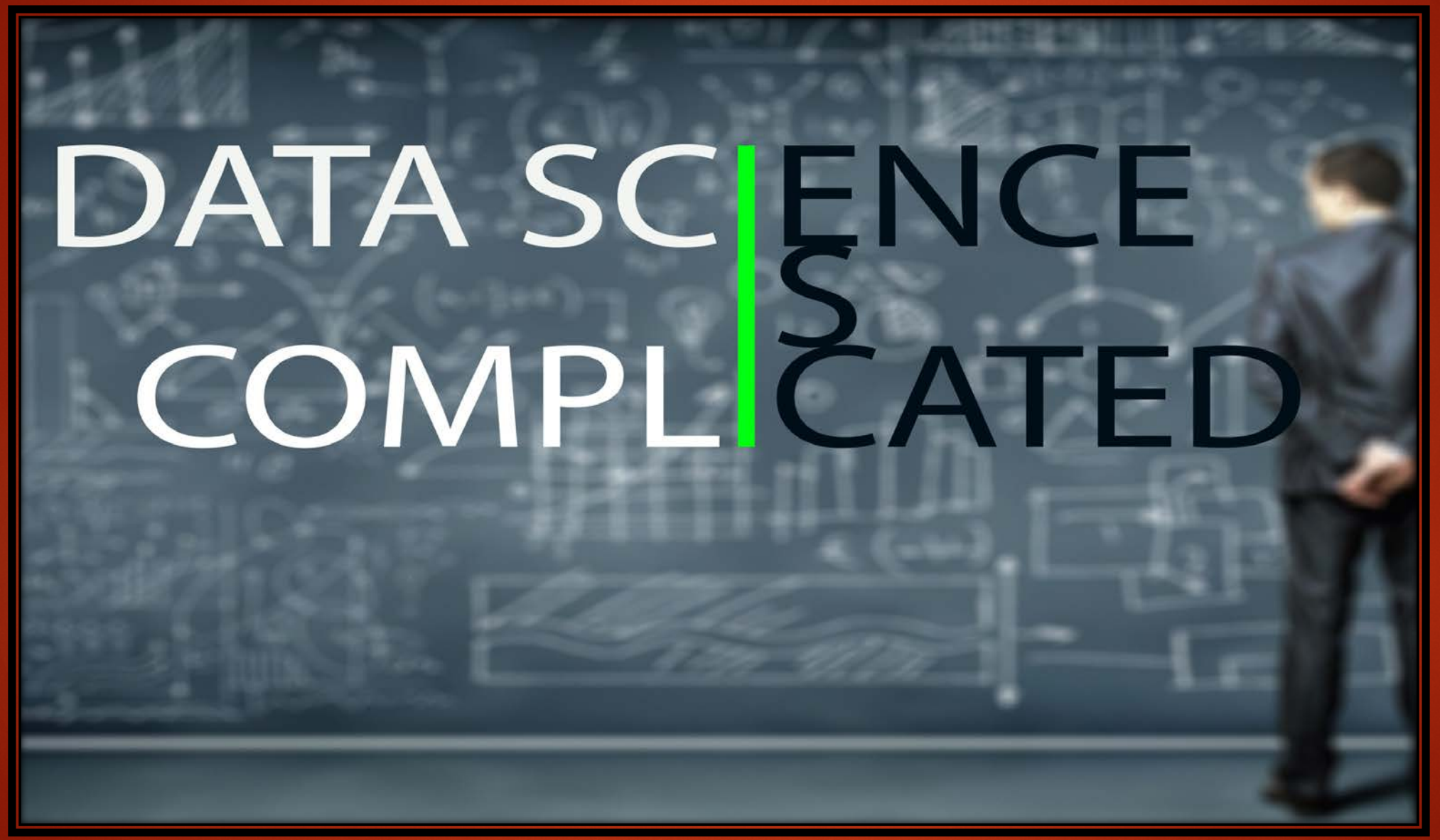
Data Analytics using R

SUVEN CONSULTANTS | NIRAJ SHARMA

Topics Covered

- Understanding Data Science & Data Analysis.
- Understanding Data & sources of data.
- Big Data.
- Difference between Data Scientist & Data Analyst.
- Challenges & Technology we have for Big Data.
- Installing & Understanding R-Progarmming.
- Learning R-Objects.
- Operators, Decision Making, Loops.
- Functions, Packages.
- Data Importing, Pre-Processing, Wrangling, Cleaning.
- Case Study.

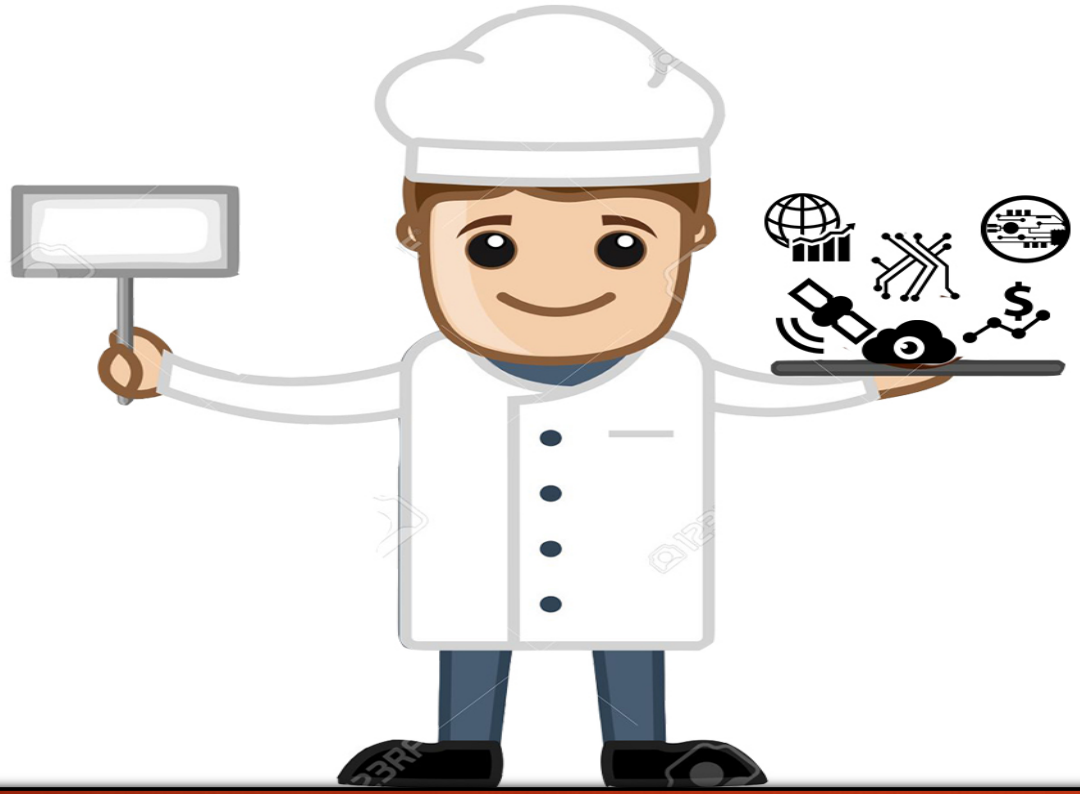
DIFFERENT INDUSTRIES USE DATA SCIENTISTS FOR DIFFERENT PURPOSES


DATA SCIENCE
IS
COMPLICATED

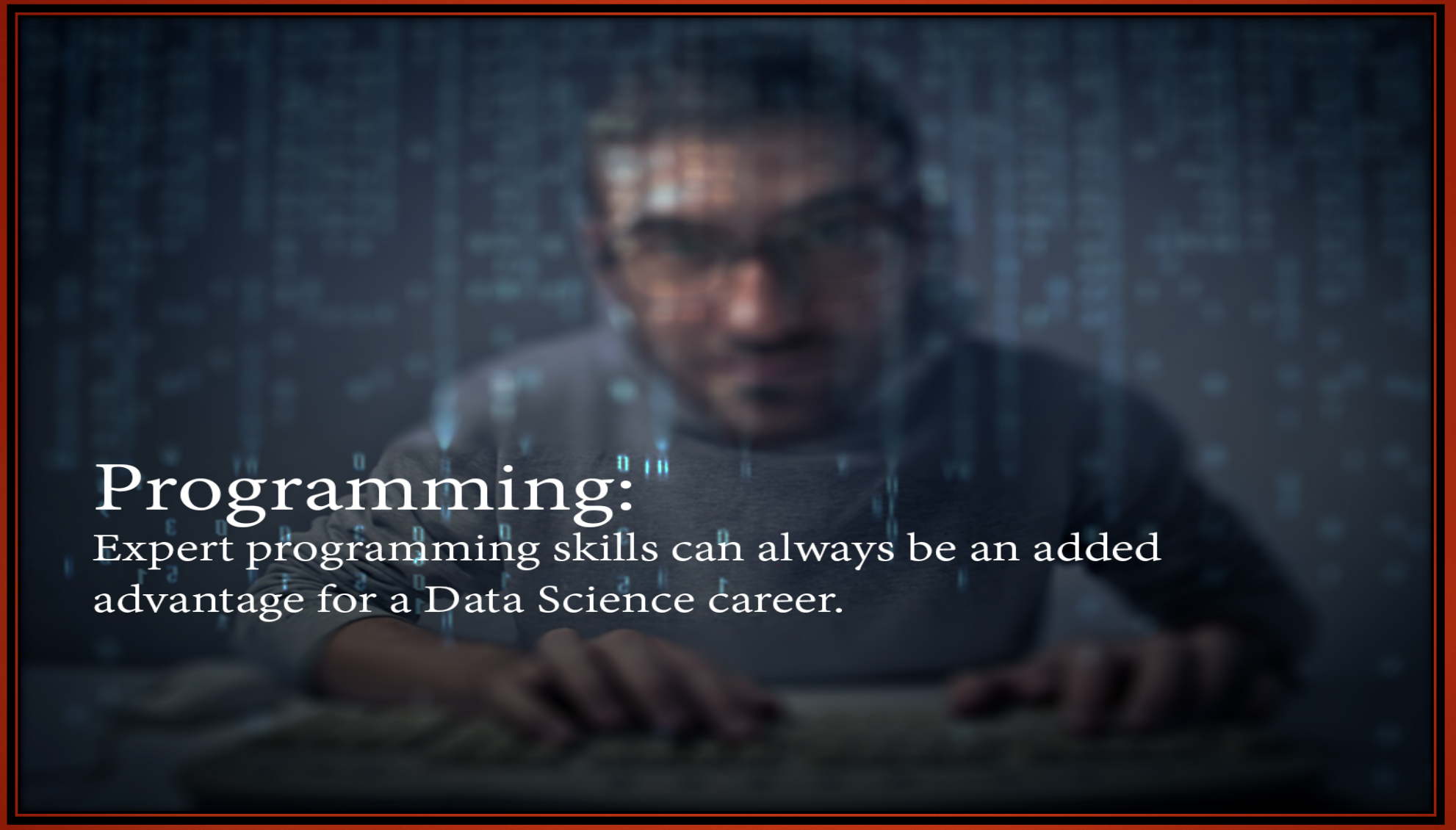
MAIN AIM OF A DATA SCIENTIST IS TO FIND MEANING IN THE CHAOS OF BIG DATA

- What is a data scientist?
- What does a data scientist do?
- What data scientist skills does it take for a big data professional to begin a career in data science?
- Where and how can the journey to pursue career as a data scientist begin?

What do you need to know to become a Data Scientist?




Statistics:
Solid foundation in statistics is
a MUST !!


Programming:
Expert programming skills can always be an added advantage for a Data Science career.

Data Visualization

Presenting data in a format that is visually appealing and understandable to end users

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, stockprice), governments (e.g., crime rates, literacy rates)

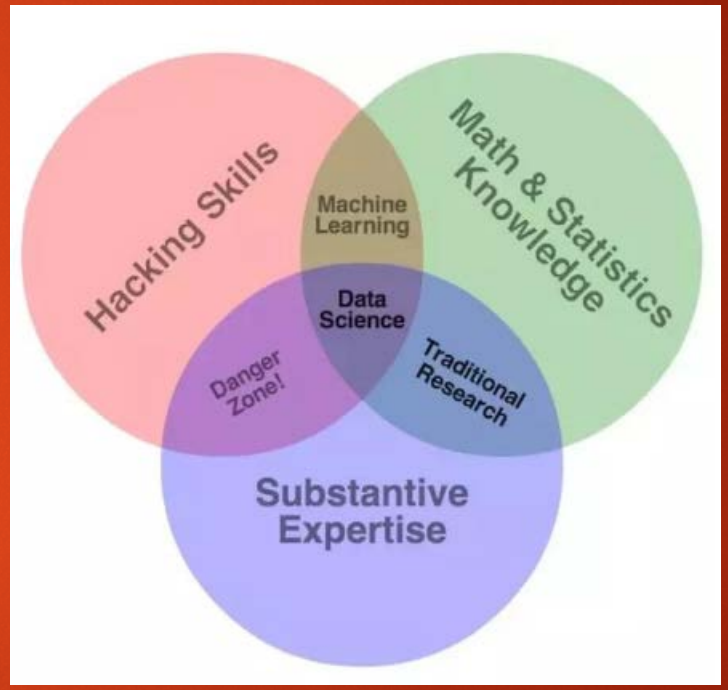
What is Data analysis ?

- Data analysis is a primary component of data mining and Business Intelligence (BI) and is key to gaining the insight that drives business decisions.
- It 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.

- 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.



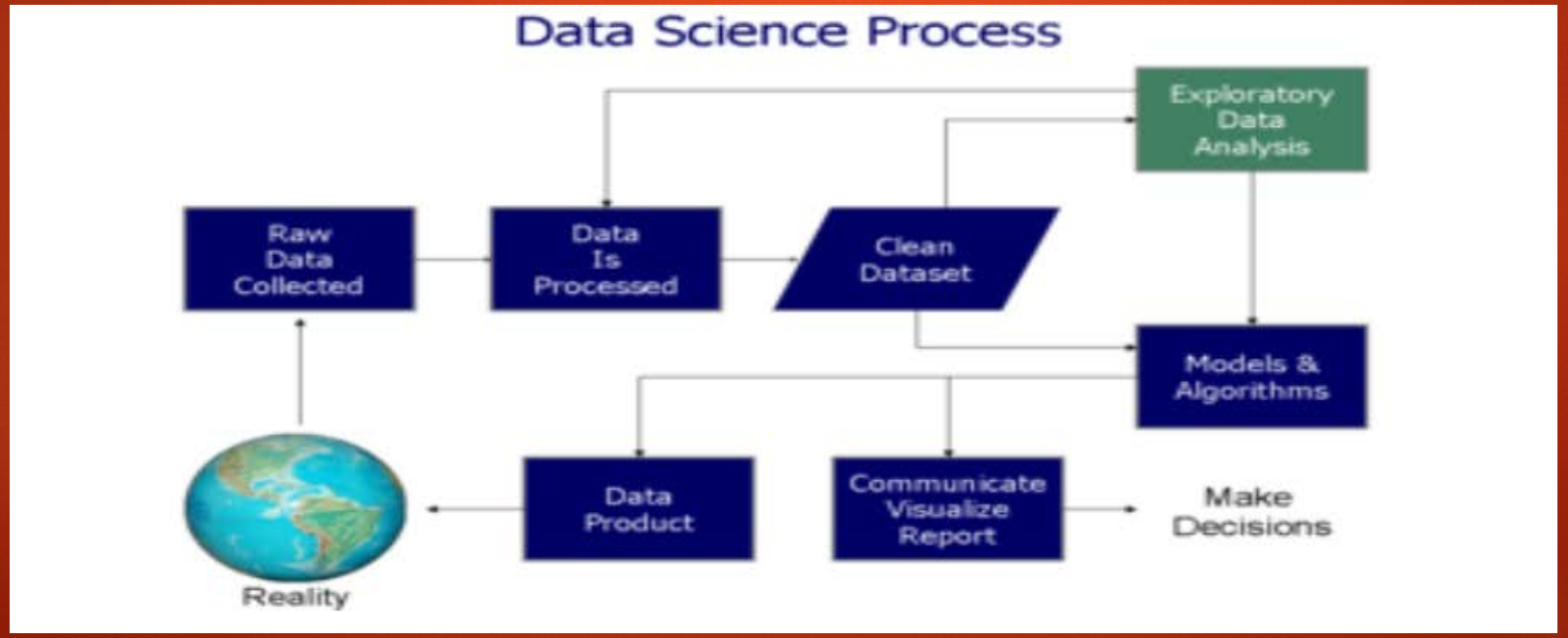
What is difference between data analysis and data science ?

- 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. They are efficient in picking the right problems, which will add value to the organization after resolving it.

- 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.

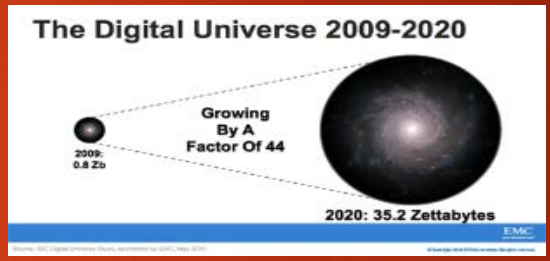
Big Data Definition

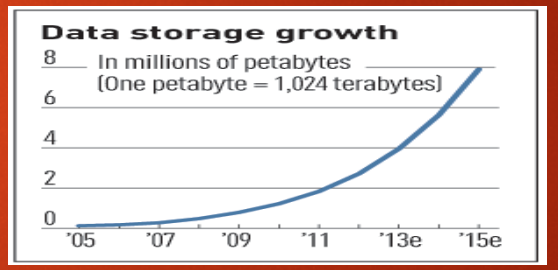
- No single standard definition…
- "***Big Data***" is data whose scale, diversity, and complexity require new architecture, techniques, algorithms, and analytics to manage it and extract value and hidden knowledge from it…

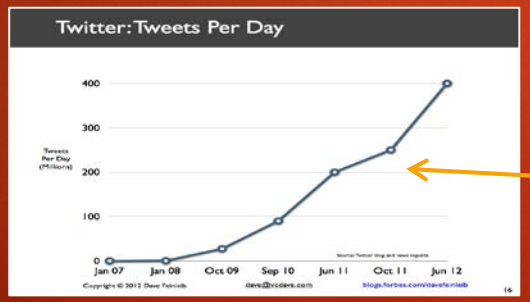


Characteristics of Big Data:
1-Scale (Volume)

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



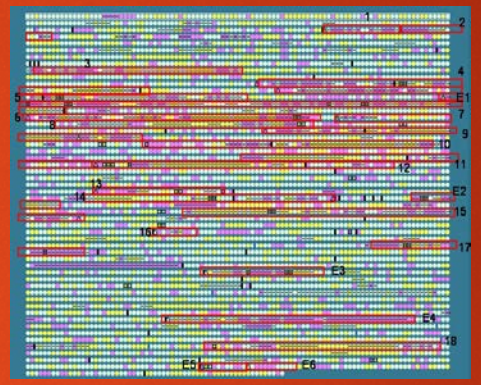


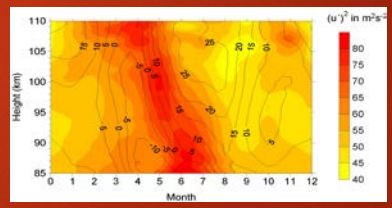


Exponential increase in collected/generated data

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





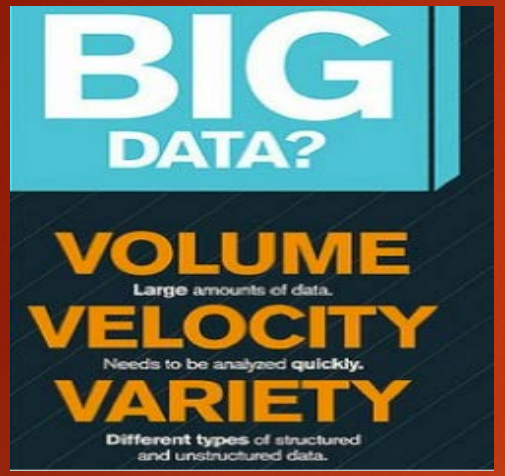
To extract knowledge➔ all these types of data need to linked together

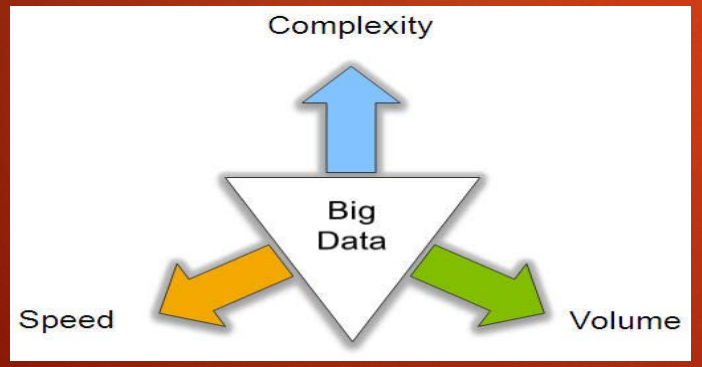
Characteristics of Big Data:
3-Speed (Velocity)

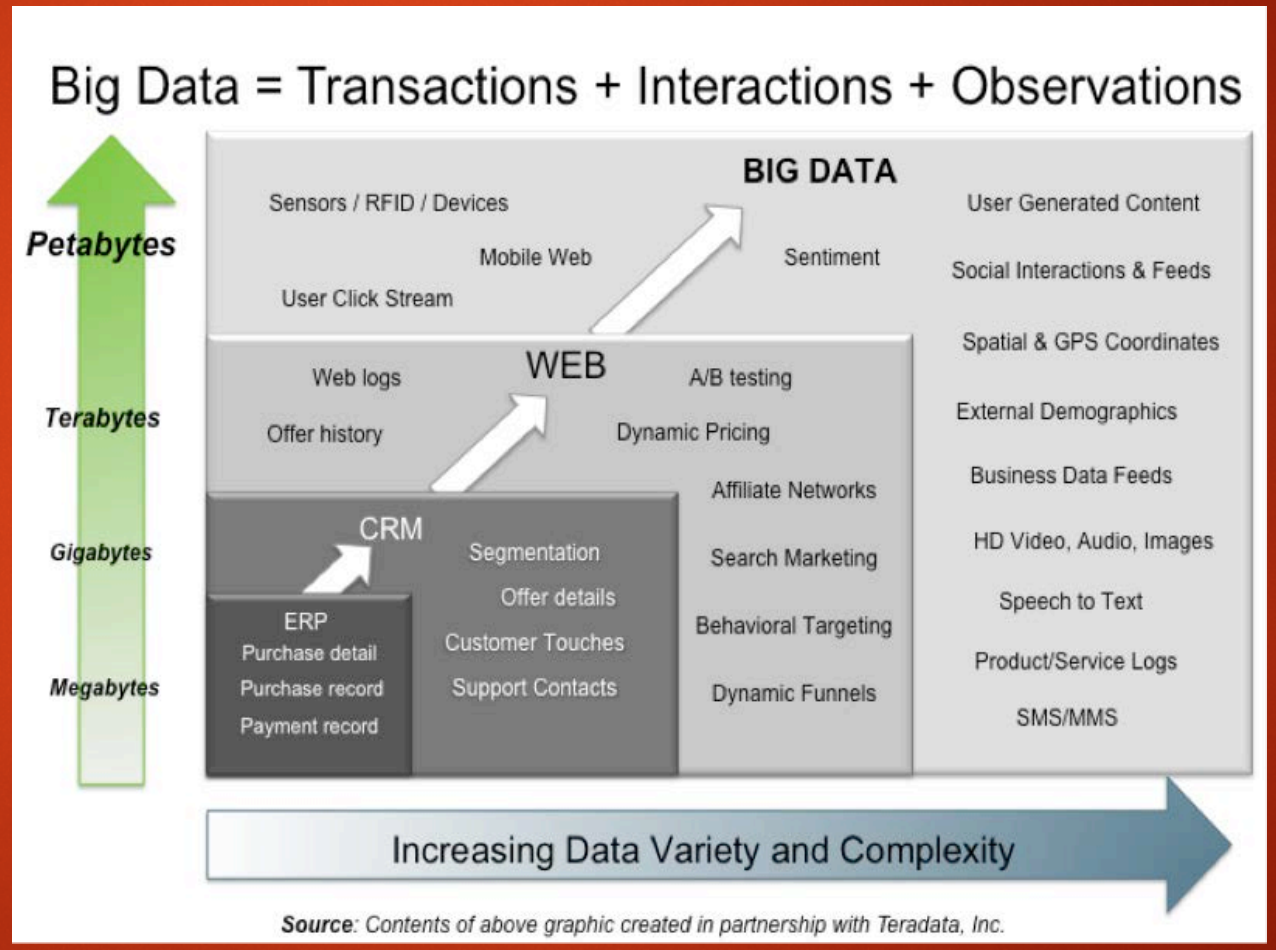
- Data is begin generated fast and need to be processed fast
- Online Data Analytics
- Late decisions ➔ missing opportunities

- **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.

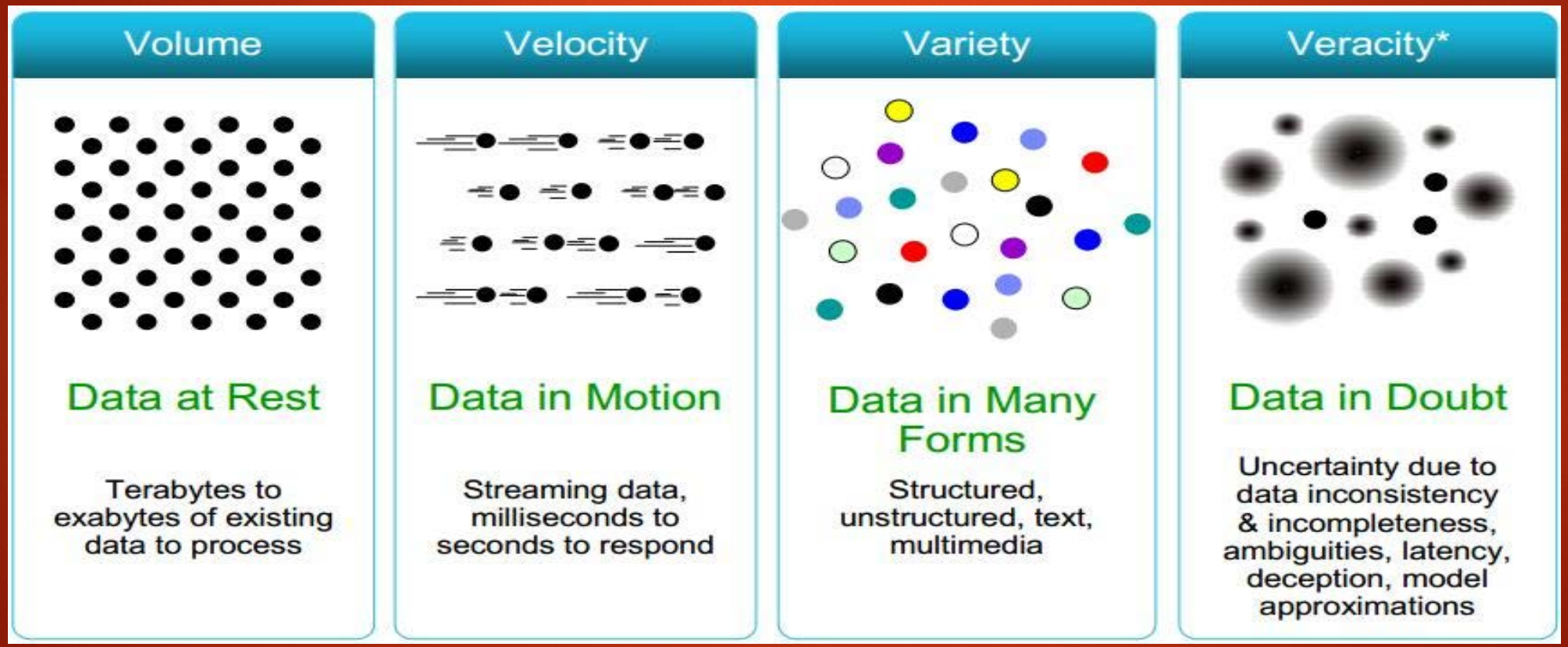
Big Data: 3V's


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
WEB
Web logs
Offer history
A/B testing
Dynamic Pricing
External Demographics
Business Data Feeds
Affiliate Networks
CRM
Segmentation
Offer details
Customer Touches
Support Contacts
Search Marketing
Behavioral Targeting
Dynamic Funnels
HD Video, Audio, Images
Speech to Text
Product/Service Logs
SMS/MMS
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



Harnessing Big Data

- **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

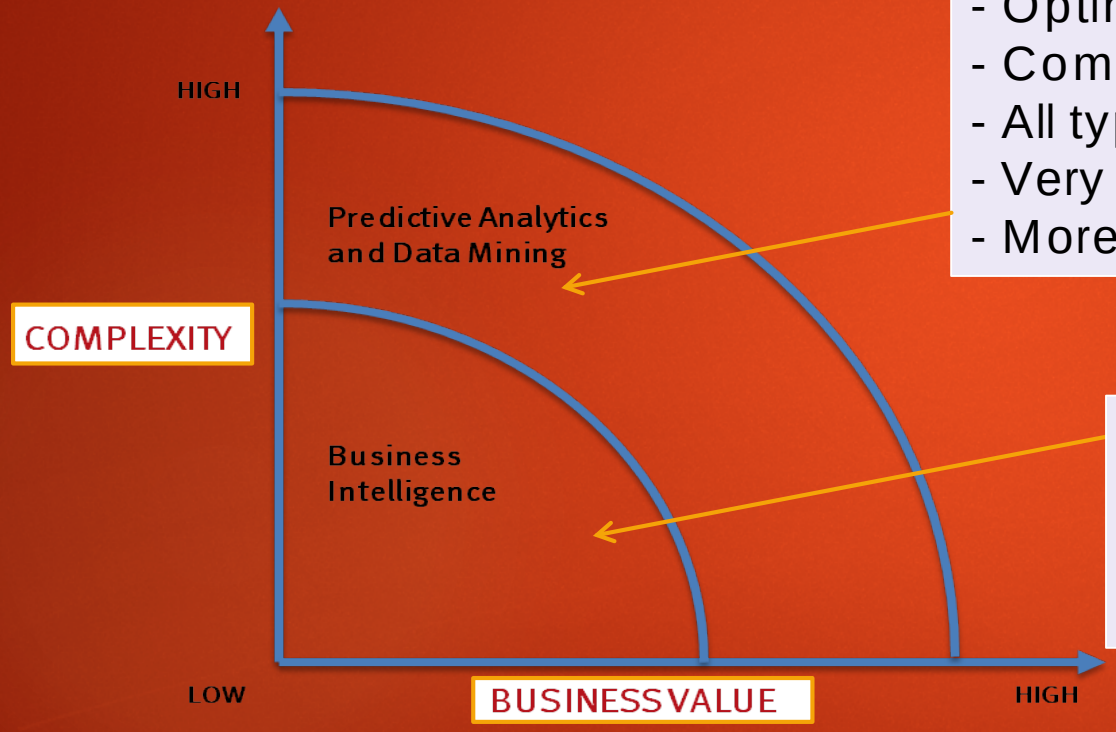
- **The Model of Generating/Consuming Data has Changed**
- **Old Model:** Few companies are generating data, all others are consuming data.




- **New Model:** All of us are generating data, and all of us are consuming data.



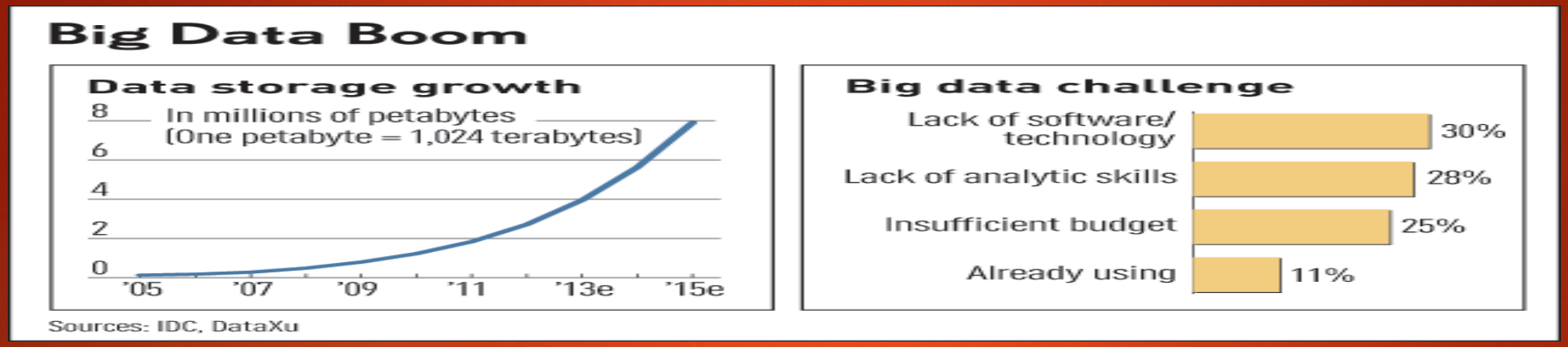
What's driving Big Data



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

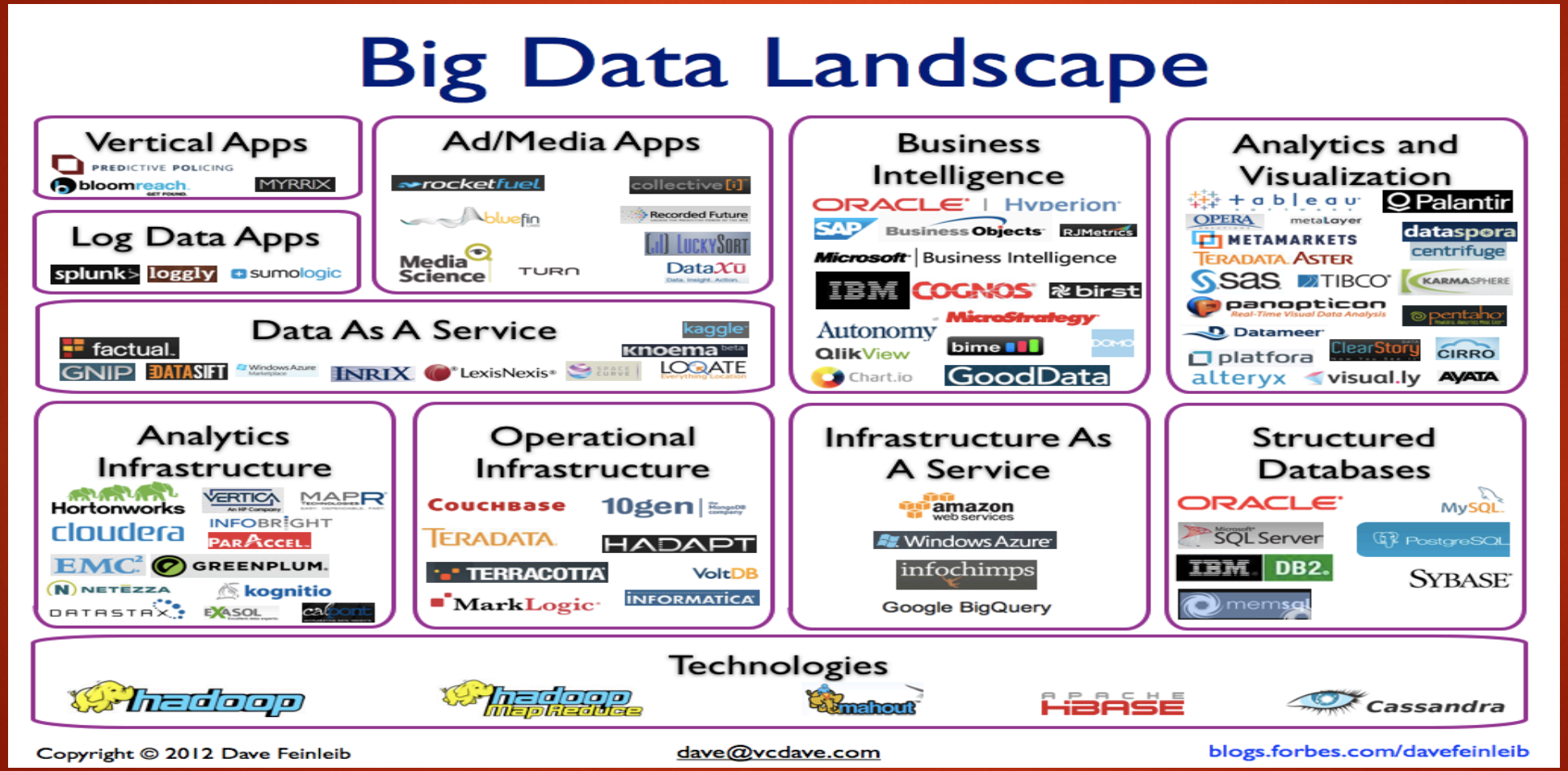
- Ad-hoc querying and reporting
- Data mining techniques
- Structured data, typical sources
- Small to mid-size datasets

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 ??



Introduction & Basics

- **What is R?**
- R is a programming language developed by Ross Ihaka and Robert Gentleman in 1993.
- R possesses an extensive catalog of statistical and graphical methods. It includes machine learning algorithm, linear regression, time series, statistical inference to name a few.
- R is entrusted many large companies also use R programming language, including Uber, Google, Airbnb, Facebook and so on.

- Data analysis with R is done in a series of steps; ***Programming***, ***Transforming***, ***Discovering***, ***Modeling*** and ***Communicate*** the results
- **Program**: R is a clear and accessible programming tool
- **Transform**: R is made up of a collection of libraries designed specifically for data science
- **Discover**: Investigate the data, refine your hypothesis and analyze them
- **Model**: R provides a wide array of tools to capture the right model for your data
- **Communicate**: Integrate codes, graphs, and outputs to a report with R Markdown or build Shiny apps to share with the world

Introduction & Basics

- Derived from Bell Labs Language **S.**
- Developed 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 precompiled binary versions are provided for various operating systems like Linux, Windows and Mac.

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 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.
- R programming features include database input, exporting data, viewing data, variable labels, missing data, etc.
- R is an open source and we can use it anywhere.
- In R, anyone is welcome to provide bug fixes, code enhancements, and new packages.

How companies are using R

- **Ford** uses R to improve the design of its vehicles.
- **Twitter** uses R to monitor user experience.
- The **US National Weather Service** uses R to predict severe flooding.
- R is being used by **The New York Times** to create infographics.
- **Google** uses R to calculate the ROI of advertising campaigns.
- **Facebook** uses R to update status and its social network graph. Also, for predicting colleague interactions with R, Facebook uses it.
- **Microsoft** uses R for the Xbox matchmaking service. Also, as a statistical engine within the Azure ML framework.
- **Mozilla** it is the foundation behind the Firefox web browser, uses R to visualize Web activity.

Local Environment Setup

- **Windows Installation**
- You can download the Windows installer version of R from https://cran.r-project.org/bin/windows/base/ and save it in a local directory.

- **Linux Installation**
- R is available as a binary for many versions of Linux at the location R. https://cran.r-project.org/bin/linux/

- **Mac Installation**
- You can download the Mac installer version of R from http://www.r-project.org/ & click on Download R for (Mac) OS X.

- **R-Studio (IDE for Development)**
- You can download installer of R-Studio from https://www.rstudio.com/products/rstudio/download/ respective for your OS.

R Data Types

- What is R Data Types?
- It can handle complex statistical operations in an easy and optimized way.
- **Vector** – A basic data structure of R containing the same type of data
- **Lists –** Lists store collections of objects when vectors are of same type and length in a matrix.
- **Matrices –** A matrix is a rectangular array of numbers or other mathematical objects. We can do operations such as addition and multiplication on Matrix in R.
- **Data Frames –** Generated by combining together multiple vectors such that each vector becomes a separate column.

Vectors in R

- In R, **Vector** is a basic data structure in R that contains element of similar type. These data types in R can be logical, integer, double, character, complex or raw.
- In R using the function *typeof()* / *class()* one can check the data type of vector.
- The function *length()* determines the number of elements in a vector.

- Create a Vector :

```
N1 <- c('white', 'pink', 'blue')
N2 <- c(15L,20L,25L)
N3 <- c(15,20,25)
N4 <- c(TRUE,FALSE,FALSE,TRUE)
N5 <- c(2+3i, 4+6i, 8+9i)
N6 <- charToRaw("Hello")
print(N1)
length(N1)
typeof(N1)
class(N1)
```

Vectors in R

- Creating a sequence

```
v <- 5:13
print(v)
```

- If the final element specified does not belong to the sequence then it is discarded.

```
v <- 3.8:11.4
print(v)
```

- Create vector with elements from 5 to 9 incrementing by 0.4.

```
print(seq(5, 9, by = 0.4))
```

- Typecasting

  If atomic variables aren't similar datatype they are typecasted to highest precedence of collective elements.

```
s <-c('apple','red',5,TRUE, 2+3i)

print(s)

class(s)
```

> character

Vectors in R

- Accessing vector Elements
- 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,
FALSE,FALSE,TRUE,FALSE)]

print(v)
```

- Accessing vector elements using 0/1 indexing.

```
y <- t[c(0,0,0,0,0,0,1)]

print(y)
```

- Accessing vector elements using range indexing.

```
u <- t[2:6]
```

Vectors in R

- Updating Vector
- Single Element Update

```
v1 <- c(23,45,67,89,76)
v1[3] <- 101 # Update data at
             index 3
```

- Multi Element Update

```
v1[2:4] <- 100
v1[-3] <- 101  # Except index 3
               update all
```

- Deleting a Element in vector

```
v1[3] <- v1[-3] # Except index 3
V1[3] <- NA # Update with NA
```

List in R

- 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.
- Type of List is List itself.
- Creating a List

```
list_data <- list("Red", "Green", c(21,32,11), TRUE, 51.23, 119.1)

print(list_data)

class(list_data)
```

List in R

- **Naming 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")
print(list_data)
```

- **Accessing List Elements**

Access the first element of the list.

```
print(list_data[1])
```

Access the thrid element.

```
print(list_data[3])
```

Access the list element using the name of the element.

```
print(list_data$A_Matrix)
```

List in R

- **Updating a List**

```
list_data[3] <- "updated
element"

print(list_data[3])
```

- **Deleting a List Element**

```
list_data[4] <- list_data[-4]

list_data[4] <- NULL
```

- **Merging a List**

```
# 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)
```

List in R

- **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 (e.g Arithmetic).

```
# Create lists.

list1 <- list(1:5)
list2 <-list(10:14)

# Convert the lists to vectors.

v1 <- unlist(list1)
v2 <- unlist(list2)

# Now add the vectors

result <- v1+v2
print(result)
```

Matrices in R

- 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.

- The basic syntax for creating a matrix in R is –
```
matrix(data, nrow, ncol, byrow, dimnames)
```

- Creating a Matrix

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)
```

Matrices in R

- **Defining Row & Column 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)
```

- Accessing Elements of Matrix

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 in R

- 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.
- 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.

- Creating an array

Create two vectors of different length

```
vector1 <- c(5,9,3)
vector2 <-
c(10,11,12,13,14,15)
```

Use vectors as input to the array.

```
result <- array(c(vector1,
vector2),dim = c(3,3,2))

print(result)
```

Array in R

- **Naming** Rows, Columns & Matrix of array.

Create two vectors of different lengths.

```
vector1 <- c(5,9,3)
vector2 <- c(10,11,12,13,14,15)


col <- c("COL1","COL2","COL3")
row <- c("ROW1","ROW2","ROW3")
mat <- c("Matrix1","Matrix2")
```

Use vectors as input to the array.

```
result <- array(c(vector1,vector2),
                dim = c(3,3,2),
          dimnames = list(row, col,mat))


print(result)
```

- **Accessing Array Elements**

Element at third row of the second matrix of the array.

```
print(result[3,,2])
```

Element in the 1st row and 3rd column of the 1st matrix.

```
print(result[1,3,1])
```

Print the 2nd Matrix.

```
print(result[,,2])
```

Array in R

- Manipulating Array Elements

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(vector3,vector4),
                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 in R

- 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
- E.g "Male, "Female" and True, False etc. They are useful in data analysis for statistical modeling

- Creating a Factor

  Create a Vector

```
data <- c("East","West",
"East","North", "North","East",
"West","West", "West","East",
"North")
```

  Use factor() to create factor

```
factor_data <- factor(data)
print(factor_data)
```

Factors in R

- Reorder the levels

```
new_order_data <- factor(factor_data,levels = c("East","West","North"))
print(new_order_data)
```

- Generating factor levels

```
Syntax: gl(n, k, labels)

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

Dataframes in R

- A data frame is a table or a two-dimensional tabular structure in which each column contains values of one variable and each row contains one set of values from each column.
- Characteristics of a data frame.
  - The column names should be non-empty.
  - The data stored in a data frame can be of numeric, factor or character type.
  - Each column should contain same number of data items.

- Creating a Dataframe

```
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)
```

Dataframe in R

- **Structure of the Data Frame**

```
str(emp.data)
```

- **Summary of Data in Data Frame**

```
summary(emp.data)
```

- **Number of Colums & Rows**

```
ncol(emp.data)
nrow(emp.data)
dim(emp.data)
attributes(emp.data)
```

- Extract Data from Dataframe

Extract Specific columns.

```
print(emp.data$emp_name)
```

Extract first two rows.

```
print(emp.data[1:2,])
```

Extract 3rd and 5th row with 2nd & 4th column.

```
result <- emp.data[c(3,5),c(2,4)]
print(result)
```

Dataframe in R

- **Expand Data Frame**
- **Add Column**
  Just add the column vector using a new column name

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

- **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.

```
# 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)
```

Data Reshaping

- Creating Dataframe from other R-Objects :
- Steps :
  - Create a Vector
  - Bind the Vector to create a Matrix
  - Create a new dataframe & bind it to previously created Matrix.

```
city <-c("Tampa", "Seattle",
             "Hartford", "Denver")
state <- c("FL","WA","CT","CO")
zipcode <- c(33602,98104,06161,80294)

addresses <- cbind(city,state,zipcode)

new.address <- data.frame(
city = c("Lowry","Charlotte"),
state = c("CO","FL"),
zipcode = c("80230","33949"),
stringsAsFactors = FALSE
)

Final <- rbind(addresses,new.address)

print(Final)
```

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 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.

Variables

- Variable Assignment

  Using equalto operator.
  ```
  var.1 = c(2,4,6,8)
  ```

  Using leftward operator.
  ```
  var.2 <- c("study","SCTPL")
  ```

  Using rightward operator.
  ```
  c(TRUE,1) -> var.3
  ```

- Find Variables

  ```
  print(ls())
  ```

  # List the variables with pattern "var".
  ```
  print(ls(pattern = "var"))
  ```

  The variables starting with **dot(.)** are hidden, to find all variables

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

- Deleting Variables

  ```
  rm(var.3)
  rm(list = ls())  #Deletes all variables
  ```

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
  - 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	print(v-t)
*	Multiplies both vectors	print(v*t)
/	Divide the first vector with the second	print(v/t)
%%	Give the remainder of the first vector with the second	print(v%%t)
%/%	The result of division of first vector with second (quotient)	print(v%/%t)
^	The first vector raised to the exponent of second vector	print(v^t)

Relational Operator

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.	print(v < t)
==	Checks if each element of the first vector is equal to the corresponding element of the second vector.	print(v == t)
<=	Checks if each element of the first vector is less than or equal to the corresponding element of the second vector.	print(v<=t)
>=	Checks if each element of the first vector is greater than or equal to the corresponding element of the second vector.	print(v>=t)
!=	Checks if each element of the first vector is unequal to the corresponding element of the second vector.	print(v!=t)

Logical Operators

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)

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)
||	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)

Miscellaneous Operators

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

- **If Statement**
- Syntax :

```
if(condition)
{
        // if condition is true
}
```

- Example

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

- **If...Else Statement**

```
if(condition) {
        // if condition is true
} else {
        // if condition is false.
}
```

- Example

```
x <- c("what","is","truth")
if("Truth" %in% x) {
     print("Truth is found")
} else {
     print("Truth is not found")
}
```

Decision Making

- **if...else if...else Statement**
- Syntax :

```
if(boolean_expression 1) {
      // boolean expression 1 is true.

} else if( boolean_expression 2) {
      // boolean expression 2 is true.

} else if( boolean_expression 3) {
      // boolean expression 3 is true.

} else {
// executes when none of the above
condition is true.
}
```

- Example :

```
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")
}
```

Decision Making

- **Switch Statement**
- Syntax :

  switch(expression, case1, case2, case3)
- Example :

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

Loops

- **Repeat Loop**
- Syntax :

```
repeat {
    // commands to execute
    if(condition) {
     // break condition
    }
}
```

- Example :

```
v <- c("Hello","loop")
cnt <- 2
repeat {
    print(v)
    cnt <- cnt+1
    if(cnt > 5) {
        // break condition
    }
}
```

Loops

- **While Loop**
- Syntax :

```
while (test_expression) {
      // statement to execute
}
```

- Example :

```
v <- c("Hello","while loop")
cnt <- 2
while (cnt < 7) {
     print(v)
     cnt = cnt + 1
}
```

- **For Loop**
- Syntax :

```
for (value in vector) {
      // statements to execute
}
```

- Example :

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

Loops

- **Break Control statement**
- Break :
  - Break out of Loop.
- Example :

```
v <- c("Hello","loop")
cnt <- 2
repeat {
    print(v)
    cnt <- cnt + 1
    if(cnt > 5) {
        break
    }
}
```

- Next
  - Skip the current iteration.
- Example :

```
v <- LETTERS[1:6]
for ( i in v) {
    if (i == "D") {
        next
    }
    print(i)
}
```

Functions

- Predefined functions :
- Functions such as mean, combine, array etc are predefined function.
  - `mean(25:82)`
  - `sum(41:50)`
- User defined functions :
- Similarly user can define their own functions, which increases code efficiency & reusability.

- Create a function to print squares of numbers in sequence.

```
new.function <- function(a) {
    for(i in 1:a) {
        b <- i^2
        print(b)
    }
}
new.function(4)
```

Functions

- Function call :

```
new.function <- function(a,b,c)
{
    result <- a * b + c
    print(result)
}
# call by position.
new.function(5,3,11)
# call by name.
new.function(a = 11, b = 5, c =
3)
```

- Function with default values:

```
new.function <- function(a = 3, b = 6)
{
    result <- a * b
    print(result)
}
```

Call without giving any argument.

```
new.function()
```

Call with new parameters.

```
new.function(9,5)
```

Functions

- Math :
  - abs(x) – Absolute value of x
  - sqrt(x) – Square root of x
  - sum(x) – Sum of x
  - cos(x) – Cosine value of x
  - sin(x) – Sine value of x
  - tan(x) – Tan value of x
  - exp(x) – Exponential value of x
  - log(x) – Log value of x
- Statistical functions :
  - mean(x) – Average of x
  - median(x) – Median of x
  - min(x) – Minimum value in x
  - max(x) – Maximum value in x
  - quantile(x) – Quantile distribution of x
  - sd(x) - Calculate standard deviation
  - var(x) - Calculate variance
  - sample() - Random samples
  - summary(x) – Min,Max, Median etc of x

Functions

- length(x) – Return no. of elements in vector x
- ls() – List objects in current environment
- paste(x) – Concatenate vectors after converting to character
- range(x) – Returns the minimum and maximum of x
- rep(1,5) – Repeat the number 1 five times
- rev(x) – List the elements of "x" in reverse order
- sign(x) – Returns the signs of the elements of x
- sort(x) – Sort the vector x
- order(x) – List sorted element numbers of x
- tolower(), toupper() –
  - Convert string to lower/upper case letters
- unique(x) –
  - Remove duplicate entries from vector.

Functions

- format(x, digits, nsmall, scientific)
  - Formating x (treated as string)
- floor(x), ceiling(x), round(x) -
  - Rounding functions
- Sys.time() – Return system time
- Sys.Date() – Return system date
- getwd() – Return working directory
- setwd() – Set working directory
- list.files() – List files in a give directory
- file.info() – Get information about files

- **Built-in constants:**
- pi, letters, LETTERS
  - # Pi, lower & uppercase letters, e.g. letters[7] = "g"
- month.abb , month.name
  - Abbreviated & full names for months

Packages in R

- 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
- After installing package also have to load it in workspace using **library('packagename')**

- Install Package
- Syntax :

  install.packages('Pckage Name')

  E.g : Install package "XML".

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

- Library locations

  ```
  .libPaths()
  ```

- List of all the packages installed

  ```
  library()
  ```

Importing Data

- Get current working directory

```
print(getwd())
```

- Set current working directory.

```
setwd('C:/Users/Documents/DS')
```

- Copy below code in text file & save it as input.csv

```
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
```

Csv file

- **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)
```

- Validating the import

```
print(is.data.frame(data))
print(ncol(data))
print(nrow(data))
```

- **Analyzing the Data**
  - Get the maximum salary.
  - Get the details of the person with max salary.
  - Get all the people working in IT department.
  - Get the persons in IT department whose salary is greater than 600
  - Extract only name & start date column.

Csv file

- Get the max salary from data frame.

```
sal <- max(data$salary)
print(sal)
```

- Get the max salary from data frame.

```
sal <- max(data$salary)
print(sal)
```

- Get the details of person in IT dept.

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

- Get the details of person in IT dept having salary greater than 600.

```
info <- subset(data, salary > 600 & dept
== "IT")
print(info)
```

- Extract only name & start date of employee

```
S1 <- c(data$name,
data$start_date)

print(S1)
```

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 - xlsx, openxlsx, XLConnect.

- Install xlsx library

```
install.packages("xlsx")
library("xlsx")
```

- Read the first worksheet in the file input.xlsx.

```
data <- read.xlsx("input.xlsx",
sheetIndex = 1)
print(data)
```

XML file

- 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.
- Save below code as input.xml using text editor.

```
<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 file

- Install package

```
install.packages("XML")

# Load library in environment
library('XML')

library("methods")

# Parse the XML file
result <- xmlParse(file = "input.xml")
print(result)


# Convert the data to a data frame.
Xml.df <- xmlToDataFrame("input.xml")
print(Xml.df)
```

JSON file

- 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.
- Mostly used for data transfer between database server to frontend.
- Frontend can be Mobile app, Web app or Desktop application.
- Save below code as input.json using text editor.

```
{
    "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 file

- Install `package`

```
Install.packages('rjson')
```

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.

```
Jsdf <- as.data.frame(result)
print(Jsdf)
```

Database : MySql

- The data is Relational database systems are stored in a normalized format.
- R can connect easily to many relational databases like MySql, Oracle,
  Sql server etc. and fetch records from them as a data frame.
- R has a built-in package named "RMySQL" which provides native connectivity between with MySql database.
- Need Xammp server installed with phpmyadmin

- Install Package

```
Install.packages('RMySQL')
library('RMySQL')
```

Connect to DB

```
mysqlconnection = dbConnect(MySQL(), user
= 'root', password = '', dbname =
'db_name',
host = 'localhost')
```

List tables

```
dbListTables(mysqlconnection)
```

Database : MySql

Query the table to get all the rows.

```
result = dbSendQuery(mysqlconnection, "select * from table_name")
```

Store the result in a R data frame object. n = 5 is used to fetch first 5 rows.

```
data  = fetch(result, n = 5)
print(data)
```

Data Pre-Processing

- Many real world data is dirty and needs to be cleaned before used in code. The process of cleaning a dataset is called Data Preprocessing.
- Preprocessing of data includes below steps :
  - 1. Taking care of missing data
  - 2. Categorical Data
  - 3. Splitting data into training and test data sets

```
ID,Country,Age,Salary,Purchased
1,France,44,72000,No
2,Spain,27,48000,Yes
3,Germany,30,54000,No
4,Spain,38,61000,No
5,Germany,40,NA,Yes
6,France,35,58000,Yes
7,Spain,NA,52000,No
8,France,48,79000,Yes
9,Germany,50,83000,No
10,France,37,67000,Yes
```

Data Pre-Processing

```
# Importing Data
    dataset <- read.csv("Data.csv")
# Showing the dataset
    print(dataset)
# So missing values present in both Age and Salary Columns taking care of missing values.
# By replacing it to the average value for non NA entries.
    dataset$Age <- ifelse(is.na(dataset$Age),
                          ave(dataset$Age, FUN = function(x)
                            mean(x, na.rm = TRUE)),
                          dataset$Age)

    dataset$Salary <- ifelse(is.na(dataset$Salary),
                          ave(dataset$Salary, FUN = function(x)
                            mean(x, na.rm = TRUE)),
                          dataset$Salary)
```

Data Pre-Processing

- Sometimes it happens that data is categorical. Being a categorical data we cannot take it into consideration for predictive analysis (e.g Regression), hence need to be encoded in numerical format.
- factor() provides label property to encode categorical data it numeric data.

```
R&D,Admin,Marketing,State,Profit
165,136,471,New,York,192
162,151,443,California,191
153,101,407,Florida,191
144,118,383,New,York,182
142,913,366,Florida,166
131,998,362,New,York,156
134,147,127,California,156
130,145,323,Florida,155
120,148,311,New,York,150
```

Data Pre-Processing

```
# Encoding categorical data
dataset$Country = factor(dataset$Country,
                         levels = c('France', 'Spain', 'Germany'),
                         labels = c(1, 2, 3))

dataset$Purchased = factor(dataset$Purchased,
                           levels = c('No', 'Yes'),
                           labels = c(0, 1))
```

Data Pre-Processing

- For predictive analysis random sampling has to be done to prove an hypothesis or to predict an accurate outcome into training & testing dataset, dataset values should not be biased from first / later half of data. Hence we can use **sample.split()** for taking random sample of data percentage wise.

```
install.packages("caTools")

library(caTools)

set.seed(123) # model reproduction

split = sample.split(
dataset$Profit, SplitRatio = 0.8)

training_set = subset(dataset,
split == TRUE)

test_set = subset(dataset, split ==
FALSE)
```

Data Wrangling

```
x<- c(1,2,3,4)
(sample(x, 15 ,replace = T))

mydata = data.frame(
                Q1 = sample(1:6, 15, replace = TRUE),
                Q2 = sample(1:6, 15, replace = TRUE),
                Q3 = sample(1:6, 15, replace = TRUE),
                Q4 = sample(1:6, 15, replace = TRUE),
                Age = sample(1:3, 15, replace = TRUE)
              )
print(mydata)
```

Data Wrangling

- 1. Calculate basic descriptive statistics

  ```
  summary(mydata)
  View(mydata)
  ```

- 2. Lists name of variables in a dataset

  ```
  names(mydata)
  ```

- 3. Calculate number of rows in a dataset

  ```
  nrow(mydata)
  ```

- 4. Calculate number of columns in a dataset

  ```
  ncol(mydata)
  ```

- 5. List structure of a dataset

  ```
  str(mydata)
  ```

- 6. See first 6 rows of dataset

  ```
  head(mydata)
  ```

- 7. First n rows of dataset

  ```
  head(mydata, n=5)
  ```

- 8. All rows but the last row

  ```
  head(mydata, n= -1)
  ```

Data Wrangling

- 11. All rows but the first row

```
tail(mydata, n= -1)
```

- 12. Select random rows from a dataset

```
library(dplyr)
sample_n(mydata, 5)
```

- 13. Selecting N% random rows

```
library(dplyr)
sample_frac(mydata, 0.1) # selects 10% random rows from data frame.
```

- Number of missing values

```
colSums(is.na(mydata))
```

- Number of missing values in a single variable

```
sum(is.na(mydata$Q1))
unique(mydata$Q1)
```

- Feature names of mydata

```
colnames(mydata)
```

- Rownames of mydata

```
rownames(mydata)
```

Data Wrangling

- Statistical Summary of data

```
summary(mydata)
```

- Correlation between features

```
cor(mydata)
```

- Mean of particular feature

```
y <- c(1,2,3,4,NA)
mean(y)
mean(mydata$Q1, na.rm=TRUE)
```

- Filter Package :

```
install.packages('dplyr')
```

- # Filter functions

```
mtcars
mtcars$mpg>20
mtcars[mtcars$mpg>20,c("mpg","hp")]

attach(mtcars)
(mpg20 <- mtcars$mpg > 20)
(mpg20 <- mpg > 20)
detach(mtcars)

(subset(mtcars, mpg>20, c("mpg", "hp")))
(subset(mtcars, mpg==max(mpg))
(subset(mtcars, select=c("mpg", "hp"))
```

Data Wrangling

- Filter

```
filter(mtcars, mpg>20)


drawrownames <- function(.data) .data %>% do( mutate (. , Name_of_my_cars =
                rownames(.)))
mtcars %>% drawrownames() %>% filter(mpg>20)


# Select
(select(mtcars, mpg, hp))


# Arrange
(arrange(mtcars, mpg))
(arrange(mtcars, desc(mpg)))
```

Data Wrangling

- # summarise

```
(summarise(mtcars, Average  = mean(mpg), Median=median(mpg)))
```

- # Group by

```
group_by(mtcars, cyl) %>% summarise(Count=n())
```

- # Chaining

```
mtcars %>% filter(mpg > 20) %>% select(mpg, hp)
mtcars %>% filter(mpg > 20) %>% select(mpg, hp) %>% arrange(desc(mpg))
```

Data Wrangling

- # Mutate (Creating new variable)

```
(mtcars %>%
  select(mpg, cyl)%>%
  mutate(newvariable = mpg*cyl))
```

  - #or

```
newvariable <- mynewdata %>% mutate(newvariable = mpg*cyl)
```

- Rename

```
mynewdata <- mtcars
mynewdata %>% rename(miles = mpg)
```

Data Cleaning

- Cleaning / Transforming data into relevant format before starting with analysis.
- Save below code as unnamed.txt in your working directory.

```
## Input
21,6.0
42,5.9
18,5.7*
21,NA
```

- Import file in R studio

```
(person <- read.csv("unnamed.txt"))


# Print the dataframe check for errors
# Give column names & Header false
(person <- read.csv(file = "unnamed.txt",
header = FALSE, col.names =
c("age","height") ))


# Validate the structure
str(person)
```

Data Cleaning

```
(person <- read.csv("unnamed.txt",header=FALSE,colClasses=c('numeric','numeric')))

(person <- read.csv(file = "unnamed.txt",
                    header = FALSE, col.names = c("age","height"),
                    stringsAsFactors = FALSE))

(subset(person , height == '5.7*' | age == '5.7*'))
print(person)
str(person)
person$height[3] <- 5.7

# Data Loss
person$height <- as.numeric(person$height)
print(person)
```

Data Cleaning

- ## Input
- Save below file as daltons.txt

%% Data on the Dalton Brothers

Gratt,1861,1892

Bob,1892

1871,Emmet,1937

% Names, birth and death dates

- File contains Names, Birth, Death Dates of Daltons Brothers.
- Organize data in proper format of Name, Birth, Death Dates
- Rule List to follow
  - Birth dates < 1890
  - Death dates > 1890

Data Cleaning

- **## Code**
  - Import File check for unwanted data & remove them

```
(txt <- read.csv("dalton.txt") )
(txt <- readLines("dalton.txt"))
(I <- grepl("^%", txt))
(dat <- txt[!I])
(fieldList <- strsplit(dat ,
                       split = ","))
```

```
assignFields <- function(x){
    out <- character(3)
    i <- grepl("[[:alpha:]]",x)
    out[1] <- x[i]
    i <- which(as.numeric(x) < 1890)
    out[2] <- ifelse(length(i)>0, x[i], NA)
    i <- which(as.numeric(x) > 1890)
    out[3] <- ifelse(length(i)>0, x[i], NA)
    out
}
```

Data Cleaning

```
standardFields <- lapply(fieldList, assignFields)
print(standardFields)

(M <- matrix(unlist(standardFields), nrow=length(standardFields), byrow=TRUE))

(colnames(M) <- c("name","birth","death"))
(daltons <- as.data.frame(M, stringsAsFactors=FALSE))
str(daltons)
daltons = transform(daltons,birth = as.numeric(birth),death = as.numeric(death))

print(daltons)
```

Case Study 1

- Jack Sparrow's Team :
  - Determining the health of the crew is an important part of any inventory of the ship.
  - Here's a vector containing the number of limbs each member has left, along with their names.

```
limbs <- c(4, 3, 4, 3, 2, 4, 4, 4)
names(limbs) <- c('One-Eye', 'Peg-Leg', 'Smitty', 'Hook', 'Scooter', 'Dan',
                  'Mikey', 'Jack Sparrow')
```

  - An average closer to 4 would be nice, but this will have to do.

```
mean(limbs)
```

  - Here's a barplot of that vector:

```
barplot(limbs)
abline(h=mean(limbs))
```

Case Study 1

- **Davy Jones Team**

```
limbs <- c(4, 3, 4, 3, 2, 4, 4, 14)
names(limbs) <- c('One-Eye', 'Peg-Leg', 'Smitty', 'Hook','Scooter', 'Dan',
                  'Mikey', 'Davy Jones')
mean(limbs)
barplot(limbs)
```

- It may be factually accurate to say that our crew has an average of 4.75 limbs,
- but it's probably also misleading.

```
abline(h=mean(limbs))
median(limbs)
abline(h=median(limbs))
```

Case Study 2

- Scenario: You are a Data Scientist working for a consulting firm. One of your colleagues from the Auditing department has asked you to help them assess the financial statement of organisation X.

- You have been supplied with two vectors of data: monthly revenue and monthly expenses for the financial year in question.

- #Data

```
revenue <- c(14574.49, 7606.46, 8611.41, 9175.41, 8058.65, 8105.44,
             11496.28, 9766.09, 10305.32, 14379.96, 10713.97, 15433.50)

expenses <- c(12051.82, 5695.07, 12319.20, 12089.72, 8658.57, 840.20,
              3285.73, 5821.12, 6976.93, 16618.61, 10054.37, 3803.96)
```

Case Study 2

- Your task is to calculate the following
  - # financial metrics:
  - # - profit for each month
  - # - profit after tax for each month (the tax rate is 30%)
  - # - profit margin for each month - equals to profit a after tax divided by revenue
  - # - good months - where the profit after tax was greater than the mean for the year
  - # - bad months - where the profit after tax was less than the mean for the year
  - # - the best month - where the profit after tax was max for the year
  - # - the worst month - where the profit after tax was min for the year

Case Study 2

- All results need to be presented as vectors.
- Results for dollar values need to be calculated with $0.01 precision, but need to be presented in Units of $1,000 (i.e. 1k) with no decimal points.
- Results for the profit margin ratio need to be presented in units of % with no decimal points.
- Note: You colleague has warned you that it is okay for tax for any given month to be negative (in accounting terms, negative tax translates into a deferred tax asset).

Case Study 2

- #Solution

```
#profit for each month
profit <- revenue - expenses
print(profit)

#profit after tax each month (the tax rate is 30%)
tax <- round(profit * 0.3 , digits=2)
print(tax)

profit.after.tax <- profit - tax
print(profit.after.tax)
```

Case Study 2

```
#profit margin for each month - equals to profit a after tax divided by revenue
profit.margin <- round(profit.after.tax / revenue, 2) * 100
print(profit.margin)

#good months - where the profit after tax was greater than the mean for the year
mean_pat <- mean(profit.after.tax)
print(mean_pat)

good.months <- profit.after.tax > mean_pat
print(good.months)
```

Case Study 2

```
#bad months - where the profit after tax was less than the mean for the year
bad.months <- !good.months
print(bad.months)

#the best month - where the profit after tax was max for the year
best.month <- profit.after.tax == max(profit.after.tax)
print(best.month)

#the worst month - where the profit after tax was min for the year
worst.month <- profit.after.tax == min(profit.after.tax)
print(worst.month)
```

Case Study 2

```
#units of thousand
revenue.1000 <- round(revenue / 1000)
expenses.1000 <- round(expenses / 1000)
profit.1000 <- round(profit / 1000)
profit.after.tax.1000 <- round(profit.after.tax / 1000)
```

Case Study 2

- #output

```
print( revenue.1000 )
print( expenses.1000 )
print( profit.1000 )
print( profit.after.tax.1000 )
print( profit.margin )
print( good.months )
print( bad.months )
print( best.month )
print( worst.month )
```

- Create a dataframe of result

```
d <- data.frame(
  revenue.1000,
  expenses.1000,
  profit.1000,
  profit.after.tax.1000,
  profit.margin,
  good.months,
  bad.months,
  best.month,
  worst.month
)
print(d)
```

Instructor Contact Information

nirajshar67@gmail.com /
rocky@suvenconsultants.com

9167687087 / 9892544177

10 am – 7 pm

https://www.linkedin.com/in/niraj7654/