <- read.csv("input.csv")

retval <- subset( data, dept == "IT")
print(retval)
```

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)
```

Excel Sheet

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")
# Load the library into R workspace.
library("xlsx")
```

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)
```

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 he file.

You can read a xml file in R using the "XML" package. This package can be installed using following command.

```
install.packages("XML")
```

XML Files Continue…

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(*.*)**.

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)
```

```
<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>
</RECORDS>
```

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)
```

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" ],

  "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"]
}
```

JSON Files Continue…

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)
```

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)
```

The data is 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)
```

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)
```