DEPARTMENT OF STATISTICS & OPERATIONS RESEARCH

AMU ALIGARH -202002 , U. P. (INDIA)

BIG DATA ANALYTICS (DSM 3002)

M.SC. III SEMESTER(DATA SCIENCE)

2022-23

DR ZAHID AHMED ANSARI

COURSE OBJECTIVE

- To study the specialized aspects of big data analytics.

COURSE OUTCOME

- On successful completion of this course, the students will be able to

1. Identify big data and its real-life implications
2. Analyze the problem of big data with the help of R and Hadoop

SYLLABUS

Contents	
Unit-I	**Introduction:** Big data, Big data toolbox: Hadoop and Spark, traditional limitations with the software(s), expanding memory with the bigmemory package, parallel computing, Boosting R performance for big data
Unit-II	**Hadoop architecture:** Hadoop Distributed File System, MapReduce framework, single- node Hadoop in Cloud, HDInsight-a multi-node Hadoop cluster on Azure, smart energy meter readings analysis example-using R on HDInsight cluster.
Unit-III	**Relational Database Management Systems:** SQLite with R, MariaDB with R on amazon EC2 instance, PostgreSQL with R on amazon RDS. Non-Relational (NoSQL) Databases: Processing big data using MongoDB, HBaseMicroprocessor in industry
Unit-IV	**Spark for big data analytics:** Spark with a multi-node HDInsight cluster, Reading the data into HIDES and Hive. Machine learning methods for big data: GLM examples, naive Bayes with H2O on Hadoop, neural networks with H2O on Hadoop

RECOMMENDED BOOKS

No	Title	Author
1	**Big Data Analytics with R**	Walkowiak, S. (2016) Packt open source
2	**BIG DATA ANALYTICS: A Practical Guide for Managers**	Pries, K. H. and Dunnigan, R, (2015) CRC Press
3	**Big Data Analytics with Rand Hadoop**	Prajapati, V. (2013) Packt open source

UNIT-I INTRODUCTION

- What is Big data?
- Big Data toolbox: Hadoop and Spark and R,
- Traditional limitations with the software(s),
- Expanding memory with the bigmemory package,
- Parallel computing using R
- Boosting R performance for big data

What is Big Data?

RAPID GROWTH OF DATA

- Every time **Leo Messi** scores at **Camp Nou** in **Barcelona**, almost one hundred thousand Barca fans cheer in support of their most prolific striker.
- Social media services such as Twitter, Instagram, and Facebook are instantaneously flooded with comments, views, opinions, analyses, photographs, and videos of yet another wonder goal from the Argentinian goal scorer.
- One such goal, scored in the semifinal of the UEFA Champions League, against Bayern Munich in May 2015, generated more than 25,000 tweets per minute in the United Kingdom alone, making it the most tweeted sports moment of 2015 in this country.

RAPID GROWTH OF DATA

- A goal like this creates a widespread excitement, not only among football fans and sports journalists.
- It is also a powerful driver for the marketing departments of numerous sportswear stores around the globe, who try to predict, with a military precision, day-today, in-store, and online sales of Messi's shirts, and other FC Barcelona related memorabilia.
- At the same time, major TV stations attempt to outbid each other in order to show forthcoming Barca games, and attract multi-million revenues from advertisement slots during the half-timebreaks.
- For a number of industries, this one goal is potentially worth much more than Messi's 20 million Euro annual salary.

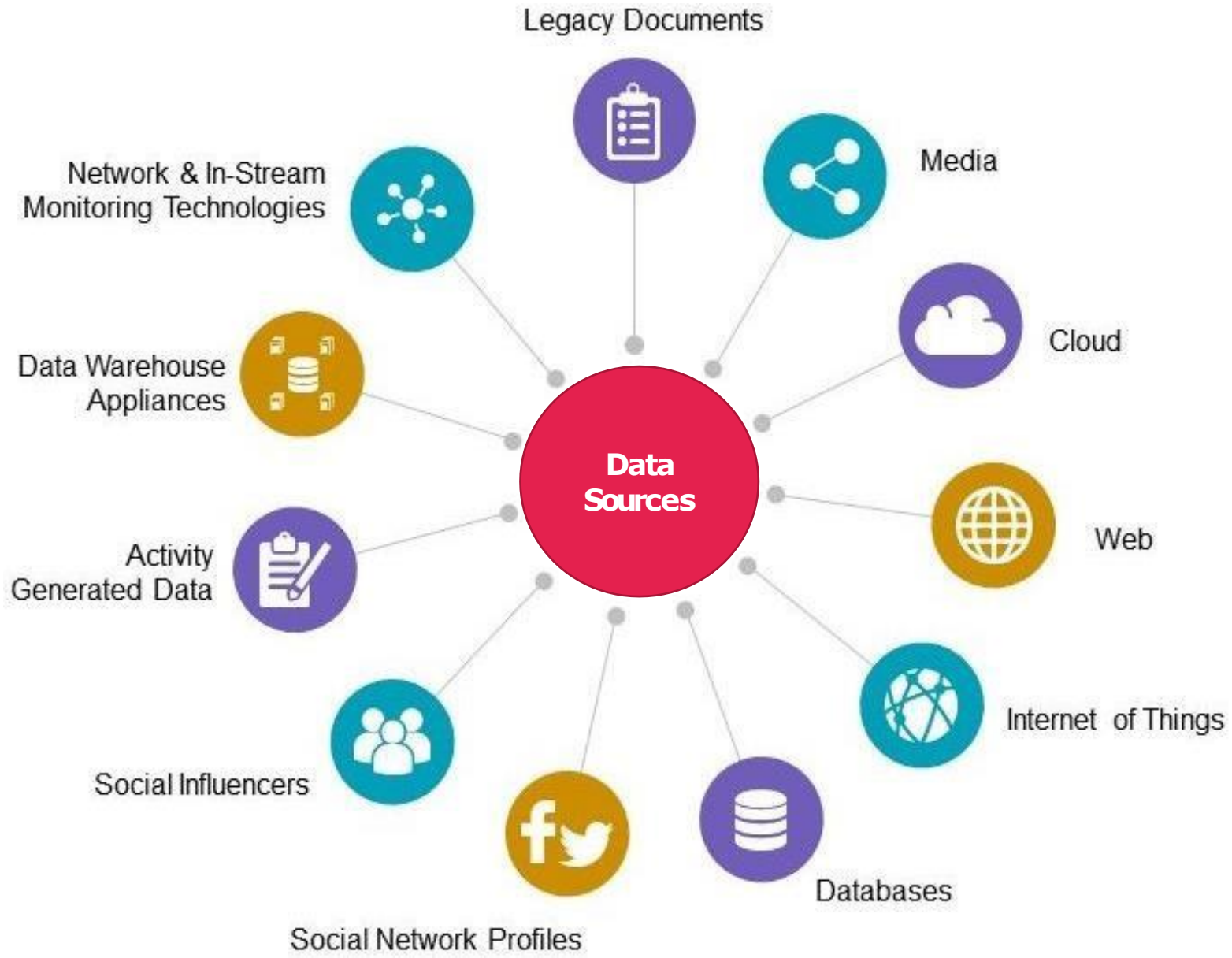
DATA ANALYTICS

- This one moment creates an abundance of information, which needs to be somehow collected, stored, transformed, analyzed, and redelivered in the form of yet another product, for example,
  - Sports news with a slow-motion replay of Messi's killing strike,
  - Additional shirts dispatched to sportswear stores, or
  - Sales spreadsheet and a Marketing briefing outlining Barca's TV revenue figures.

ABUNDANCE OF DATA

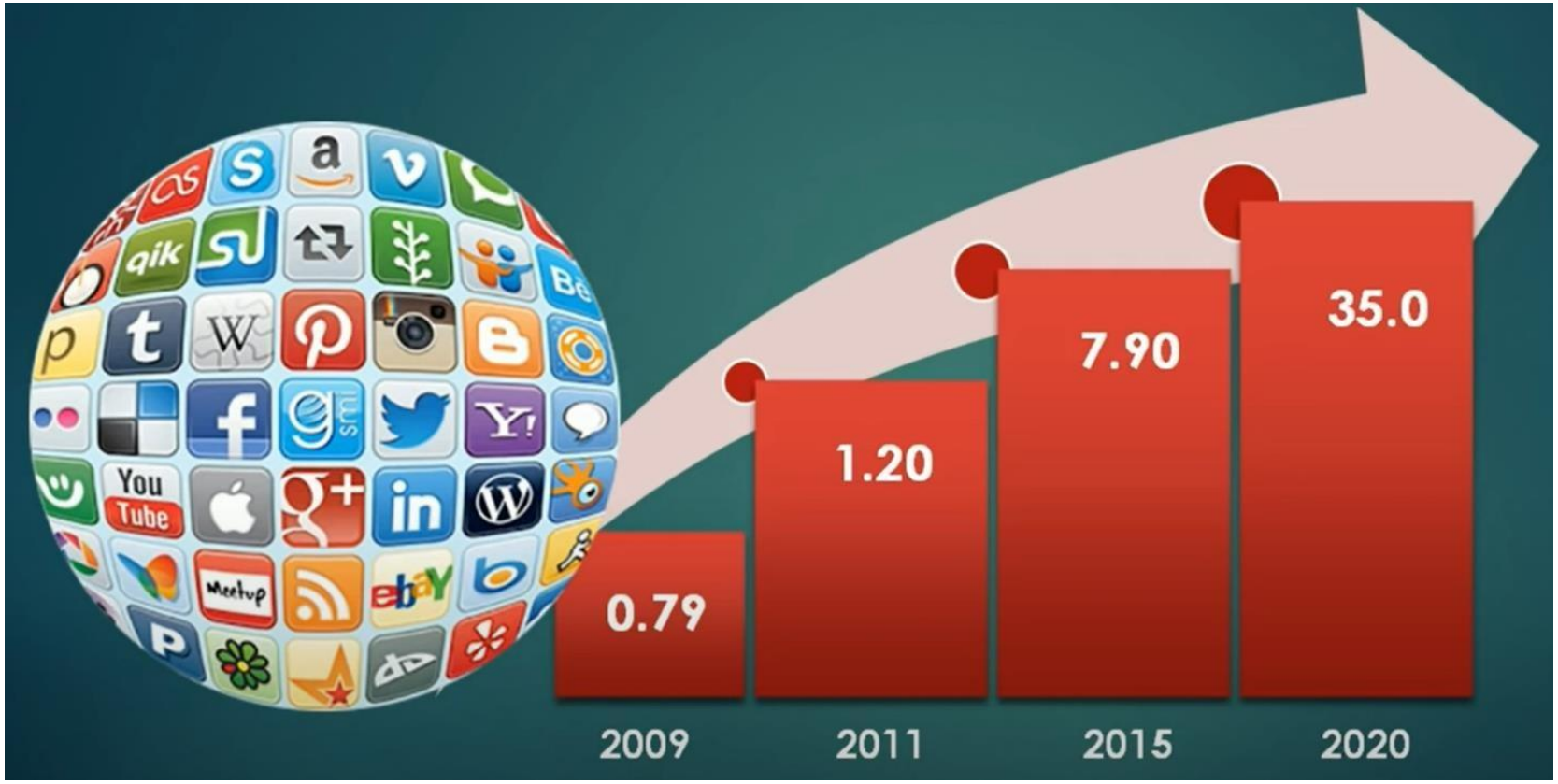
- Such moments, like memorable Messi's goals against Bayern Munich, happen on a daily basis. Actually, they are probably happening right now.
- If you want to check what currently makes the world buzz, go to the Twitter web page and click on the **Moments** tab to see the most trending hashtags and topics at this very moment.
- Each of these less, or more, important events generates vast amounts of data in many different formats, from social media status updates to YouTube videos and blog posts to mention just a few.
- But here is the first problem: We can very quickly fill up all the available storage on our hard drives, or run out of processing power and memory resources to crunch the collected data.
- If you end up having such issues when managing your data, you are probably dealing with something that has been vaguely denoted as **Big Data**.

Rapid growth of data



WHY DATA ANALYTICS?

Value	Metric	
1	B	Byte
1000	kB	Kilobyte
1000^2	MB	Megabyte
1000^3	GB	Gigabyte
1000^4	TB	Terabyte
1000^5	PB	Petabyte
1000^6	EB	Exabyte
1000^7	ZB	**Zettabyte**



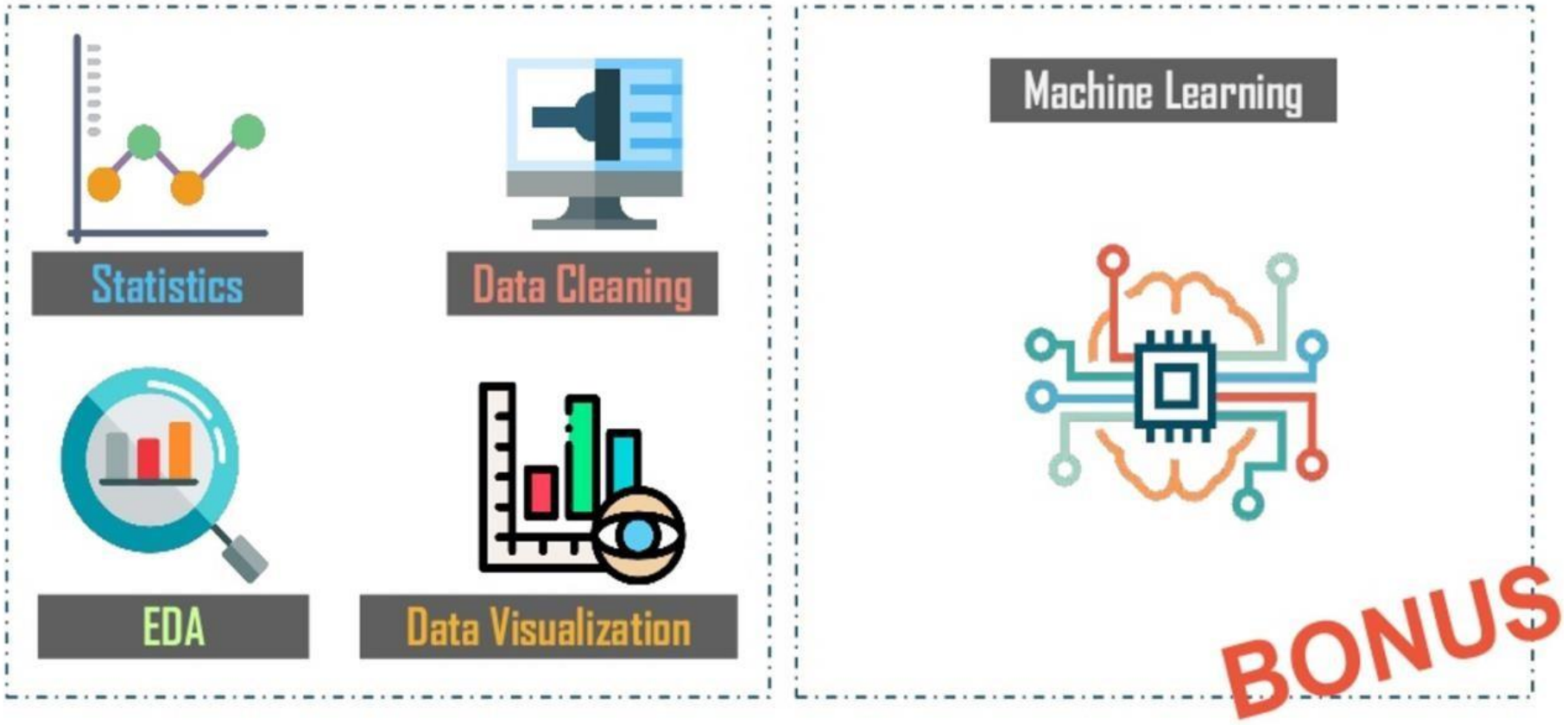
There is data explosion.
We are downing in data and struggling to discover useful information from it

WHAT IS DATA ANALYTICS?

- Process of Examining data sets in order to draw relevant conclusions about the information they contain

WHAT A DATA ANALYST DOES?

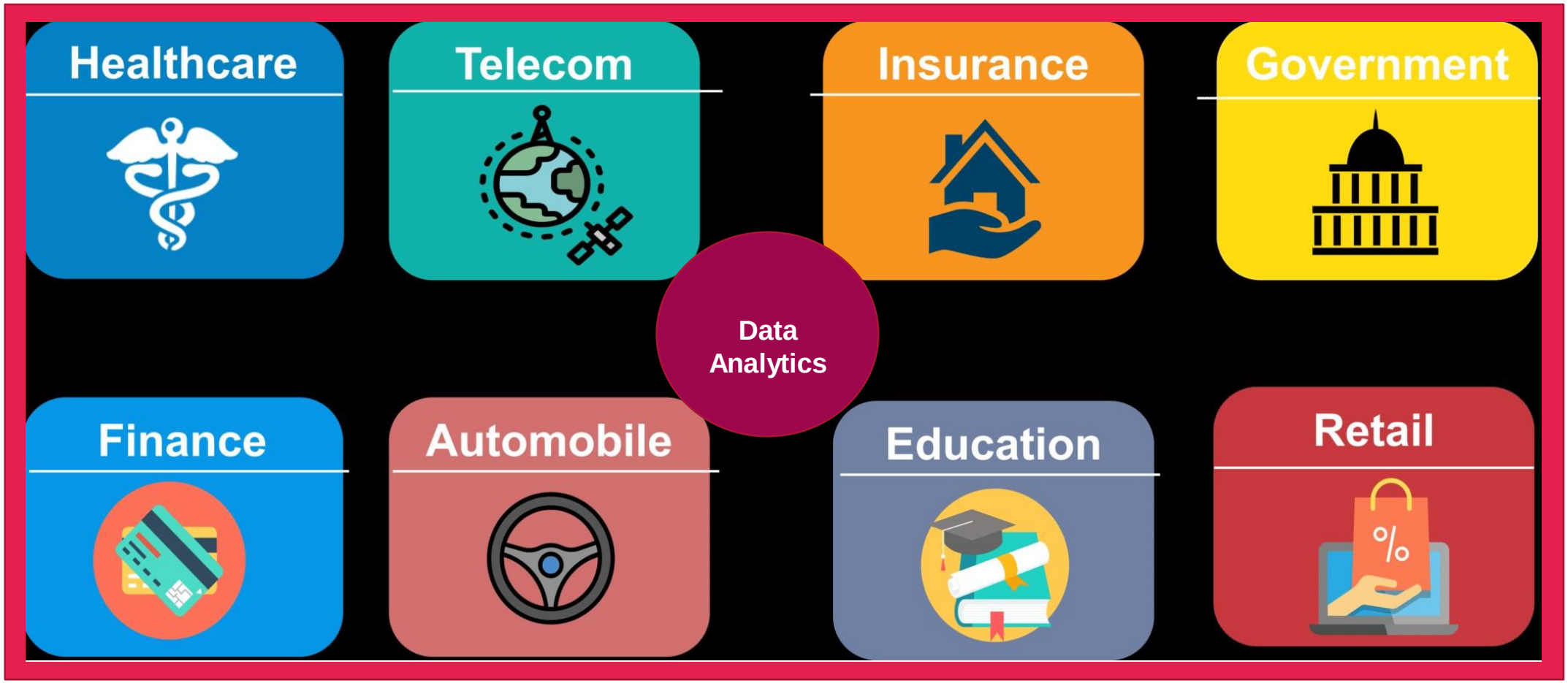
DATA ANALYST SKILLS



Machine Learning

Machine learning is the process of teaching a computer system how to make accurate predictions through experience when fed with the data without explicitly being programmed. Machine Learning Knowledge is extra bonus.

SOME APPLICATION DOMAINS



BIG DATA

- If you ask ten, randomly selected, students what they understand by the term *Big Data* they will probably give you ten, very different, answers.
- By default, most will immediately conclude that Big Data has something to do with the size of a data set, the number of rows and columns etc..
- But it's when we inquire about when exactly *normal* data becomes *Big?*
  - Some will try to convince you that even 100 MB is quite a big file or big enough to be scary.
  - Some others will probably say that 1 GB heavy data would definitely make them anxious.
  - Some will suggest that 5 GB would be problematic, as even Excel suddenly slows down or doesn't want to open the file.
  - In many areas of medical science (e.g. genome studies) file sizes easily exceed 100 GB each
  - Most industry data centers deal with data in the region of 2 TB to 10 TB at a time.
  - Leading organizations such as Google, Facebook, or YouTube manage petabytes of information on a daily basis.
- What is then the threshold to qualify data as Big?

WHEN NORMAL DATA BECOMES BIG?

- **"One byte more than you are comfortable with"** is a well-known phrase used by Big Data conference speakers.
- In fact, all our students, whether they said Big Data was as little as 100MB or as much as 10 petabytes, were more or less correct in their responses. As long as an individual (and his/her equipment) is not comfortable with a certain size of data, we should assume that this is Big Data for them.
- ***Big Data any data that cause significant processing, management, analytical, and interpretational problems.***

What is Big Data?

"The basic idea behind the phrase **'Big Data'** is that everything we do is increasingly leaving a digital trace (or data), which we can use and analyze to become smarter The driving forces in this brave new world are access to ever-increasing volumes of data and our ever-increasing technological capability to mine that data for commercial insights."

Bernard Marr

"**Big Data** refers to the dynamic, large and disparate volumes of data being created by people, tools and machines; it requires new, innovative and scalable technology to collect, host and analytically process the vast amount of data gathered in order to derive real-time business insights that relate to consumers, risk, profit, performance, productivity management and enhanced shareholder value."

EY

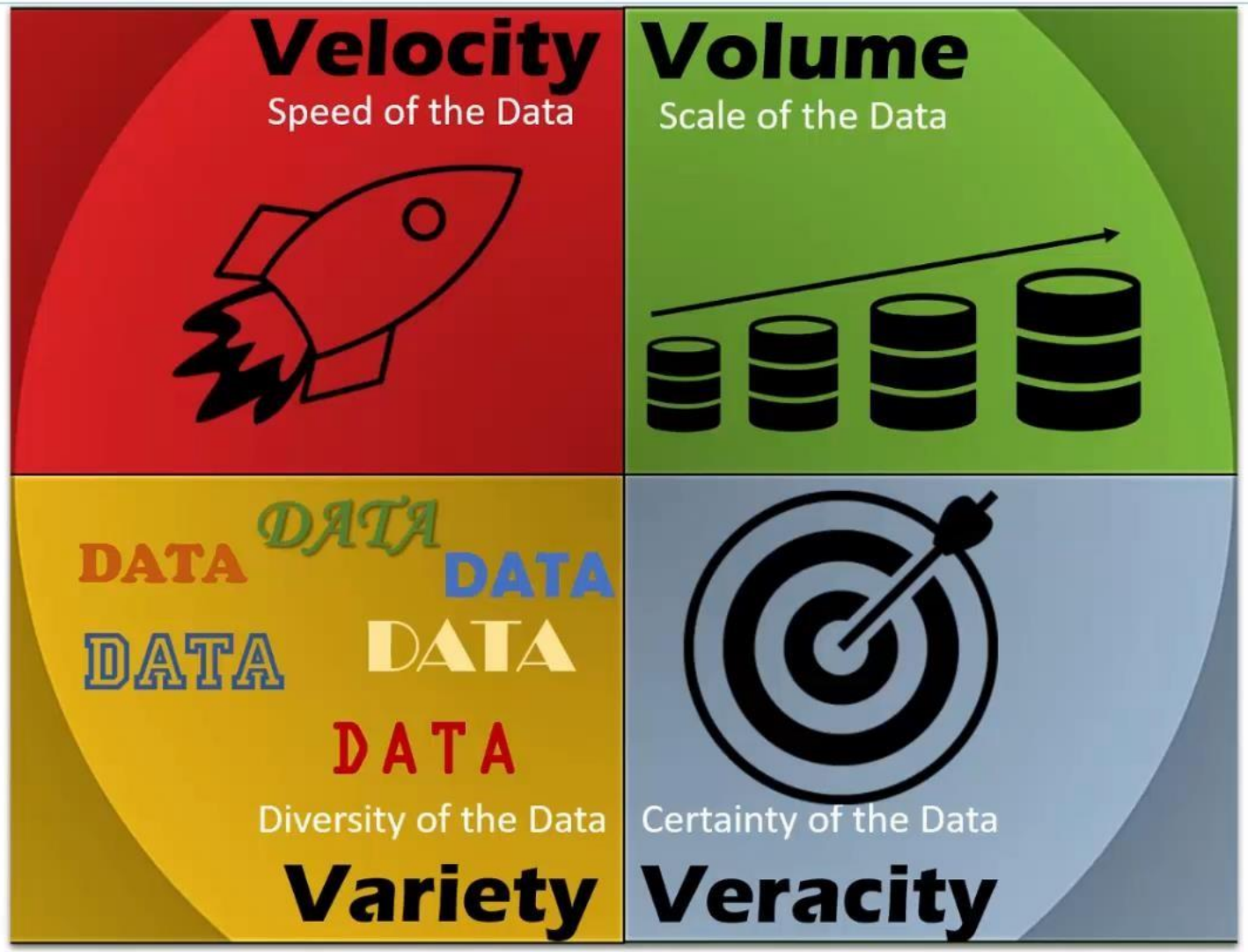
"**Big Data** is high-volume, high-velocity and/or high-variety information assets that demand cost-effective, innovative forms of information processing that enable enhanced insight, decision making, and process automation."

Gartner

"**Big Data** is a collection of data from traditional and digital sources inside and outside your company that represents a source for ongoing discovery and analysis."

Lisa Arthur

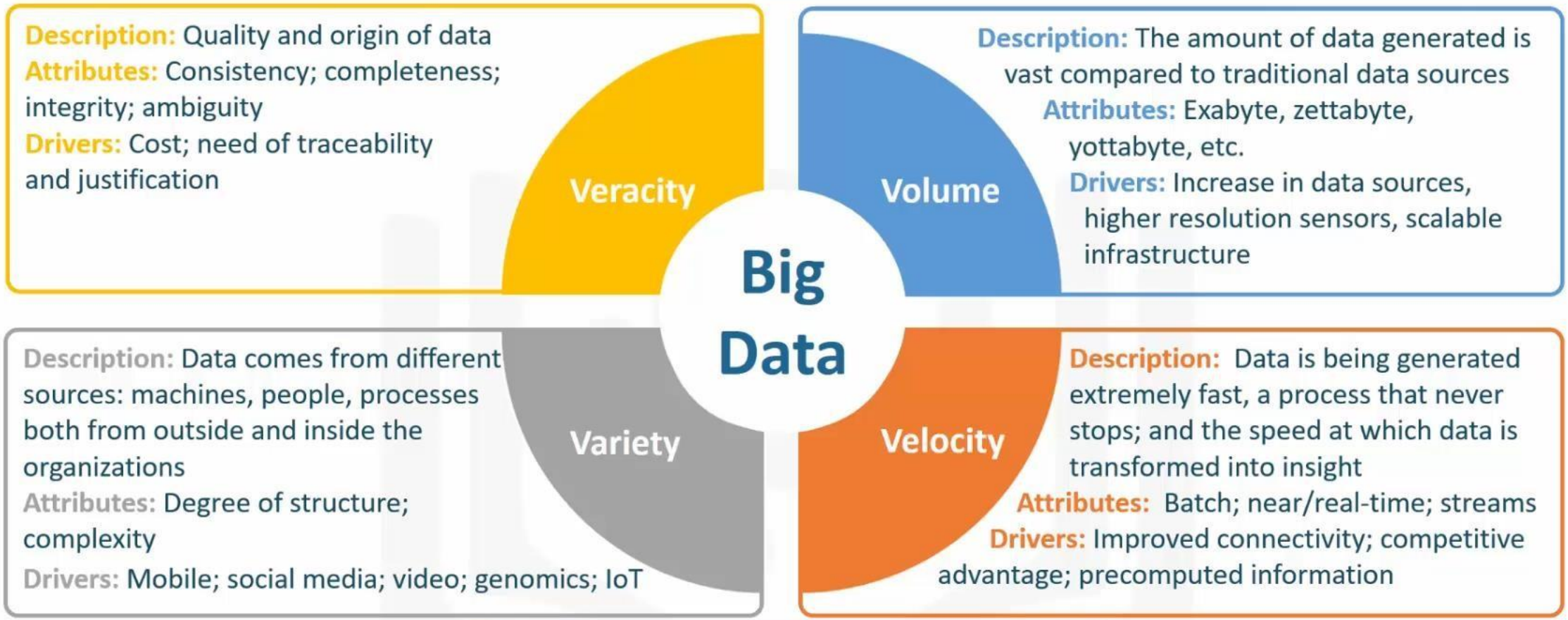
What are the 4 V's of Big Data?



BIG DATA UNIVERSITY

FIRESIDE ANALYTICS INC.
@FIRESIDE_INFO

What are the 5 V's of Big Data?



Source: EY, 2014; ITU, 2013

The 5 V's – Velocity

Source: Data Never Sleeps 4.0 - www.domo.com

Velocity

Every 60 Seconds →

2,430,555
Instagram likes →

972,222
Tinder swipes →

This amount of data is generated every minute. So think how much accumulates over hours, days, and in years?

BIG DATA UNIVERSITY

FIRESIDE ANALYTICS INC.
@FIRESIDE_INFO

The 5 V's - Volume

Source: "Bringing big data to the enterprise" - https://www-01.ibm.com/software/data/bigdata/what-is-big-data.html October 2016

Volume

Everyday, we create about **2.5 Quintillion Bytes** of data

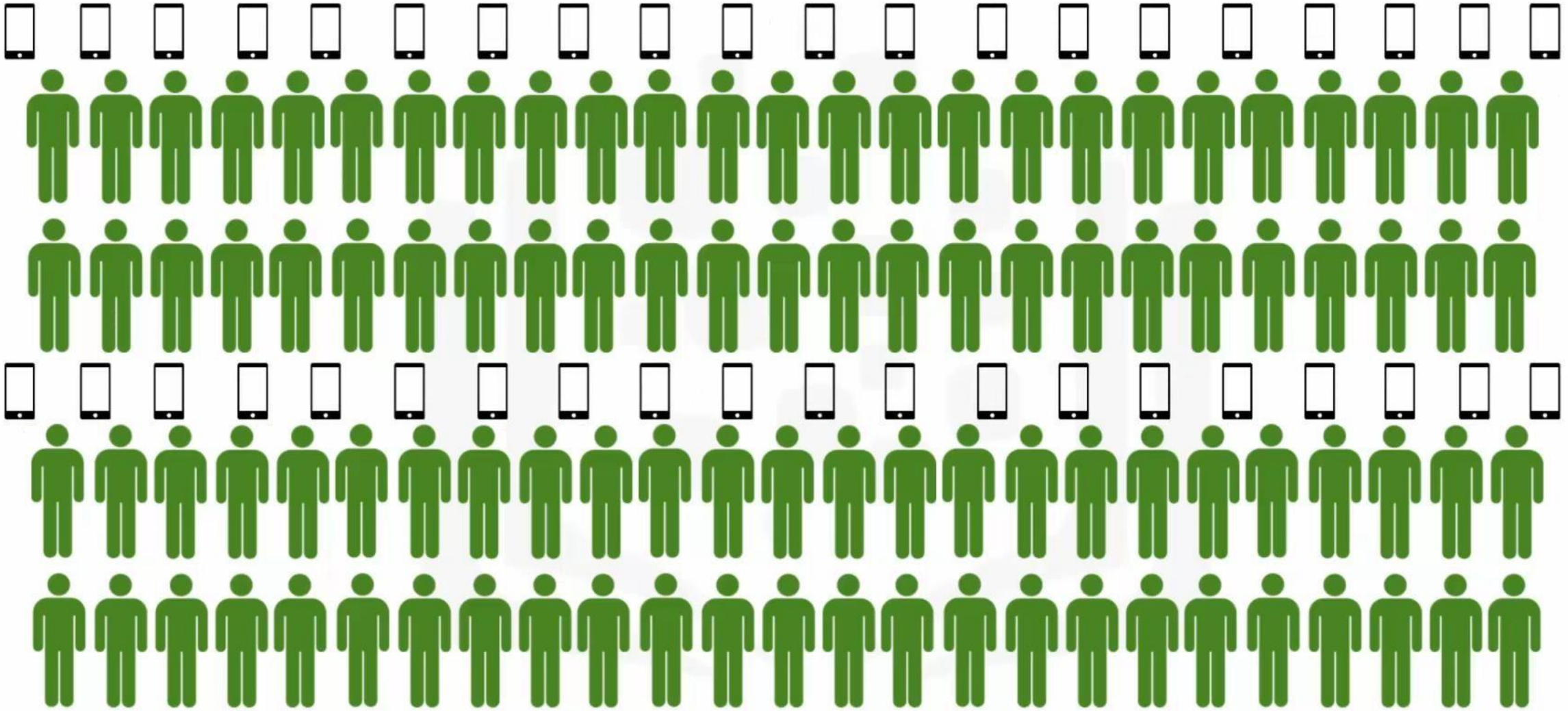
X 10 million

BIG DATA UNIVERSITY

FIRESIDE ANALYTICS INC.
@FIRESIDE_INFO

The 5 V's - Volume

The world population is approximately seven billion people, and vast majority uses digital devices which generate, capture, and store data.



BIG DATA UNIVERSITY

FIRESIDE ANALYTICS INC.
@FIRESIDE_INFO

The 5 V's - Volume

With more than one device, for example, mobile devices, desktop computers, laptops, et cetera, we're seeing even more data being produced.

The 5 V's - Variety

Source: "The biggest data challenges that you might not even know you have", Christie Schneider, Content Manager, IBM Watson - https://www.ibm.com/blogs/watson/2016/05/biggest-data-challenges-might-not-even-know/

Different types of data, text, pictures, film, sound etc.

health data from wearable devices,

Many different types of data from devices connected to the IOT.

BIG DATA UNIVERSITY

The 5 V's - Veracity

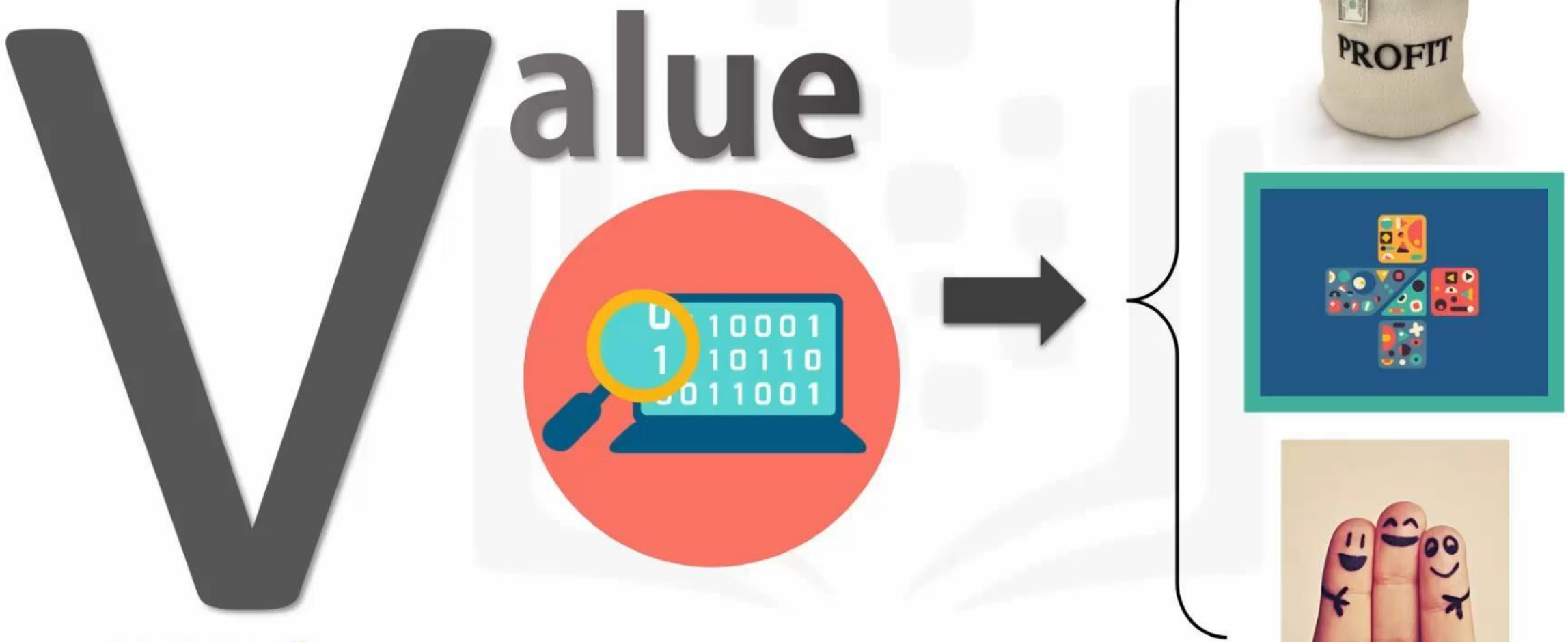
Source: "IBM Software Defined Environment", By Dino Quintero, William M Genovese, KiWaon Kim, Ming Jun MJ Li, Fabio Martins, Ashish Nainwal, Dusan Smolej, Marcin Tabinowski, Ashu Tiwary, IBM Redbooks

80% of data is considered to be unstructured and we must devise ways to produce reliable and accurate insights.

BIG DATA UNIVERSITY

FIRESIDE ANALYTICS INC.
@FIRESIDE_INFO

The Emerging V - Value



BIG DATA UNIVERSITY

FIRESIDE ANALYTICS INC.
@FIRESIDE_INFO

THE EMERGING V IS VALUE.

- This V refers to our ability and need to turn data into value.
- Value isn't just profit.
- It may be medical or social benefits, or customer, employee, or personal satisfaction.
- The main reasons for why people invest time to understand Big Data is to derive value from it.

BIG DATA TOOLBOX
DEALING WITH THE GIANT

- Just like doctors cannot treat all medical symptoms with generic paracetamol and ibuprofen, data scientists need to use more potent methods to store and manage vast amounts of data.
- Knowing already how Big Data can be defined, and what requirements have to be met in order to qualify data as *Big*, we can now take a step forward and introduce a number of tools that are specialized in dealing with these enormous data sets.
- Although traditional techniques may still be valid in certain circumstances, Big Data comes with its own ecosystem of scalable frameworks and applications that facilitate the processing and management of unusually large or fast data.

MOST POPULAR BIG DATA ANALYTICS TOOLS

1. 

2. 

3. 

4. 

5. 

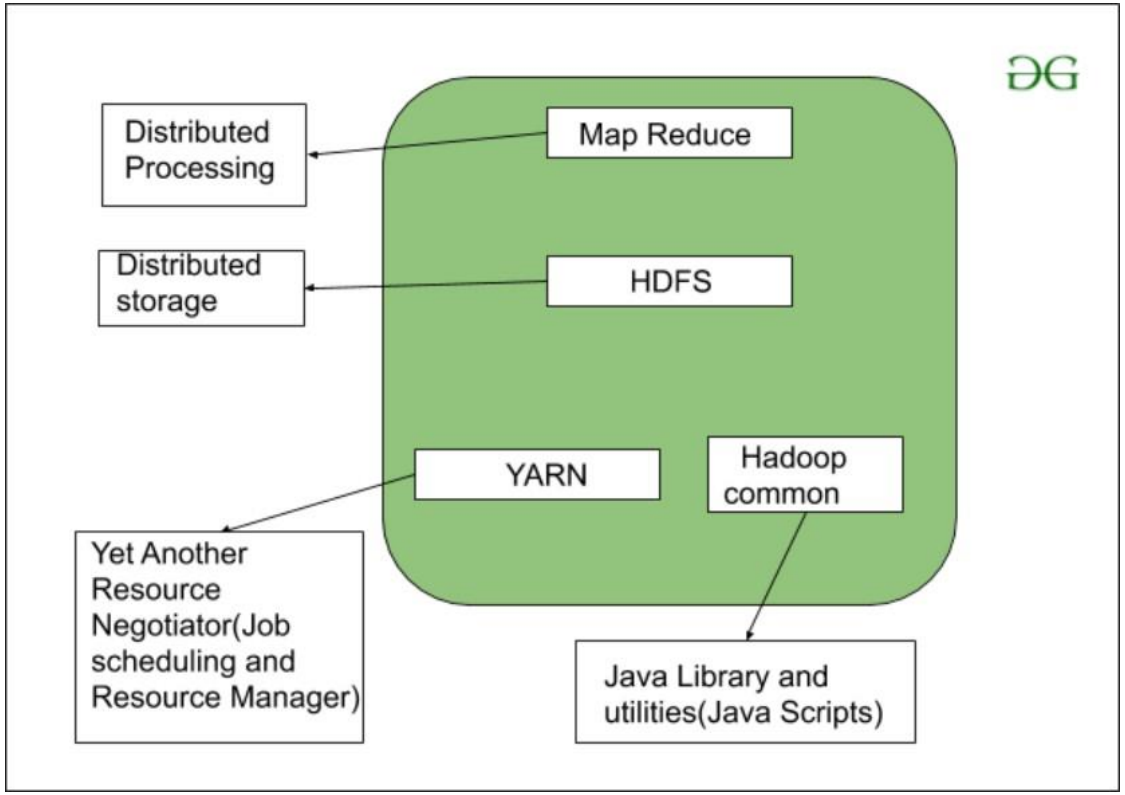
Apache Hadoop for Big Data Analytics

- It’s a Java-based open-source platform that is being used to store and process big data.
- It is built on a cluster system that allows the system to process data efficiently and let the data run parallel.
- It can process both structured and unstructured data from one server to multiple computers.
- **Hadoop** also offers **cross-platform** support for its users. Today, it is the best ***big data analytic tool*** and is popularly used by many tech giants such as Amazon, Microsoft, IBM, etc.

- **Features of Apache Hadoop**
- Free to use and offers an efficient storage solution for businesses.
- Offers quick access via HDFS (Hadoop Distributed File System).
- Highly flexible and can be easily implemented with MySQL, and JSON.
- Highly scalable as it can distribute a large amount of data in small segments.
- It works on small commodity hardware like JBOD or a bunch of disks.

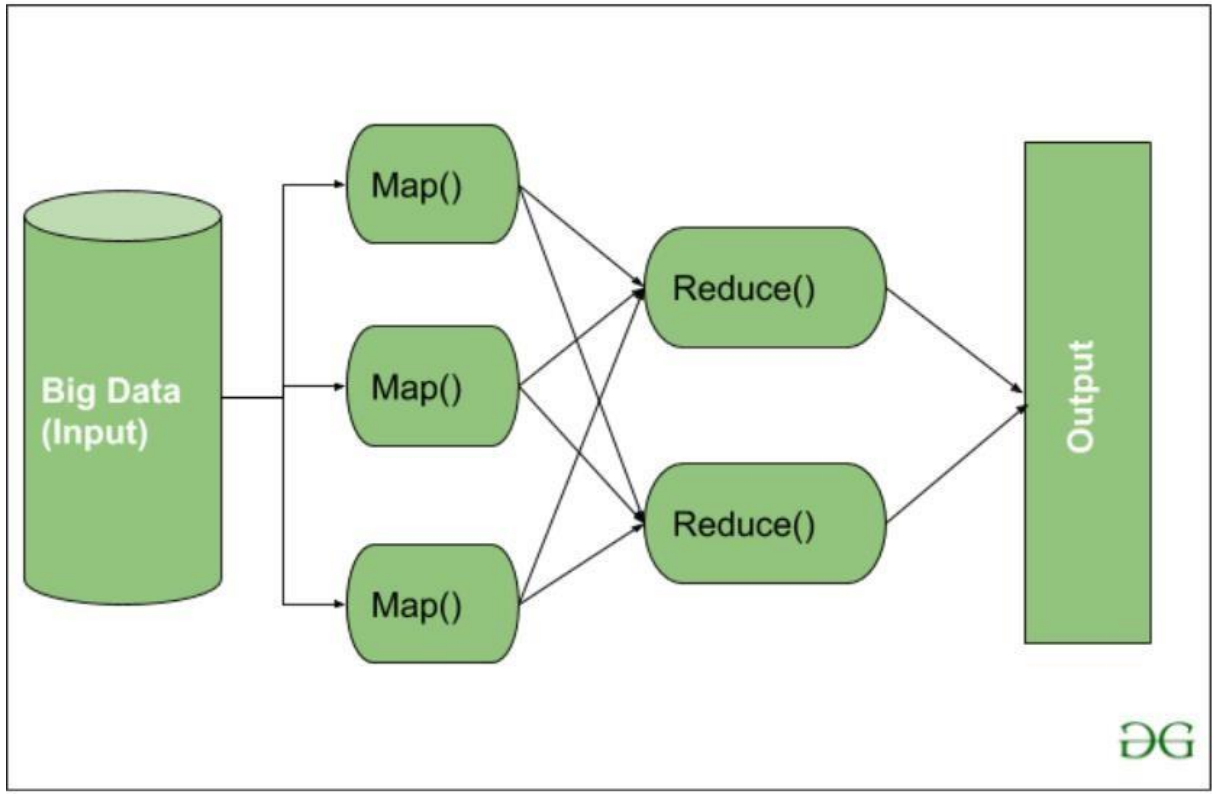
HADOOP – ARCHITECTURE

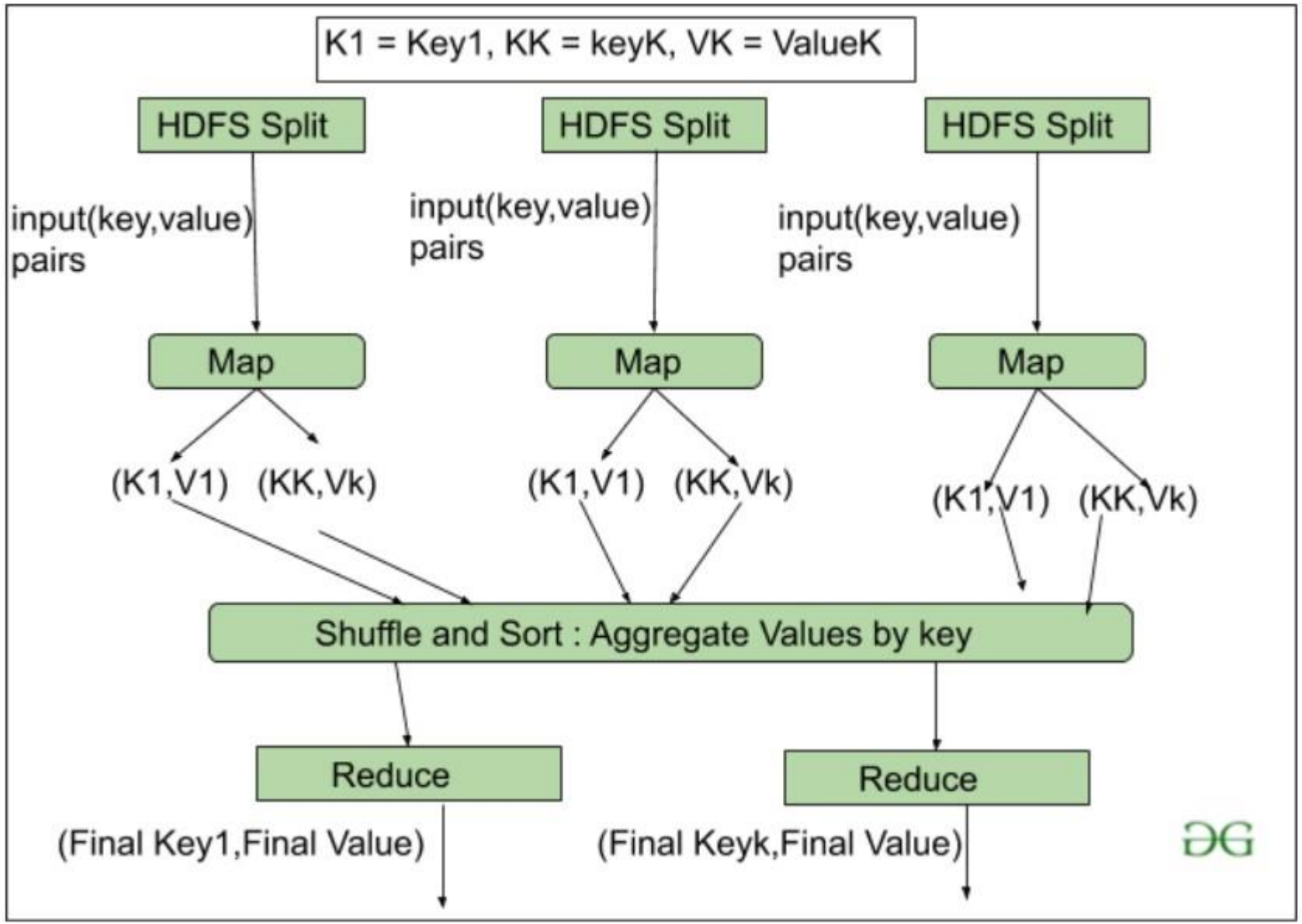
- Hadoop is a framework written in Java that utilizes a large cluster of commodity hardware to maintain and store big size data.
- Hadoop works on MapReduce Programming Algorithm that was introduced by Google.
- The Hadoop Architecture Mainly consists of 4 components.
  - MapReduce
  - HDFS(Hadoop distributed File System)
  - YARN(Yet Another Resource Framework)
  - Common Utilities or Hadoop Common



MAPREDUCE

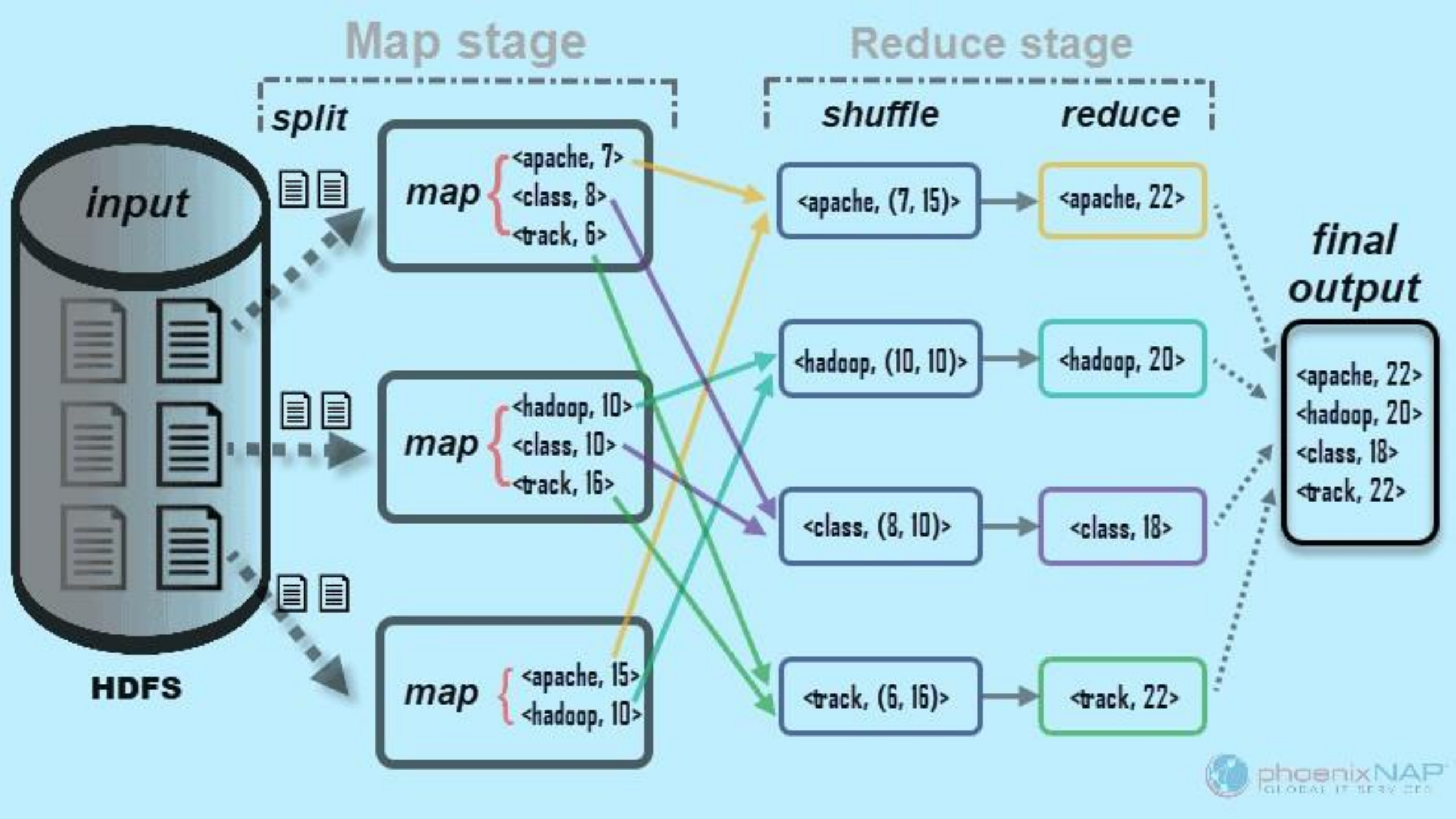
- MapReduce is an Algorithm that is based on the YARN framework.
- The major feature of MapReduce is to perform the distributed processing in parallel in a Hadoop cluster which Makes Hadoop working so fast.
- When you are dealing with Big Data, serial processing is no more of any use.
- MapReduce has mainly 2 tasks which are divided phase-wise:
  - In first phase, **Map** is utilized and
  - in next phase **Reduce** is utilized.




K1 = Key1, KK = keyK, VK = ValueK
HDFS Split
HDFS Split
HDFS Split
input(key,value) pairs
input(key,value) pairs
input(key,value) pairs
Map
Map
Map
(K1,V1) (KK,Vk)
(K1,V1) (KK,Vk)
(K1,V1) (KK,Vk)
Shuffle and Sort : Aggregate Values by key
Reduce
Reduce
(Final Key1,Final Value)
(Final Keyk,Final Value)

HOW HADOOP MAP AND REDUCE WORK

- As the name suggests, MapReduce works by processing input data in two stages – **Map** and **Reduce**.
- To demonstrate this, we will use a simple example with counting the number of occurrences of words in each document.
- The final output we are looking for is: How many times the words Apache, Hadoop, Class, and Track appear in total in all documents.
- The example environment consists of three nodes. The input contains six documents distributed across the cluster. We will keep it simple here, but in real circumstances, there is no limit. You can have thousands of servers and billions of documents.


Map stage
Reduce stage
split
shuffle
reduce
input
HDFS
map
<apache, 7>
<class, 8>
<track, 6>
map
<hadoop, 10>
<class, 10>
<track, 16>
map
<apache, 15>
<hadoop, 10>
<apache, (7, 15)>
<hadoop, (10, 10)>
<class, (8, 10)>
<track, (6, 16)>
<apache, 22>
<hadoop, 20>
<class, 18>
<track, 22>
final output
<apache, 22>
<hadoop, 20>
<class, 18>
<track, 22>
phoenixNAP

HOW HADOOP MAP AND REDUCE WORK

1. First, in the **map stage**, the input data (the six documents) is **split** and distributed across the cluster (the three servers). In this case, each map task works on a split containing two documents. During mapping, there is no communication between the nodes. They perform independently.
2. Then, map tasks create a **<key, value>** pair for every word. These pairs show how many times a word occurs. A word is a key, and a value is its count. For example, one document contains three of four words we are looking for: *Apache* 7 times, *Class* 8 times, and *Track* 6 times. The key-value pairs in one map task output look like this:
   - **<apache, 7>**
   - **<class, 8>**
   - **<track, 6>**

- This process is done in parallel tasks on all nodes for all documents and gives a unique output.

HOW HADOOP MAP AND REDUCE WORK

3. After input splitting and mapping completes, the outputs of every map task are **shuffled**. This is the first step of the **Reduce stage**. Since we are looking for the frequency of occurrence for four words, there are four parallel Reduce tasks. The reduce tasks can run on the same nodes as the map tasks, or they can run on any other node.

   The shuffle step ensures the keys Apache, Hadoop, Class, and Track are sorted for the reduce step. This process groups the values by keys in the form of <key, value-list> pairs.

4. In the **reduce step** of the Reduce stage, each of the four tasks process a <key, value-list> to provide a final key-value pair. The reduce tasks also happen at the same time and work independently.

- In our example from the diagram, the reduce tasks get the following individual results:
  - **<apache, 22>**
  - **<hadoop, 20>**
  - **<class, 18>**
  - **<track, 22>**

HOW HADOOP MAP AND REDUCE WORK

5. Finally, the data in the **Reduce stage** is grouped into one output. MapReduce now shows us how many times the words *Apache*, *Hadoop*, *Class*, and *track* appeared in all documents. The aggregate data is, by default, stored in the HDFS.

- The example we used here is a basic one. MapReduce performs much more complicated tasks.
- Some of the use casesinclude:
  - Turning Apache logs into tab-separated values (TSV).
  - Determining the number of unique IP addresses in weblog data.
  - Performing complex statistical modeling and analysis.
  - Running machine-learning algorithms using different frameworks, such as Mahout.

HDFS

- HDFS(Hadoop Distributed File System) is utilized for storage purpose in a Hadoop cluster.
- It mainly designed for working on commodity Hardware devices (inexpensive devices), working on a distributed file system design.
- HDFS is designed in such a way that it believes more in storing the data in a large chunk of blocks rather than storing small data blocks.
- HDFS in Hadoop provides Fault-tolerance and High availability to the storage layer and the other devices present in that Hadoop cluster.
- Data storage Nodes in HDFS:
  - NameNode(Master)
  - DataNode(Slave)

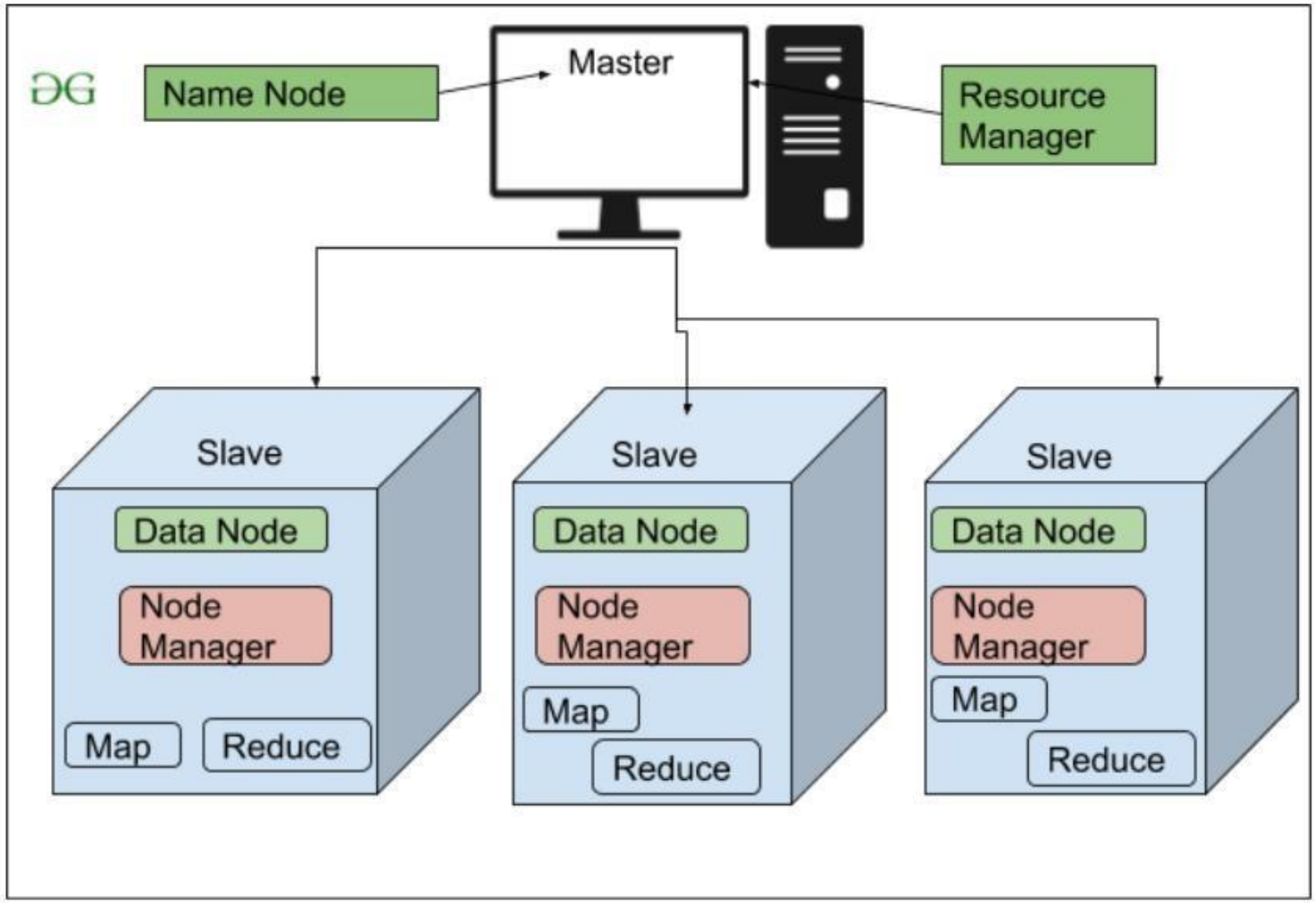
NAMENODE

- **NameNode** works as a Master in a Hadoop cluster that guides the **Datanode** (Slaves).
- Namenode is mainly used for storing the Metadata i.e. the data about the data.
- Meta Data can be the transaction logs that keep track of the user's activity in a Hadoop cluster.
- Meta Data can also be the name of the file, size, and the information about the location (Block number, Block ids) of Datanode that Namenode stores to find the closest DataNode for Faster Communication.
- Namenode instructs the DataNodes with the operation like delete, create, Replicate, etc.

DATANODE

- DataNodes works as a Slave. DataNodes are mainly utilized for storing the data in a Hadoop cluster, the number of DataNodes can be from 1 to 500 or even more than that.
- The more number of DataNode, the Hadoop cluster will be able to store more data.
- So it is advised that the DataNode should have High storing capacity to store a large number of file blocks.

HIGH LEVEL ARCHITECTURE OF HADOOP



YARN(YET ANOTHER RESOURCE NEGOTIATOR)

- **YARN** is a Framework on which MapReduce works. **YARN** performs 2 operations that are:
  1. **Job Scheduling:** The Purpose of Job schedular is to divide a big task into small jobs so that each job can be assigned to various slaves in a Hadoop cluster and Processing can be Maximized. Job Scheduler also keeps track of which job is important, which job has more priority, dependencies between the jobs and all the other information like job timing, etc.
  2. **Resource Management:** The use of Resource Manager is to manage all the resources that are made available for running a Hadoop cluster.

HADOOP COMMON OR COMMON UTILITIES

- **Hadoop Common** or **Common Utilities** are nothing but our java library and java files or we can say the java scripts that we need for all the other components present in a Hadoop cluster.
- These utilities are used by **HDFS**, **YARN**, and **MapReduce** for running the cluster.
- **Hadoop Common** verify that Hardware failure in a Hadoop cluster is common so it needs to be solved automatically in software by Hadoop Framework.

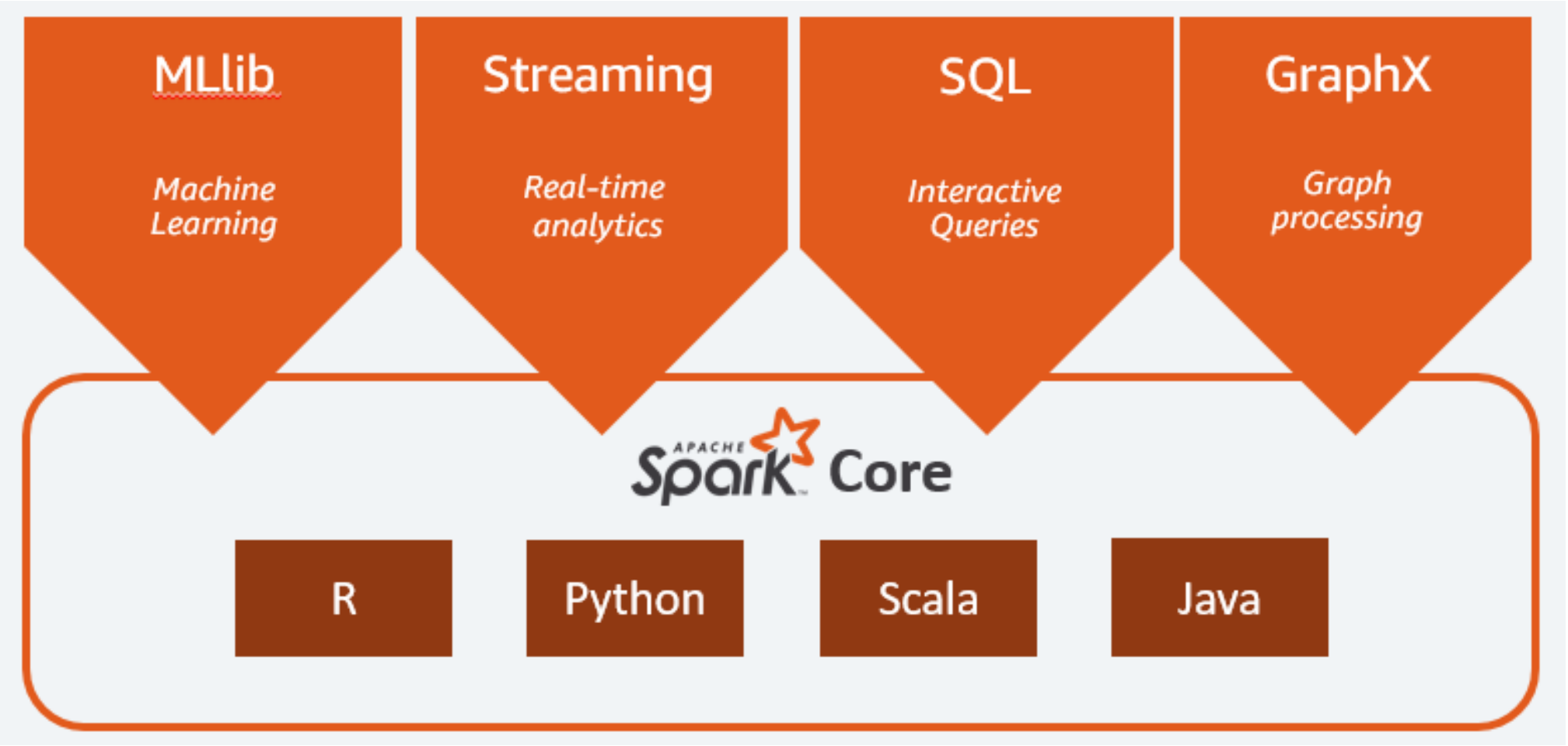
Apache Spark for Big Data Analytics

HADOOP SPARK-ED UP

- Despite the huge popularity of Hadoop in both academic and industrial settings, many users complain that Hadoop is generally quite slow and some computationally demanding data processing operations can take hours to complete.
- **Spark**, which makes use of, and is deployed on top of, the existing HDFS, has been designed to excel in iterative calculations in order to fine-tune Hadoop's in-memory performance by up to 100 times faster, and about 10 times faster when run on disk.

HADOOP SPARK-ED UP

- **Apache Spark** is an open-source, distributed processing system used for big data workloads.
- It utilizes in-memory caching, and optimized query execution for fast analytic queries against data of any size.
- It provides development APIs in Java, Scala, Python and R, and supports code reuse across multiple workloads—batch processing, interactive queries, real-time analytics, machine learning, and graph processing.
- You'll find it used by organizations from any industry, including at FINRA, Yelp, Zillow, DataXu, Urban Institute, and CrowdStrike.

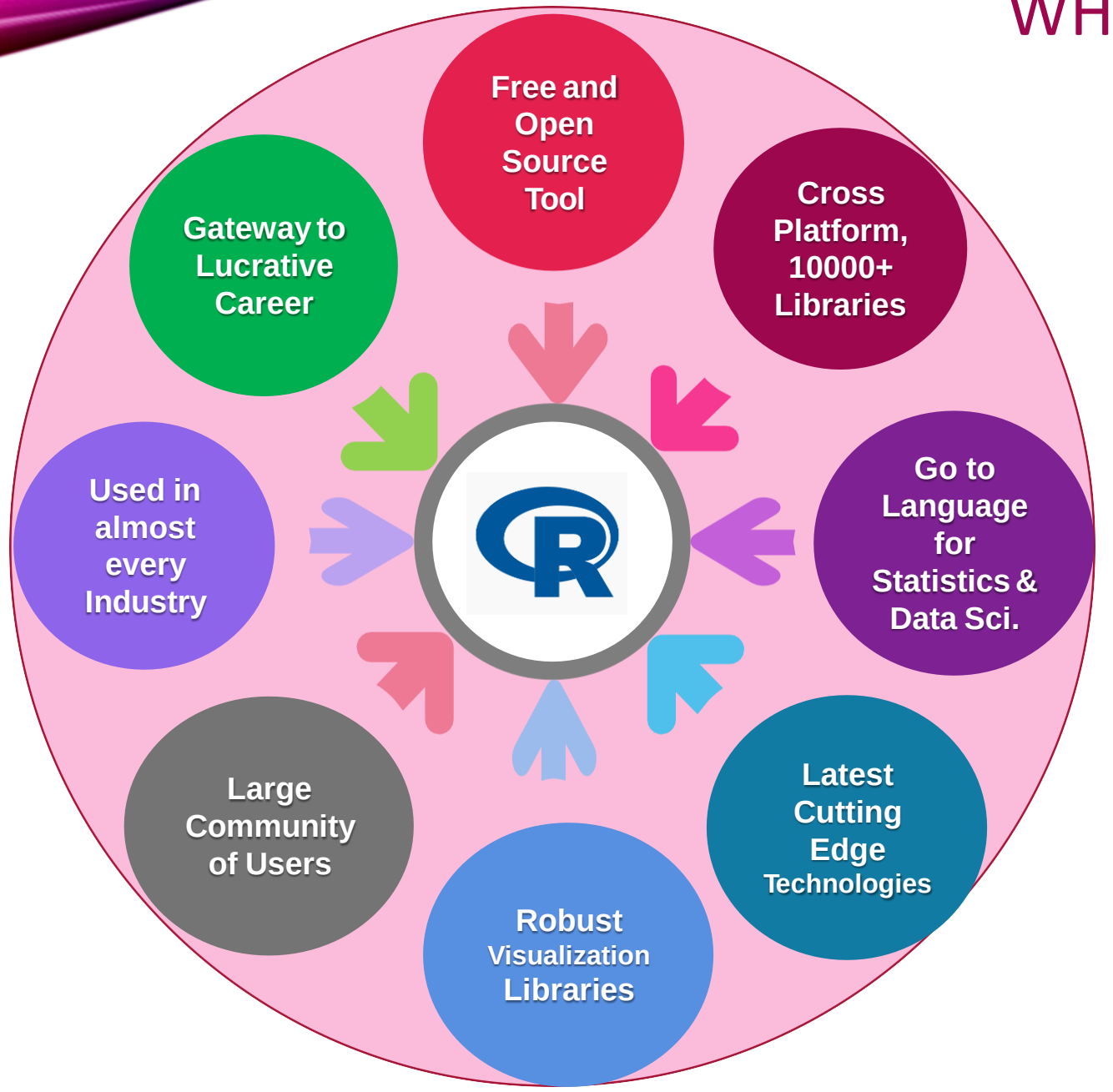

MLlib
Machine Learning
Streaming
Real-time analytics
SQL
Interactive Queries
GraphX
Graph processing
Spark Core
R
Python
Scala
Java

HADOOP SPARK-ED UP

- Spark comes with its own small but growing ecosystem of additional tools and applications that support large-scale processing of structured data
  - By implementing SQL queries into Spark programs (through **Spark SQL**),
  - Enabling fault-tolerant operations on streaming data (**Spark Streaming**),
  - Allowing users to perform sophisticated machine-learning models (**MLlib**), and
  - Carrying out *out-of-the-box* parallel community detection algorithms such as PageRank, label propagation, and many others on graphs and collections through the GraphX module.
- Owing to the open source, community-run Apache status of the project, Spark has already attracted an enormous interest from people involved in Big Data analysis and machine learning.

R–The Unsung Big Data Hero

WHY TO USE R?


Free and Open Source Tool
Cross Platform, 10000+ Libraries
Go to Language for Statistics & Data Sci.
Latest Cutting Edge Technologies
Robust Visualization Libraries
Large Community of Users
Used in almost every Industry
Gateway to Lucrative Career

TOP COMPANIES USING R FOR ANALYTICS

HOW COMPANIES ARE USING R?

Facebook

Facebook uses R to update its Status and Social Networking Graph

Twitter

Twitter Uses R to monitor User Experience

New York Times

R is used by New York Times to create Infographics

Google

Google uses R to calculate the ROI of the advertising campaigns

Getting Started with R Studio

R-STUDIO IDE

- **RStudio** is an integrated development environment (IDE) for R. It includes a console, syntax-highlighting editor that supports direct code execution, as well as tools for plotting, history, debugging and workspace management
- **R-Console:** Using R console, analysts can write R commands and execute them. The codes can be written using R Script.
- **R-Script:** R script is the interface where analysts can write codes. To run the codes, press Ctrl+ Enter, or use the “Run” button on top of R Script.
- **REnvironment:** R environment is the space which display the variables, vectors and functions used in your code.
- **Graphical Output:** This is the space to display the graphical output

RStudio

File Edit Code View Plots Session Build Debug Profile Tools Help

Go to file/function | Addins | Project: (None)

project.R | tail(WHR, 10) | head(WHR, 10) | CHR | WHR

Source on Save | Run | Source

```
55 # Let us plot graph of mean data of all components
56 ggplot(CHR, aes(x=Group.1, y=Happiness.Score, fill=Group.1)) +
57   geom_bar(stat="identity") + ggtitle("Happiness Score of Each Continenet") +
58   ylab("Happiness Score") + xlab("Continent")
59
60
61
62
63
64
65
```

58:46 (Top Level) R Script

R Script

Console | Terminal | Jobs

D:/SkyfiLabs/Data Analytics Using R/

```
> # Let us plot graph of mean data of all components
> ggplot(CHR, aes(x=Group.1, y=Happiness.Score, fill=Group.1)) +
+   geom_bar(stat="identity") + ggtitle("Happiness Score of Each Continenet") +
+   ylab("Happiness Score") + xlab("Continent")
>
```

R Console

Environment | History | Connections

Import Dataset | List

Global Environment

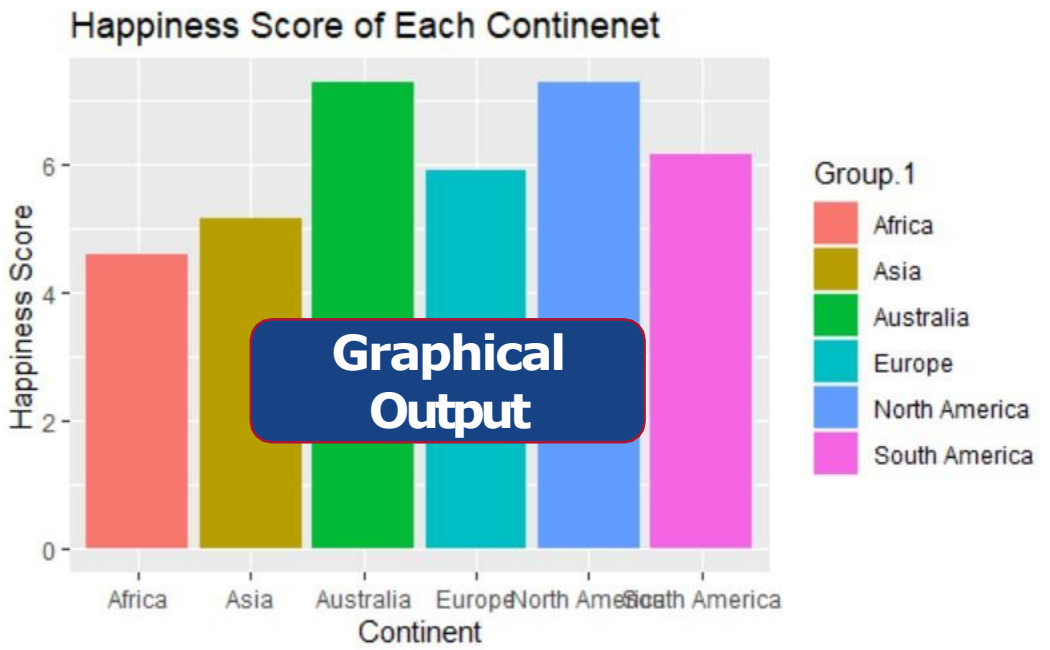
Data

CHR	6 obs. of 9 variables
WHR	158 obs. of 12 variables

R Environment

Files | Plots | Packages | Help | Viewer

Zoom | Export | Publish



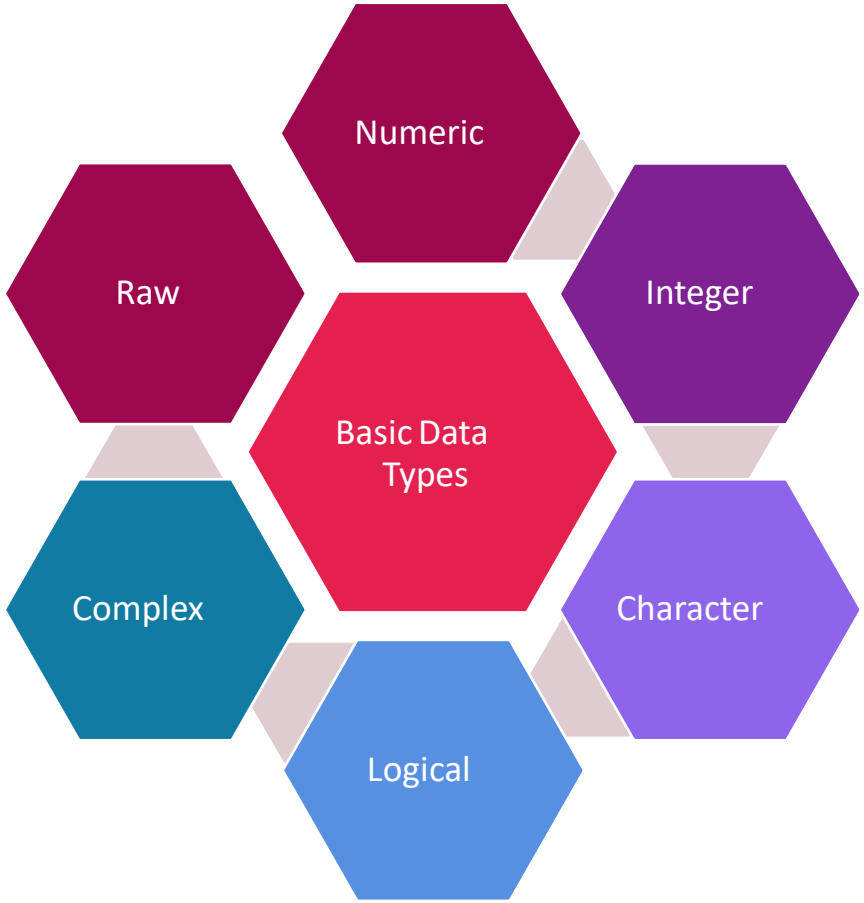
Graphical Output

Type here to search

ENG 12:12 18-06-2020

RData Types

BASIC DATA TYPES



Numeric

All kinds of numerical data except Integer Ex: a <- 3.8

Integer

Whole Numbers, Non Decimal Ex: b <- 8L

Character

Character data in double quotes Ex: c <- "Hello 123"

Logical

True or False Ex: e <- TRUE or f <- FALSE

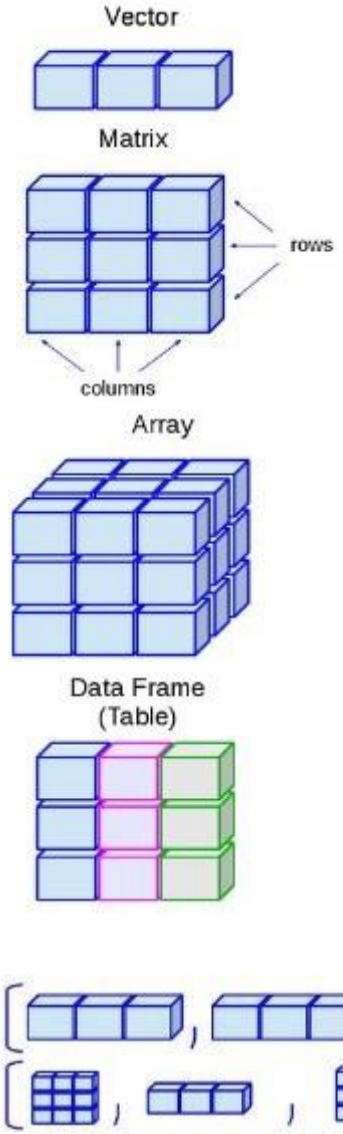
Complex

Complex Numbers Ex: g <- 5 + 4i

Raw

Raw format Ex: h <- charToRaw("Hello")
"Hello" is stored as 48 65 6c 6c 6f

DATA TYPES



Vector

Ex: apple <-c('red', 'green', 'yellow')

Matrix

Ex: M =matrix(c('a','a','b','c','b','a'), nrow =2, ncol =3, byrow =TRUE)

Array

Ex: a <-array(c('green','yellow'),dim =c(3,3,2))

Data Frame

Ex: df <-data.frame(height =c(152, 171.5, 165), weight =c(81,93, 78))

List

Ex: l <-list(apple, M, a, df)

Working with Matrices

MATRIX OPERATIONS

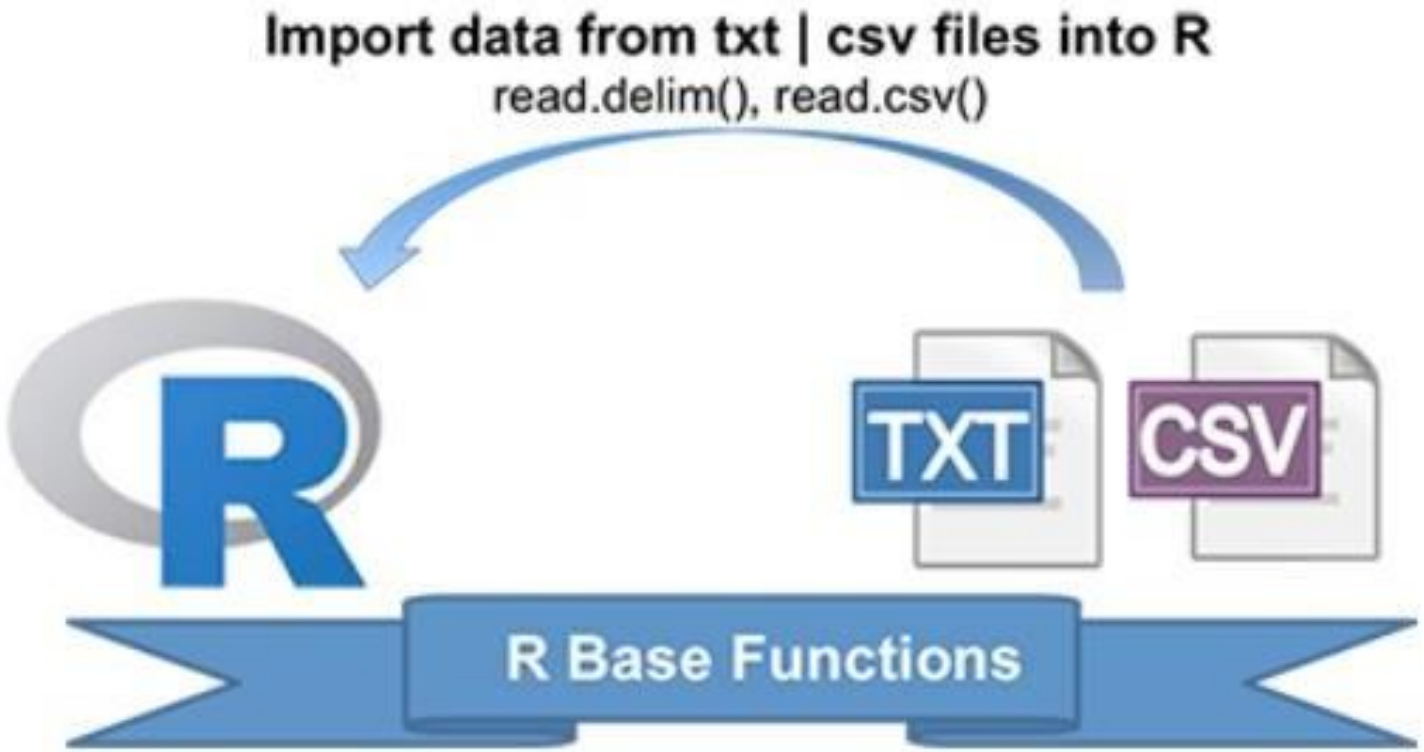
Function/Operation	Description	Syntax
matrix	To create a matrix	A <- matrix(1:6,3,2)
is.matrix	Is Something a Matrix	is.matrix(A)
nrow	Returns number of rows	nrow(A)
ncol	Returns number of columns	ncol(A)
dim	Returns dimension of matrix	dim(A)
+	Matrix Addition	A + B where dim(A) = dim(B)
-	Matrix Subtraction	A - B where dim(A) = dim(B)
*****	Scalar Multiplication	c * A where c is scalar
%*%	Matrix Multiplication	A * B where ncol(A) = nrow(B)
t	Transpose of a Matrix	t(A)
solve	Inverse of a matrix	solve(A)
det	Determinant of a Matrix	det(A)

MATRIX OPERATIONS

Function/Operation	Description	Syntax
colSum	Column wise sum of elements	colSum(A)
rowSum	Row wise sum of elements	rowSum(A)
sum	Sum of all elements	sum(A)
colMean	Column wise avearge of elements	colMean(A)
rowMean	Row wise average of elements	rowMean(A)
mean	Average of all elements	mean(A)
cbind	Concatenation of columns	cbind(A,B)
rbind	Concatenation of rows	rbind(A,B)

IMPORT/EXPORT DATA FILES

IMPORTING DATA FILES



IMPORT FILES INTO DATA FRAMES

Read the specified csv file

```
my_data <- read.csv("mtcars.csv")
```

Read the chosen csv file

```
my_data <- read.csv(file.choose())
```

Read the specified text file

```
my_data <- read.delim(("mtcars.txt")
```

Read the chosen txt file

```
my_data <- read.delim(file.choose())
```

Read the specified file

```
my_data <-read.table("mtcars.txt", sep =":", header =TRUE)
```

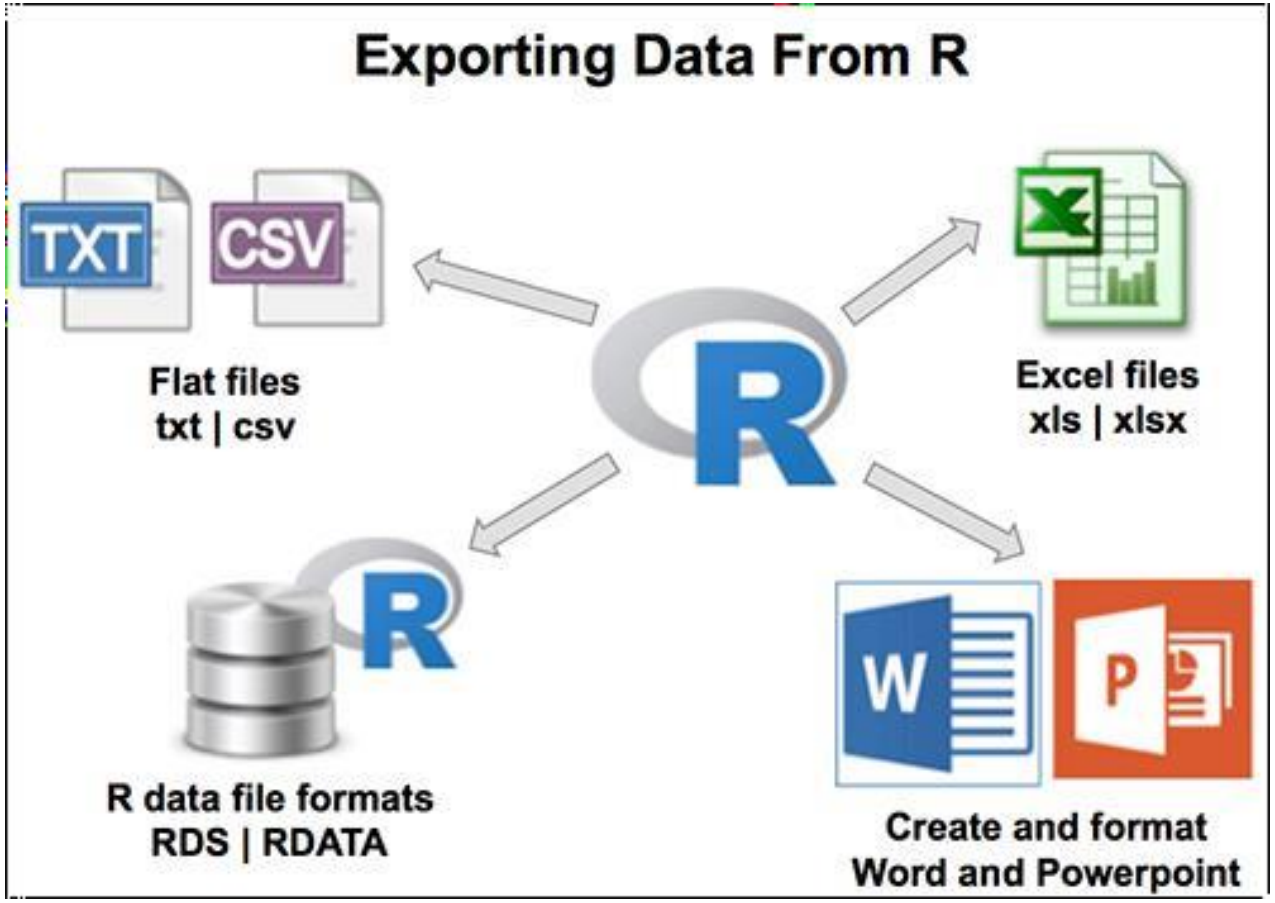
Read the file from Internet

```
my_data <-read.delim("http://www.sthda.com/upload/boxplot_format.txt")
```

Read the Excel file

```
library("readxl")
my_data <-read_excel("my_file.xlsx", sheet =2)
```

EXPORT DATA FROM R TO FILES



EXPORT DATA FROM R TO FILES

Export to csv file

```
write.csv(my_data,"mtcars_modified.csv")
```

Export to the text file

```
write.table(my_txt_data, file ="mtcars_mod.txt", sep ="\t")
```

Export to Excel file first sheet

```
write.xlsx(my_data, file ="mtcars_mod.xlsx",sheetName ="First", append =FALSE)
```

Export to Excel file second sheet

```
write.xlsx(my_data, file ="mtcars_mod.xlsx",sheetName ="Second", append =TRUE)
```

Save a Single RObject

```
saveRDS(my_data, "mtcars.rds") #Restore using d <-readRDS("mtcars.rds")
```

Save multiple RObject

```
save(data1, data2, file ="data.RData") #Load again using load("data.RData")
```

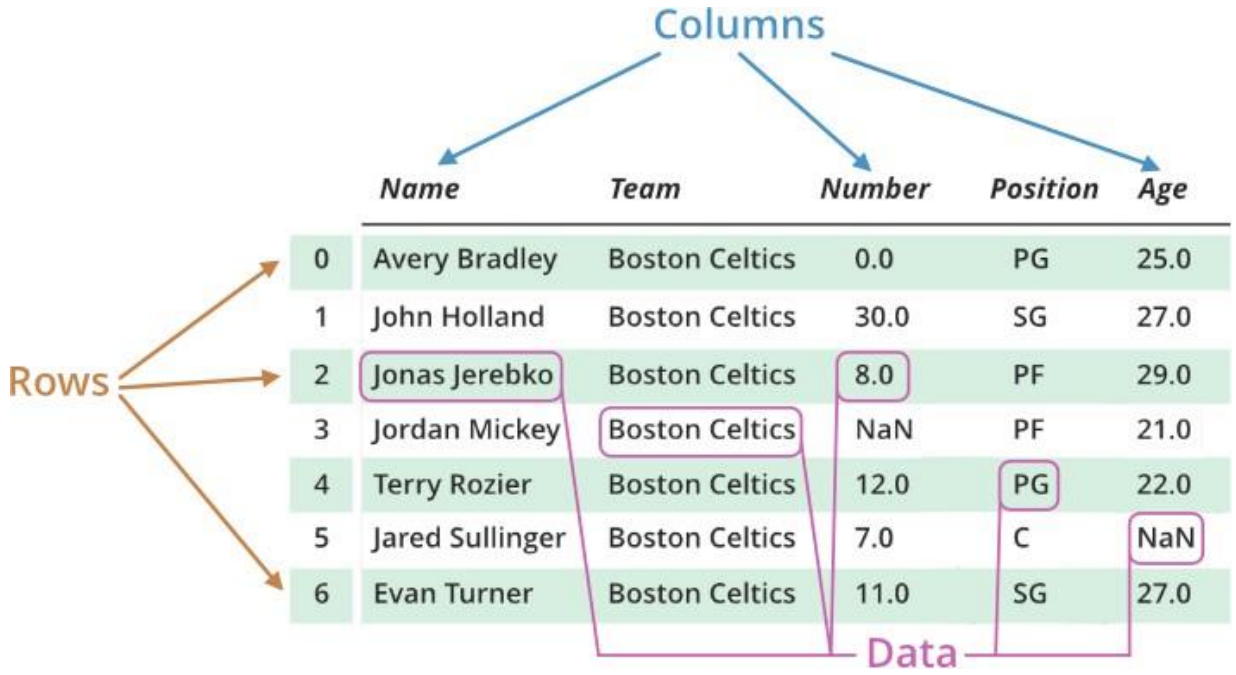
Save Entire Workspace

```
save.image(file ="my_ws.RData") #Load again using load(" my_ws.RData ")
```

Working with Data Frames

DATA FRAME

- **Data Frame** is used for storing data tables
- It is a combination of various **vectors of equal length**
- Looks like an **spreadsheet** with many **rows** and **column**
- Here each **column** is a **variable** and each **row** is an **observation**



	Name	Team	Number	Position	Age
0	Avery Bradley	Boston Celtics	0.0	PG	25.0
1	John Holland	Boston Celtics	30.0	SG	27.0
2	Jonas Jerebko	Boston Celtics	8.0	PF	29.0
3	Jordan Mickey	Boston Celtics	NaN	PF	21.0
4	Terry Rozier	Boston Celtics	12.0	PG	22.0
5	Jared Sullinger	Boston Celtics	7.0	C	NaN
6	Evan Turner	Boston Celtics	11.0	SG	27.0

DATA FRAME RELATED FUNCTIONS

Function	Description	Syntax
data.frame	To create data frames	df <- data.frame(x, y)
ncol	Returns number of columns	ncol(df)
nrow	Returns number of rows	nrow(df)
dim	Returns dimension of dataframe	dim(df)
names	Returns names of the columns	names(df)
rownames	Returns names of the rows	rownames(df)
view	Opens window to view the entire data set	view(df)
head	Returns first few rows	head(df)
tail	Returns last few rows	tail(df)
str	Shows the structure of the data frame	str(df)
summary	Provides summary statistics on the columns of the data frame	summary(df)

DATA FRAME OPERATIONS

Operation	Description
df[1,2]	Return element at 1st row and 2nd Column of df
df[1:2,]	Return 1st and 2nd rows of df
df[,1:2]	Return 1st and 2nd Column of df
df[1:2]	Return 1st and 2nd Column of df
df[1:2,1:2]	Return elements corresponding to 1st and 2nd rows and 1st and 2nd Column
df[,]	Returns all rows and columns
df[,c("a","b")]	Return Columns "a" and "b"
subset	Extract subset based on conditions e.g. subset(df, a = "abc")
df[[2]][2]="abc"	Assign "abc" to element at 2nd row and 2nd Column
edit(df)	Edit a data frame
df[-3,-1]	Return elements except of 3rd row and 1st column

STEPS IN DATA ANALYTICS

1. **Defining the Questions: What to Measure ? How to Measure ?**
2. **Collection of Relevant Data**
3. **Manipulation of Data**
4. **Visualization of Data**
5. **Interpretation of Results**

Data Manipulation Using dplyr

DATA MANIPULATION

1. **Data Manipulation** is process of converting data into simpler and easier to read format

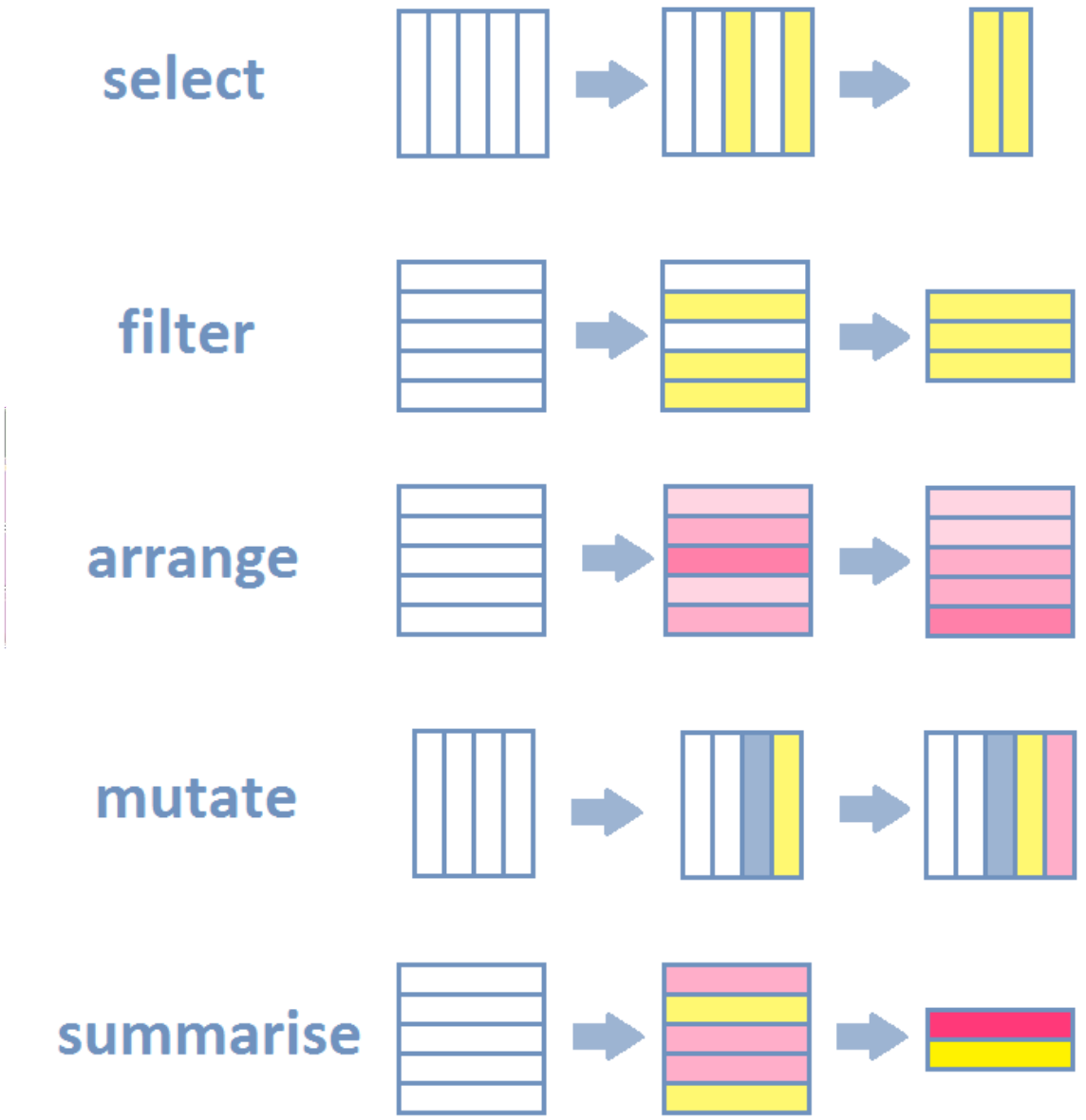
2. **Data Manipulation** is Used to **organize data**

3. **Data Manipulation** is done by many packages in Re.g. **dplyr**, **tidyverse** etc

4. **dplyr** package is referred as **Grammar of Data Manipulation**

5. **dplyr** simplifies **data manipulation** by providing **various functions to deal with data**

DPLYR FUNCTIONS



DPLYR FUNCTIONS

- **dplyr** consists of different cases which provide individual functionality
  - Extract/Manipulate Rows
  - Extract/Manipulate Columns
  - Summarize Rows
  - Group Rows

EXTRACT/MANIPULATE ROWS

Function	Description
filter()	Extract rows satisfying criteria
slice()	Selects rows by position
distinct()	Remove rows with duplicate values
sample_n()	Randomly select n rows
sample_frac()	Randomly select fraction of rows
arrange()	Order rows by values of column(s) (low to high), use desc() for high to low
add _row()	Add one or more rows to a table. Value of each column should be declared

FILTER

- Extract rows satisfying specified criteria
- Select storm whose name is in the vector [Anna, Alberto]

```
s <- filter(storms, storm %in% c(Anna, Alberto))
```

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Anna	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	10010	1996-06-13

s

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Anna	40	1013	1997-07-01

SLICE

- Select rows by position
- Select rows 2 to 4 from storms

```
s <- slice(storms, 2:4)
```

`storms`

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Anna	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	10010	1996-06-13

slice

storm	wind	pressure	date
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Anna	40	1013	1997-07-01

DISTINCT

- Remove duplicate values of the specified column(s)
- Remove duplicate values of wind

```
s <- distinct(storms, wind)
```

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	45	1005	1995-06-04
Anna	50	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	10010	1996-06-13

s

wind
110
45
50

RANDOM_N

- Randomly select n rows
- Randomly select 2 rows

```
s <- random_n(storms, 2)
```

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Anna	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	10010	1996-06-13

random_n

s

storm	wind	pressure	date
Alex	45	1009	1998-07-30
Arlene	50	1010	1999-06-13

RANDOM_FRAC

- Randomly select fraction of rows
- Randomly select 20% of rows from storms

```
s <- random_frac(storms, 0.2)
```

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Anna	40	1013	1997-07-01
Arlene	50	1010	1999-06-13

s

storm	wind	pressure	date
Anna	40	1013	1997-07-01

ARRANGE

- Order rows by values of the specified column(s) (low to high), use desc() for high to low
- Arrange storms based on winds

```
s <- arrange(storms, wind)
```

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Anna	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	10010	1996-06-13

s

storm	wind	pressure	date
Anna	40	1013	1997-07-01
Alex	45	1009	1998-07-30
Arthur	45	10010	1996-06-13
Arlene	50	1010	1999-06-13
Allison	65	1005	1995-06-04
Alberto	110	1007	2000-08-12

ADD_ROW

- Add row to a table. Value of each column should be specified
- Add a new row with values storm="Arthur", wind=45, pressure=10010, date='1996-06-13

```
s <- add_row(storms, storm="Arthur",wind=45,pressure=10010,date='1996-06-13)
```

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Anna	40	1013	1997-07-01
Arlene	50	1010	1999-06-13

add_row

s

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Anna	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	10010	1996-06-13

EXTRACT/MANIPULATE COLUMNS/VARIABLES

Function	Description
select()	Extract columns as a table
rename()	Renames existing columns
mutate()	Computes new column as a function of one or more columns
transmute()	Computes new columns and only display's new column
add_column()	Adds new column (requires tibble library)

SELECT

- Extract columns as a table
- Select columns storm and date from storms

```
s <- select(storms, storm, date)
```

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Anna	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	10010	1996-06-13

s

storm	date
Alberto	2000-08-12
Alex	1998-07-30
Allison	1995-06-04
Anna	1997-07-01
Arlene	1999-06-13
Arthur	1996-06-13

RENAME

- renames existing columns
- Rename the column storm to sname

```
s <- rename(storms, sname=storm)
```

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Anna	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	10010	1996-06-13

s

sname	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Anna	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	10010	1996-06-13

MUTATE

- Creates new column as a function of one or more columns
- Create column wind_km to store the wind speed in km / hour

```
s <- mutate(storms, wind_km=wind * 1.60934)
```

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Anna	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	10010	1996-06-13

mutate

s

sname	wind	pressure	date	Wind_km
Alberto	110	1007	2000-08-12	177.0274
Alex	45	1009	1998-07-30	72.4203
Allison	65	1005	1995-06-04	104.6071
Anna	40	1013	1997-07-01	64.3736
Arlene	50	1010	1999-06-13	80.4670
Arthur	45	10010	1996-06-13	72.4203

TRANSMUTE

- Computes only new column as a function of one or more columns
- Compute and return only column wind_km to store wind speed in km / hour

```
s <- transmute(storms, wind_km=wind * 1.60934)
```

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Anna	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	10010	1996-06-13

transmute

s

Wind_km
177.0274
72.4203
104.6071
64.3736
80.4670
72.4203

ADD_COLUMN

- Adds new column (requires tibble library)
- Add column foo with values 1:6

```
s <- add_column(storms, foo=1:6)
```

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Anna	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	10010	1996-06-13

Add_column

s

sname	wind	pressure	date	foo
Alberto	110	1007	2000-08-12	1
Alex	45	1009	1998-07-30	2
Allison	65	1005	1995-06-04	3
Anna	40	1013	1997-07-01	4
Arlene	50	1010	1999-06-13	5
Arthur	45	10010	1996-06-13	6

SUMMARISE AND GROUP

Function	Description
summarise()	Summarize function takes vectors (data from datset) as inputs and return one value (e.g. mean, sum, min, max etc.)
group_by()	Returns grouped copy of a table based on the specified column(s)
ungroup()	Returns ungrouped copy of table

SUMMARISE

- summarize calculates the aggregates and return one value for each aggregate specified
- Get average, min and max of wind from storms

```
s <- summarise(storms, avg_wind=mean(wind), min_wind=min(wind), max_wind=max(wind))
```

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Anna	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	10010	1996-06-13

s

avg_wind	min_wind	max_wind
59.16667	40	110

GROUP AND SUMMARISE

- Group by returns grouped copy of a table based on the specified column(s), summarize return one value for each group for each aggregate
- Get average of wind and median of pressure from storms grouped by country

```
s1 <- group_by(storms, country)
s <- summarise(s1, avg_wind=mean(wind), med_pressure=median(pressure))
```

storms

storm	wind	pressure	date	country
Alberto	110	1007	2000-08-12	USA
Alex	45	1009	1998-07-30	USA
Allison	65	1005	1995-06-04	USA
Anna	40	1013	1997-07-01	Canada
Arlene	50	1010	1999-06-13	Canada
Arthur	45	10010	1996-06-13	Canada

group by summarise

s

country	avg_wind	med_pressure
USA	45	1013
Canada	73.3	1007

GROUP AND SUMMARISE USING PIPE

- Group by returns grouped copy of a table based on the specified column(s), summarize return one value for each group for each aggregate
- Get average of wind and median of pressure from storms grouped by country

```
s <- storms %>% group_by(country) %>% summarise(mean(wind), median(pressure))
```

storms

storm	wind	pressure	date	country
Alberto	110	1007	2000-08-12	USA
Alex	45	1009	1998-07-30	USA
Allison	65	1005	1995-06-04	USA
Anna	40	1013	1997-07-01	Canada
Arlene	50	1010	1999-06-13	Canada
Arthur	45	10010	1996-06-13	Canada

s

country	avg_wind	med_pressure
USA	45	1013
Canada	73.3	1007

UNGROUP AND SUMMARISE

- Ungroup returns ungrouped copy of a grouped table, summarize return one value for each aggregate
- ungroup the table and get average of wind and median of pressure from storms after

```
s1 <- ungroup(s1)
s <- summarise(s1, avg_wind=mean(wind), med_pressure=median(pressure))
```

storms

storm	wind	pressure	date	country
Alberto	110	1007	2000-08-12	USA
Alex	45	1009	1998-07-30	USA
Allison	65	1005	1995-06-04	USA
Anna	40	1013	1997-07-01	Canada
Arlene	50	1010	1999-06-13	Canada
Arthur	45	10010	1996-06-13	Canada

s

avg_wind	med_pressure
59.2	1010

UNGROUP AND SUMMARISE USING PIPE

- Ungroup returns ungrouped copy of a grouped table, summarize return one value for each aggregate
- ungroup the table and get average of wind and median of pressure from storms after

```
s <- ungroup(s1) %>% summarise(avg_wind=mean(wind), med_pressure=median(pressure))
```

storms

storm	wind	pressure	date	country
Alberto	110	1007	2000-08-12	USA
Alex	45	1009	1998-07-30	USA
Allison	65	1005	1995-06-04	USA
Anna	40	1013	1997-07-01	Canada
Arlene	50	1010	1999-06-13	Canada
Arthur	45	10010	1996-06-13	Canada

s

avg_wind	med_pressure
59.2	1010

THANK YOU!

- The answer is not very straightforward, and the exact number is not set in stone. To give an approximate estimate we first need to differentiate between simply storing the data, and processing or analyzing the data.
- If your goal was to preserve 1,000 YouTube videos on a hard drive, it most likely wouldn't be a very demanding task. Data storage is relatively inexpensive nowadays, and new rapidly emerging technologies bring its prices down almost as you read this book.
- It is amazing just to think that only 20 years ago, $300 would merely buy you a 2GB hard drive for your personal computer, but 10 years later the same amount would suffice to purchase a hard drive with a 200 times greater capacity.

- Obviously, you can go for cheaper and more traditional hard disks in order to store your 1,000 YouTube videos; there is a large selection of available products to suit every budget.
- It would be a slightly different story, however, if you were tasked to process all those 1,000 videos, for example by creating shorter versions of each or adding subtitles.
- Even worse if you had to analyze the actual footage of each movie, and quantify, for example, how many seconds per video red colored objects of the size of at least 20×20 pixels are shown.
- Such tasks do not only require considerable storage capacities, but also, and primarily, the processing power of the computing facilities at your disposal. You could possibly still process and analyze each video, one by one, using a top-of-the-range personal computer, but 1,000 video files would definitely exceed its capabilities and most likely your limits of patience too.
- In order to speed up the processing of such tasks, you would need to quickly find some extra cash to invest into further hardware upgrades, but then again this would not solve the issue.

- Currently, personal computers are only **vertically scalable** to a very limited extent.
- As long as your task does not involve heavy data processing, and is simply restricted to file storage, an individual machine may suffice.
- However, at this point, apart from large enough hard drives, we would need to make sure we have a sufficient amount of **Random Access Memory** (**RAM**), and fast, heavy-duty processors on compatible motherboards installed in our units.
- Upgrades of individual components, in a single machine, may be costly, short-lived due to rapidly advancing new technologies, and unlikely to bring a real change to complex data crunching tasks. Strictly speaking, this is not the most efficient and flexible approach for Big Data analytics to say the least.

- We would most probably have to process the data on a cluster of machines working in parallel. This task would require our system to be **horizontally scalable,** meaning that we would be capable of easily increasing (or decreasing) the number of units (nodes) connected in our cluster as we wish.
- A clear advantage of horizontal scalability over vertical scalability is that we would simply be able to use as many nodes working in parallel as required by our task, and we would not be bothered too much with the individual configuration of each and every machine in our cluster.

- The size of data is not, however, the only factor that makes the data Big. Although the simplified definition of Big Data, previously presented, explicitly refers to the *one byte* as a measurement of size, we should dissect the second part of the statement, in a few sentences, to have a greater understanding of what Big Data actually means.
- Data do not just come to us and *sit* in a file. Nowadays, most data change, sometimes very rapidly. Near real-time analytics of Big Data currently gives huge headaches to in-house data science departments, even at international large financial institutions or energy companies.
- In fact stock-market data, or sensor data, are pretty good, but still quite extreme examples of high-dimensional data that are stored and analyzed at milliseconds intervals.
- Several seconds of delay in producing data analyses, on near real-time information, may cost investors quite substantial amounts, and result in losses in their portfolio value, so the speed of processing fast-moving data is definitely a considerable issue at the moment.

- Moreover, data are now more complex than ever before.
- Information may be scrapped off the websites as unstructured text, JSON format, HTML files, through service APIs, and so on.
- Excel spreadsheets and traditional file formats such as **Comma-Separated Values** (CSV) or tab-delimited files that represent structured data are not in the majority any more.
- It is also very limiting to think of data as of only numeric or textual types.
- There is an enormous variety of available formats that store, for instance, audio and visual information, graphics, sensors, and signals, 3D rendering and imaging files, or data collected and compiled using highly specialized scientific programs or analytical software packages such as **Stata** or **Statistical Package for the Social Sciences** (**SPSS**) to name just a few
- A large list of most available formats is accessible through Wikipedia at

  https://en.wikipedia.org/wiki/Listoffileformats

- The size of data, the speed of their inputs/outputs and the differing formats and types of data were in fact the original three *Vs: Volume*, *Velocity*, and *Variety*, described in the article titled *3D Data Management: Controlling Data Volume, Velocity, and Variety* published by Doug Laney back in 2001, as major conditions to treat any data as Big Data.
- Doug's famous three Vs were further extended by other data scientists to include more specific and sometimes more qualitative factors such as data *variability* (for data with periodic peaks of data flow), *complexity* (for multiple sources of related data), *veracity* (coined by IBM and denoting trustworthiness of data consistency), or *value* (for examples of insight and interpretation).
- No matter how many Vs or Cs we use to describe Big Data, it generally revolves around the limitations of the available IT infrastructure, the skills of the people dealing with large data sets and the methods applied to collect, store, and process these data.

- As we have previously concluded that Big Data may be defined differently by different entities (for example individual users, academic departments, governments, large financial companies, or technology leaders), we can now rephrase the previously referenced definition in the following general statement:
- ***Big Data any data that cause significant processing, management, analytical, and interpretational problems.***
- Also, for the purpose of this book, we will assume that such problematic data will generally start from around 4 GB to 8 GB in size, the standard capacity of RAM installed in most commercial personal computers available to individual users in the years 2014 and 2015.
- This arbitrary threshold will make more sense when we explain traditional limitations of the R language later on in this chapter, and methods of Big Data in-memory processing across several chapters in this book.

HADOOP – THE ELEPHANT IN THE ROOM

- If you have been in the Big Data industry for as little as one day, you surely must have heard the unfamiliar sounding word *Hadoop*, at least every third sentence during frequent tea break discussions with your work colleagues or fellow students.
- Named after Doug Cutting's child's favorite toy, a yellow stuffed elephant, Hadoop has been with us for nearly 11 years.
- Its origins began around the year 2002 when Doug Cutting was commissioned to lead the **Apache Nutch** project-a scalable open source search engine.
- Several months into the project, Cutting and his colleague Mike Cafarella ran into serious problems with the scaling up and robustness of their Nutch framework owing to growing storage and processingneeds.

- The solution came from none other than Google, and more precisely from a paper titled *The Google File System* authored by Ghemawat, Gobioff, and Leung, and published in the proceedings of the *19th ACM Symposium on Operating Systems Principles.*
- The article revisited the original idea of **Big Files** invented by Larry Page and Sergey Brin, and proposed a revolutionary new method of storing large files partitioned into fixed-size 64 MB chunks across many nodes of the cluster built from cheap commodity hardware.
- In order to prevent failures and improve efficiency of this setup, the file system creates copies of chunks of data, and distributs them across a number of nodes, which were in turn mapped and managed by a master server.
- Several months later, Google surprised Cutting and Cafarella with another groundbreaking research article known as *MapReduce: Simplified Data Processing on Large Clusters,* written by Dean and Ghemawat, and published in the Proceedings of the *6th Conference on Symposium on Operating Systems Design and Implementation*.
- The MapReduce framework became a kind of mortar between bricks, in the form of data distributed across numerous nodes in the file system, and the outputs of data transformations and processing tasks.

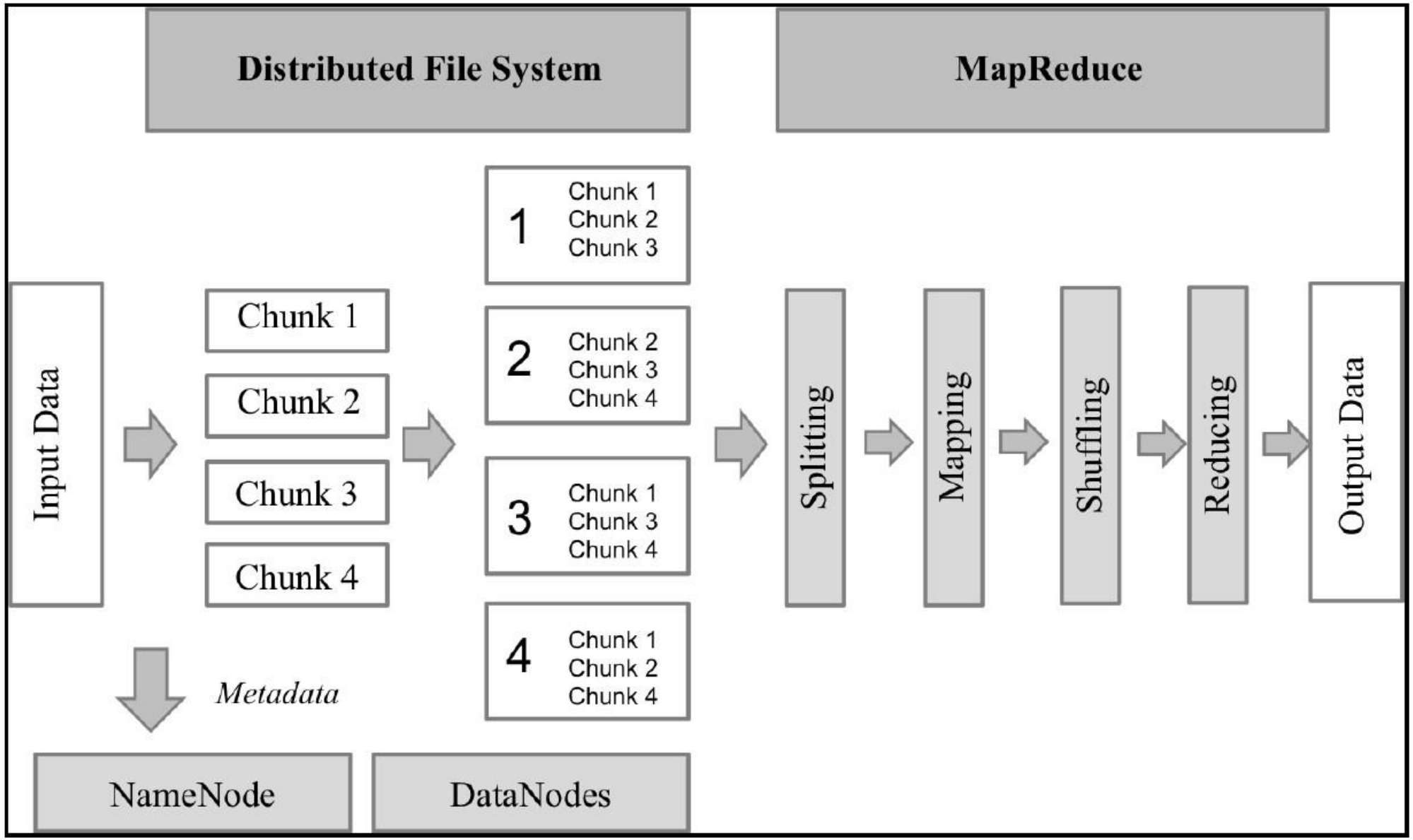
MAPREDUCE MODEL

- The MapReduce model contains three essential stages.

1. The first phase is the **Mapping** procedure, which includes indexing and sorting data into the desired structure based on the specified key-value pairs of the mapper (that is, a script doing the mapping).
2. The Shuffle stage is responsible for the redistribution of the mapper's outputs across nodes, depending on the key; that is, the outputs for one specific key are stored on the same node.
3. The Reduce stage results in producing a kind of summary output of the previously mapped and shuffled data, for example, a descriptive statistic such as the arithmetic mean for a continuous measurement by each key (for example a categorical variable).

- A simplified data processing workflow, using the MapReduce framework in Google and **Distributed File System,** is presented in the following figure:

A simplified Distributed File System architecture and stages of the MapReduce framework



- The ideas of the **Google File System** model, and the MapReduce paradigm, resonated very well with Cutting and Cafarella's plans, and they introduced both frameworks into their own research on Nutch.
- For the first time their web crawler algorithm could be run in parallel on several commodity machines with minimal supervision from a human engineer.

- In 2006, Cutting moved to Yahoo! and in 2008, Hadoop became a separate Nutch independent Apache project. Since then, it's been on a never-ending journey towards greater reliability and scalability to allow bigger and faster workloads of data to be effectively crunched by gradually increasing node numbers.
- In the meantime, Hadoop has also become available as an add-on service on leading cloud computing platforms such as Microsoft Azure, Amazon **Elastic Cloud Computing** (**EC2**), or Google Cloud Platform.
- This new, unrestricted, and flexible way of accessing shared and affordable commodity hardware, enabled numerous companies as well as individual data scientists and developers to dramatically cut their production costs and process larger than ever data in a more efficient and robust manner.

- A few Hadoop record-breaking milestones are worth mentioning at this point.
- In the well-known real-world example at the end of 2007, the New York Times was able to convert more than 4TB of images, within less than 24 hours, using a cluster built of 100 nodes on Amazon EC2, for as little as $200. A job that would have potentially taken weeks of hard labor, and a considerable amount of working man-hours, could now be achieved at a fraction of the original cost, in a significantly shorter scope of time.
- A year later 1TB of data was already sorted in 209 seconds and in 2009, Yahoo! set a new record by sorting the same amount of data in just 62 seconds.
- In 2013, Hadoop reached its best **Gray Sort Benchmark** (https://sortbenchmark.org) score so far. Using 2,100 nodes, Thomas Graves from Yahoo! was able to sort 102.5TB in 4,328 seconds, so roughly 1.42TB per minute.

- In recent years, the Hadoop and MapReduce frameworks have been extensively used by the largest technology businesses such as Facebook, Google, Yahoo!, major financial and insurance companies, research institutes, and Academia, as well as Big Data individual enthusiasts.
- A number of enterprises offering commercial distributions of Hadoop such as **Cloudera** or **Hortonworks** have spun off from the original Hadoop project, and evolved into separate entities, providing additional Big Data proprietary tools, and extending the applications and usability of Hadoop ecosystem.
- Although MapReduce and Hadoop revolutionized the way we process vast quantities of data, and hugely propagated Big Data analytics throughout the business world and individual users alike, they also received some criticism for their still present performance limitations, reliability issues, and certain programming difficulties.

DATABASES

- There are many excellent online and offline resources and publications available to readers, on both SQL-based **Relational Database Management System** (**RDBMS**)**,** and more modern non-relational and **Not Only SQL** (**NoSQL**) databases.
- We need to know several practical examples on how to store large amounts of information in such systems, carry out essential data crunching and processing of the data using known and tested R packages, and extract the outputs of these Big Data transformations from databases directly into your R session.

- In Unit-III, *R with Relational Database Management System (RDBMSs)*, we will begin with a very gentle introduction to standard and more traditional databases built on the relational model developed in 1970s by Oxford-educated English computer scientist Edgar Codd.
- At this stage, it is only important for you to know that, in the RDBMS, the data is stored in a structured way in the form of tables with fields and records.
- Depending on the specific industry that you come from, fields can be understood as either variables or columns, and records may alternatively be referred to as observations or rows of data.
- In other words, fields are the smallest units of information and records are collections of these fields.
- Fields, like variables, come with certain attributes assigned to them, for example, they may contain only numeric or string values, double or long, and so on. These attributes can be set when inputting the data into the database.

- The RDBMS have proved extremely popular and most (if not all) currently functioning enterprises that collect and store large quantities of data have some sort of relational databases in place.
- The RDBMS can be easily queried using the **Structured Query Language** (**SQL**)-an accessible and quite natural database management programming language, firstly created at IBM by Donald Chamberlin and Raymond Boyce, and later commercialized and further developed by Oracle.
- Since the original birth of the first RDBMS, they have evolved into fully supported commercial products with Oracle, IBM, and Microsoft being in control of almost 90% of the total RDBMS market share.
- In our examples of R's connectivity with RDBMS in Unit III we will employ a number of the most popular relational and open source databases available to users, including MySQL, PostgreSQL, SQLite, and MariaDB.

- However, this is not where we are going to end our journey through the exciting landscape of databases and their Big Data applications.
- Although RDBMS perform very well in the cases of heavy transactional load, and their ability to process quite complex SQL queries is one of their greatest advantages, they are not so good with (near) real-time and streaming data.
- Also they generally do not support unstructured and hierarchical data, and they are not easily horizontally scalable.
- In response to these needs, a new type of database has recently evolved, or to be more precise, it was revived from a long state of inactivity as nonrelational databases were known and in use, parallel with RDBMS, even forty years ago, but they never became as popular.

- NoSQL and non-relational databases, unlike SQL-based RDBMS, come with no predefined schema, thus allowing the users a much-needed flexibility without altering the data.
- They generally scale horizontally very well and are praised for the speed of processing, making them ideal storage solutions for (near) real-time analytics in such industries as retail, marketing, and financial services.
- They also come with their own flavors of SQL like queries; some of them, for example, the MongoDB NoSQL language, are very expressive and allow users to carry out most data transformations and manipulations as well as complex data aggregation pipelines or even database-specific MapReduce implementations.
- The rapid growth of interactive web-based services, social media, and streaming data products resulted in a large array of such purpose-specific NoSQL databases being developed.
- In Chapter 6, *R with Non-Relational and (NoSQL) Databases*, we will present several examples of Big Data analytics with R using data stored in three leading open source non-relational databases: a popular document-based NoSQL, MongoDB, and a distributed Apache Hadoop-complaint **HBase database**

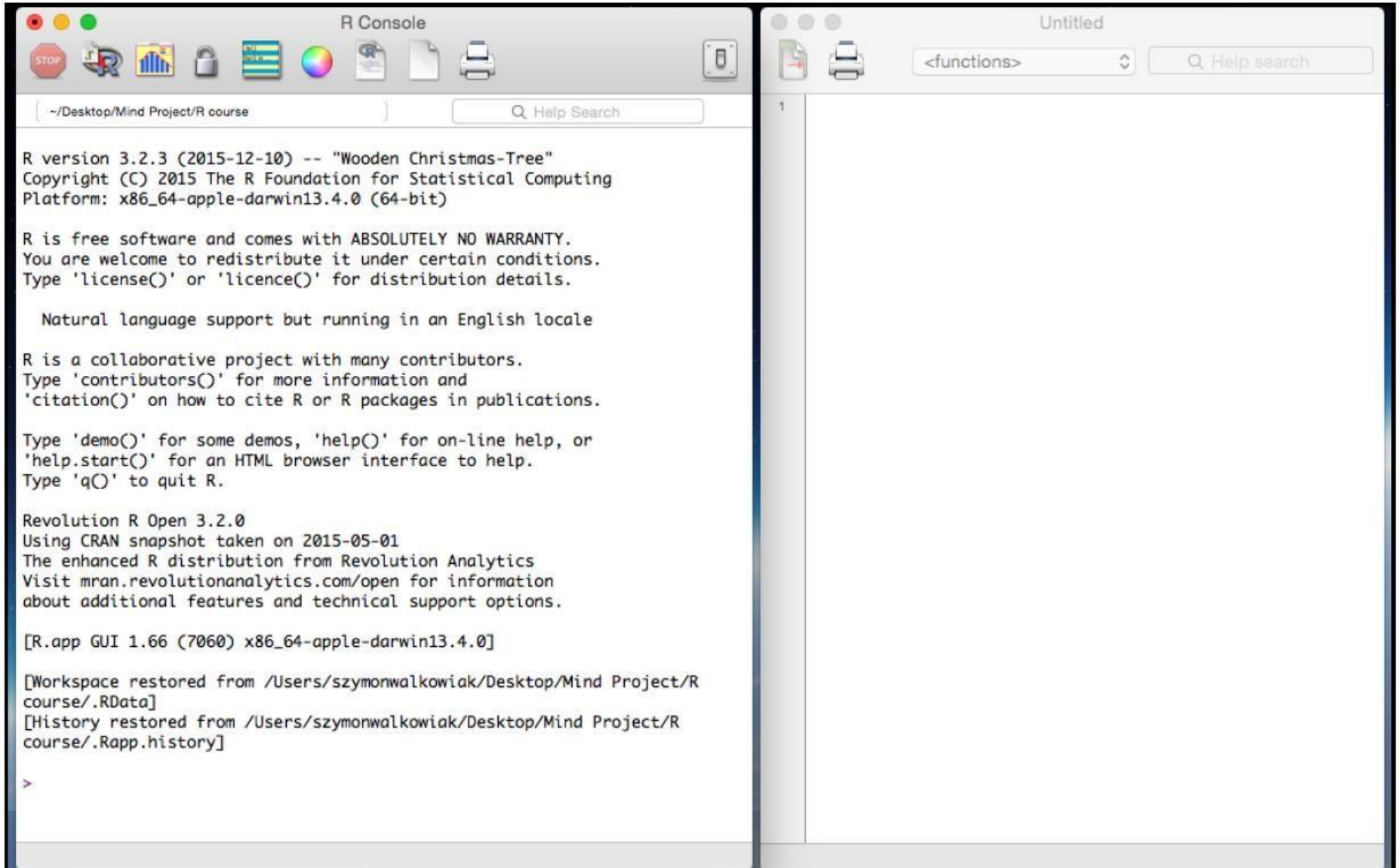
- As of the end of July 2016, there are over 240 third-party packages developed by independent Spark users available at the http://spark-packages.org/ website. A large majority of them allow further integration of Spark with other more or less common Big Data tools on the market.
- In Chapter 7, *Faster than Hadoop: Spark and R* and Chapter 8, *Machine Learning for Big Data in R* we will discuss the methods of utilizing Apache Spark in our Big Data analytics workflows using the R programming language. However before we do so, we need to familiarize ourselves with the most integral part of this book-the R language itself.

R – THE UNSUNG BIG DATA HERO

- By now you have been introduced to the notion of Big Data, its features, and its characteristics, as well as the most widely used Big Data analytics tools and frameworks such as Hadoop, HDFS, MapReduce framework, relational and non-relational databases, and Apache Spark project.
- Now the time has finally come to present the true hero and the main subject of this course-the R language.
- Currently, R comes in various shapes or forms, as there are several open source and commercial implementations of the language. The most popular are the free R GUI available to download from the **R Project CRAN** website at https://cran.r-project.org/
- **RStudio IDE** available from https://www.rstudio.com/. Owing to their popularity and functionalities, both implementations deserve a few words of introduction.

- As the R core is a multi-platform tool, RStudio is also available to users of the Windows, Mac, or Linux operating systems.
- We will be using RStudio desktop edition as well as the open source version of the RStudio Server throughout this book extensively, so please make sure you download and install the desktop free version on your machine in order to be able to follow some of the examples included in Chapter 2, *Introduction to R Programming Language and Statistical Environment* and Chapter 3, *Unleashing the Power of R from Within*.
- When we get to cloud computing (*Online Chapter*, *Pushing R Further*, https://www.packtpub.com/sites/default/files/downloads/5396645 7OSPushingRFurther.pdf) we will explain how to install and use RStudio Server on a Linux run server.

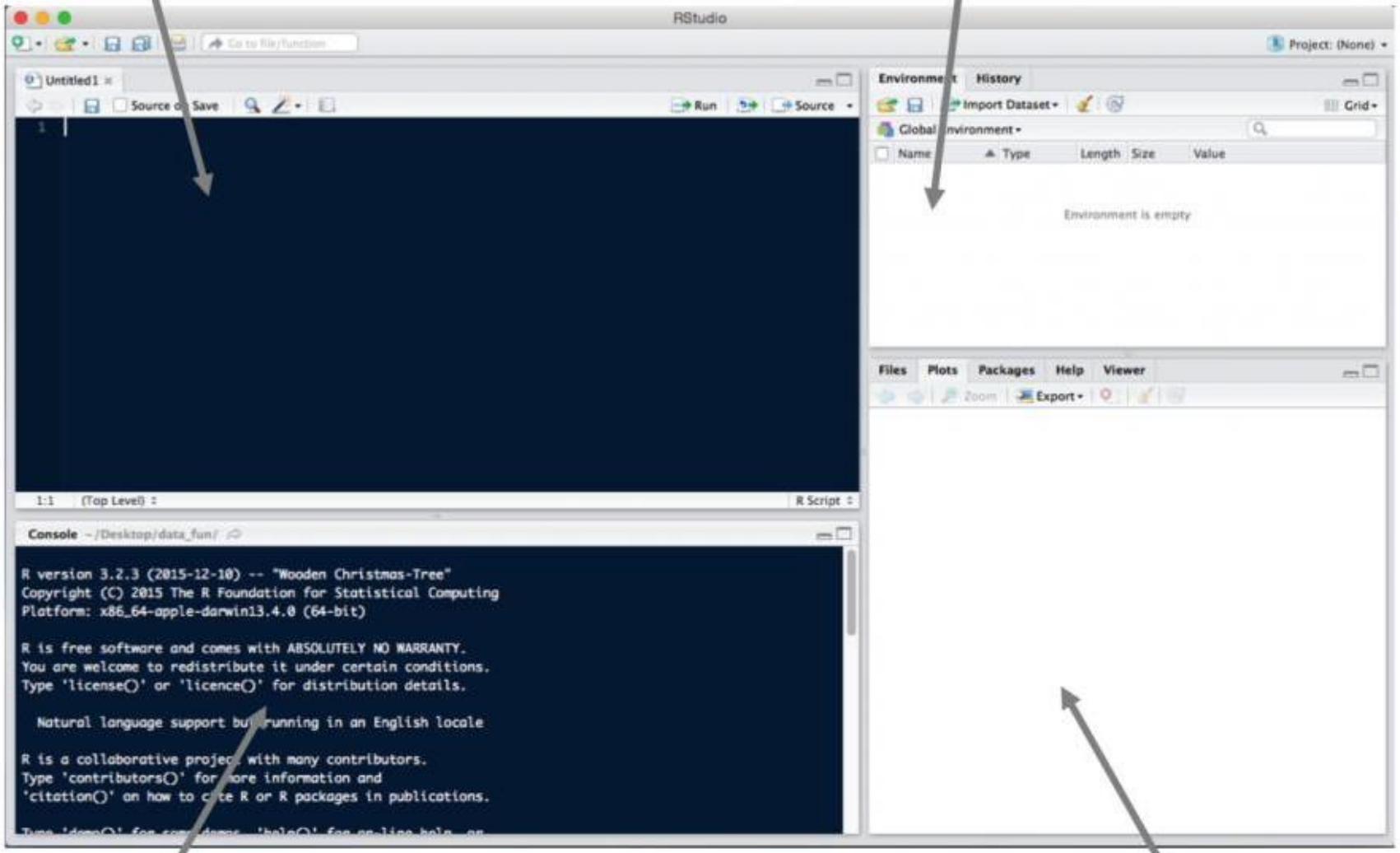
- The following are screenshots of graphical user interfaces (on Mac OS X) of both the R base installation, available from CRAN, and the RStudio IDE for desktop.
- Please note that for Windows users, GUIs may look a little different than the attached examples; however, the functionalities remain the same in most cases. RStudio Server has the same GUI as the desktop version; there are however some very minor differences in available options and settings:



R core GUI for Mac OS X with the console panel (on the left) and the code editor (on the right)

Code editor

Environment explorer and history panel



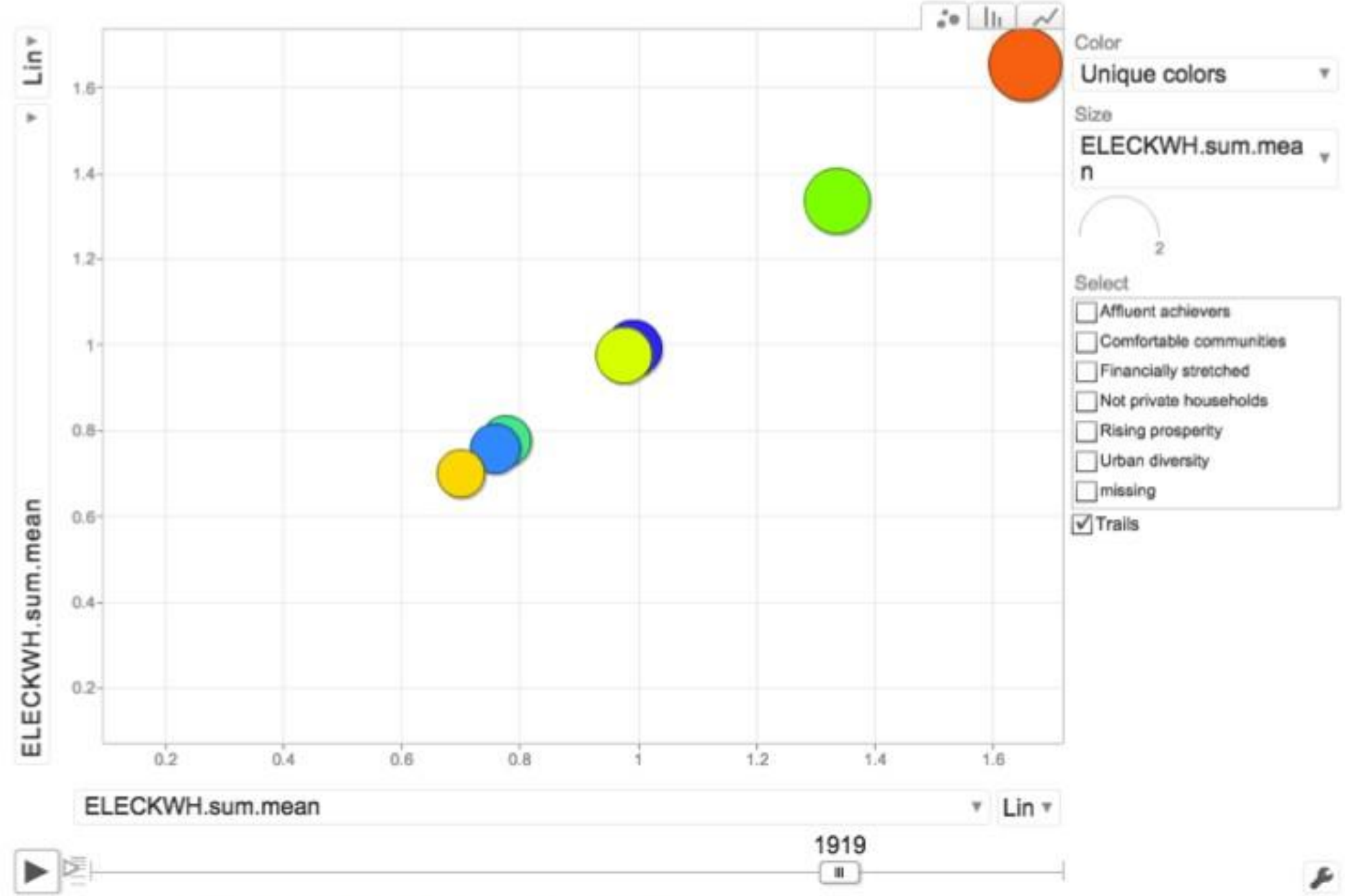
RStudio IDE for Mac OS X with execution console, code editor panels, and other default views.

Console – code execution and outputs

File explorer, packages management, plots viewer and help files

- After many years of R being used almost exclusively in Academia and research, recent years have witnessed an increased interest in the R language coming from business customers and the financial world.
- Companies such as Google and Microsoft have turned to R for its high flexibility and simplicity of use. In July 2015, Microsoft completed the acquisition of Revolution Analytics-a Big Data and predictive analytics company that gained its reputation for their own implementations of R with built-in support for Big Data processing and analysis.

- Apart from the obvious selling points of the R language, such as its open source license, a lack of any setup fees, unlimited access to a comprehensive set of ready-made statistical functions, and a highly active community of users, R is also widely praised for its data visualization capabilities and ability to work with data of many different formats.
- Popular and very influential newspapers and magazines such as the Washington Post, Guardian, the Economist, or the New York Times, use R on a daily basis to produce highly informative diagrams and info graphics, in order to explain complex political events or social and economical phenomena.
- The availability of static graphics through extremely powerful packages such as ggplot2 or lattice, has lately been extended even further to offer interactive visualizations using the shiny or ggobi frameworks, and a number of external packages (for example rCharts created and maintained by Ramnath Vaidyanathan) that support JavaScript libraries for spatial analysis, for example leaflet.js, or interactive data-driven documents through morris.js, D3.js, and others.
- Moreover, R users can now benefit from Google Visualization API charts, such as the famous motion graph, directly through googleVis package developed by Markus Gesmann, Diego de Castillo, and Joe Cheng (the following screenshot shows that googleVis package in action):



An example of a motion chart created by the Google Visualization package- an R interface to Google charts

- As mentioned earlier, R works with data coming from a large array of different sources and formats. This is not just limited to *physical* file formats such as traditional comma-separated or tab-delimited formats, but it also includes less common files such as JSON (a widely used format in web applications and modern NoSQL databases or images, other statistical packages and proprietary formats such as Excel workbooks, Stata, SAS, SPSS, and Minitab files, *scrapping* data from the Internet, or direct access to SQL or NoSQL databases as well as other Big Data containers (such as Amazon S3), or files stored in the HDFS.
- However if you would like to get a feel for a variety of data import methods in R right now, please visit the CRAN page at https://cran.r-project.org/manuals.html, which lists a set of the most essential manuals including the *R Data Import/Export* document outlining the import and export methods of data to and from R.
- Finally, the code in the R language can easily be called from other programming platforms such as Python or **Julia**. Moreover, R itself is able to implement functions and statements from other programming languages such as the C family, Java, SQL, Python, and many others. This allows for greater veracity and helps to integrate the R language into existing data science workflows.

SUMMARY

- In this Unit we discussed the ambiguity of Big Data definitions and highlighted its major features. We also talked about a deluge of Big Data sources, and mentioned that even one event, such as Messi's goal, can lead to an avalanche of large amounts of data being created almost instantaneously.
- You were then introduced to some most commonly used Big Data tools, such as Hadoop, its Distributed File System and the parallel MapReduce framework, traditional SQL and NoSQL databases, and the Apache Spark project, which allows faster (and in many cases easier) data processing than in Hadoop.
- We ended the chapter by presenting the origins of the R programming language, its gradual evolution into the most widely-used statistical computing environment, and the current position of R amongst a spectrum of Big Data analytics tools.
- In the next chapter you will finally have a chance to get your hands dirty and learn, or revise, a number of frequently used functions in R for data management, transformations, and analysis.

WORKING OF MAPREDUCE

- Here, we can see that the *Input* is provided to the Map() function then it's *output* is used as an input to the Reduce function and after that, we receive our final output. Let's understand What this Map() and Reduce() does.
- As we can see that an Input is provided to the Map(), now as we are using Big Data. The Input is a set of Data. The Map() function here breaks this DataBlocks into **Tuples** that are nothing but a key-value pair. These key-value pairs are now sent as input to the Reduce().
- The Reduce() function then combines this broken Tuples or key-value pair based on its Key value and form set of Tuples, and perform some operation like sorting, summation type job, etc. which is then sent to the final Output Node. Finally, the Output is Obtained.
- The data processing is always done in Reducer depending upon the business requirement of that industry. This is How First Map() and then Reduce is utilized one by one.

MAP TASK

- **RecordReader:** The purpose of ***recordreader*** is to break the records. It is responsible for providing **key-value** pairs in a **Map()** function. The key is its locational information and value is the data associated with it.
- **Map:** A map is nothing but a user-defined function whose work is to process the Tuples obtained from record reader. The Map() function either does not generate any key-value pair or generate multiple pairs of these tuples.
- **Combiner:** Combiner is used for grouping the data in the Map workflow. It is similar to a Local reducer. The intermediate key-value that are generated in the Map was combined with the help of this combiner. Using a combiner is not necessary as it is optional.
- **Partitioner:** Partitional is responsible for fetching key-value pairs generated in the Mapper Phases. The partitioner generates the shards corresponding to each reducer. Hashcode of each key is also fetched by this partition. Then partitioner performs it's(Hashcode) modulus with the number of reducers(*key.hashcode()%(number of reducers)).*

REDUCE TASK

- **Shuffle and Sort:** The Task of Reducer starts with this step, the process in which the Mapper generates the intermediate key-value and transfers them to the Reducer task is known as *Shuffling*. Using the Shuffling process the system can sort the data using its key value. Once some of the Mapping tasks are done Shuffling begins that is why it is a faster process and does not wait for the completion of the task performed by Mapper.
- **Reduce:** The main function or task of the Reduce is to gather the Tuple generated from Map and then perform some sorting and aggregation sort of process on those key-value depending on its key element.
- **OutputFormat:** Once all the operations are performed, the key-value pairs are written into the file with the help of record writer, each record in a new line, and the key and value in a space-separated manner.

APACHE SPARK VS. APACHE HADOOP

- Outside of the differences in the design of Spark and Hadoop MapReduce, many organizations have found these big data frameworks to be complimentary, using them together to solve a broader business challenge.

- Hadoop is an open source framework that has the Hadoop Distributed File System (HDFS) as storage, YARN as a way of managing computing resources used by different applications, and an implementation of the MapReduce programming model as an execution engine.
- In a typical Hadoop implementation, different execution engines are also deployed such as Spark, Tez, and Presto.

- Spark is an open source framework focused on interactive query, machine learning, and real-time workloads.
- It does not have its own storage system, but runs analytics on other storage systems like HDFS, or other popular stores like Amazon Redshift, Amazon S3, Couchbase, Cassandra, and others.
- Spark on Hadoop leverages YARN to share a common cluster and dataset as other Hadoop engines, ensuring consistent levels of service, and response.