Big Data and Analytics

Speaker
Vikramank Singh

Notemybook Services Pvt Ltd


WHAT WE KNOW…
THE REST…
BIG DATA

What is Big Data

Big Data is Like Teenage Sex

Everyone talks about it, nobody really knows how to do it, everyone thinks everyone else is doing it, so everyone claims they are doing it.

- Dan Ariely -

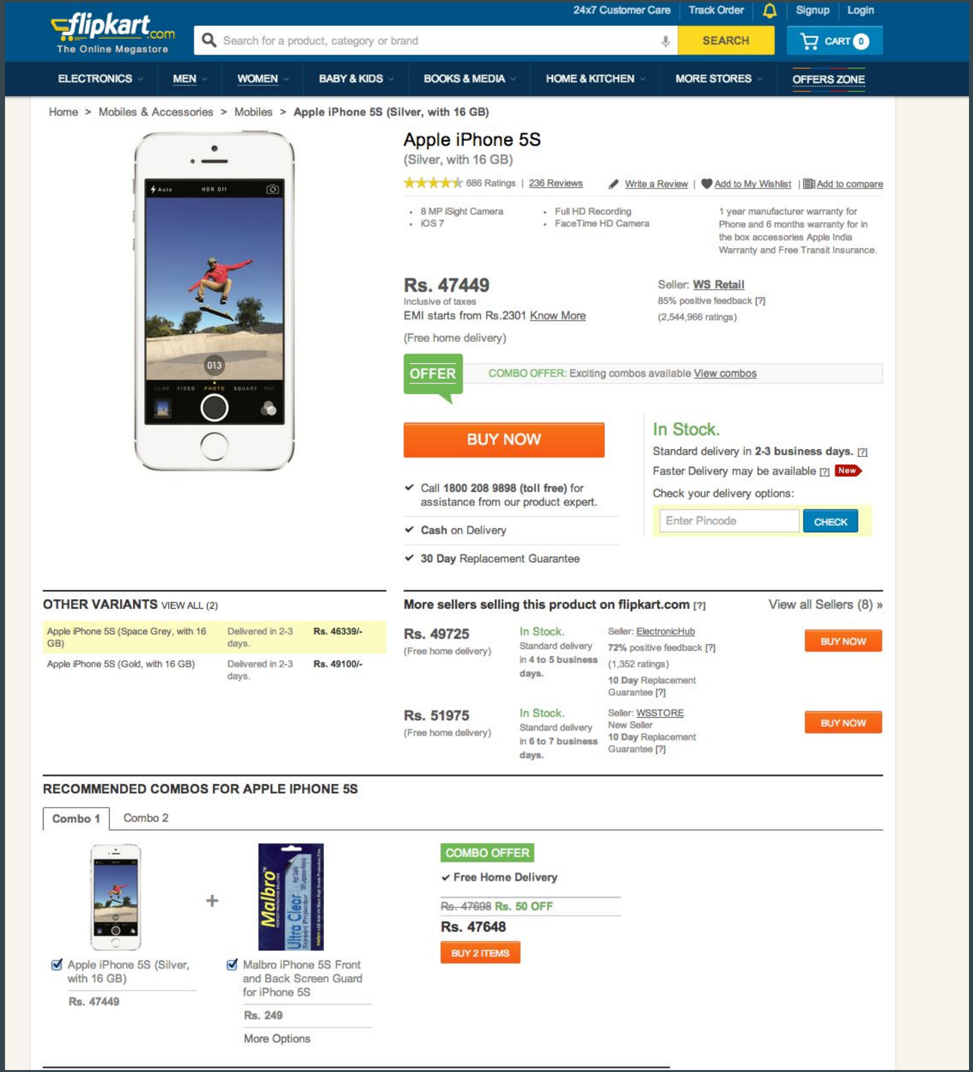
Big data is the term used to describe the **process of analyzing** complex set of data sets to **discover information** that could help make **better decisions** or find certain patterns that were previously unknown.

What is Analytics

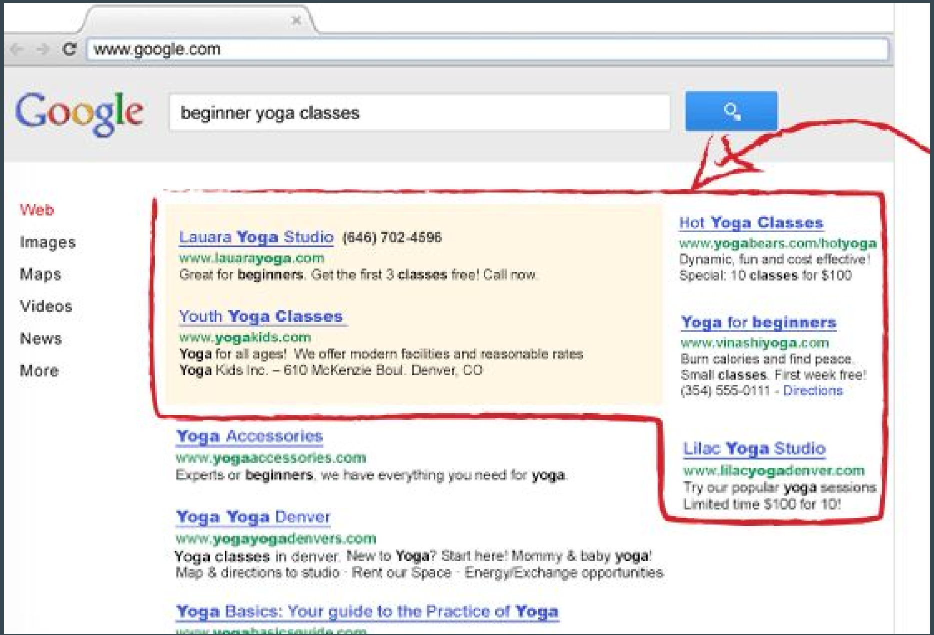
A Systematic **Computational Analysis** of Data or Statistics


470
4,971


T-Mobile
12:59 PM
Edit
fitbit
Today
Surge
Synced today, 12:48 PM
17,540 steps
47 bpm resting
14 miles
2,116 calories burned
29 floors
117 active minutes
1
Dashboard
Challenges
Friends
Account



flipkart.com
The Online Megastore
24x7 Customer Care
Track Order
Signup
Login
Search for a product, category or brand
SEARCH
CART 0
ELECTRONICS
MEN
WOMEN
BABY & KIDS
BOOKS & MEDIA
HOME & KITCHEN
MORE STORES
OFFERS ZONE
Home > Mobiles & Accessories > Mobiles > Apple iPhone 5S (Silver, with 16 GB)
Apple iPhone 5S
(Silver, with 16 GB)
686 Ratings | 236 Reviews
Write a Review
Add to My Wishlist
Add to compare
8 MP iSight Camera
iOS 7
Full HD Recording
FaceTime HD Camera
1 year manufacturer warranty for Phone and 6 months warranty for in the box accessories Apple India Warranty and Free Transit Insurance.
Rs. 47449
Inclusive of taxes
EMI starts from Rs.2301 Know More
(Free home delivery)
Seller: WS Retail
85% positive feedback [?]
(2,544,966 ratings)
OFFER
COMBO OFFER: Exciting combos available View combos
BUY NOW
In Stock.
Standard delivery in 2-3 business days. [?]
Faster Delivery may be available [?] New
Check your delivery options:
Enter Pincode
CHECK
Call 1800 208 9898 (toll free) for assistance from our product expert.
Cash on Delivery
30 Day Replacement Guarantee
OTHER VARIANTS VIEW ALL (2)
Apple iPhone 5S (Space Grey, with 16 GB)
Delivered in 2-3 days.
Rs. 46339/-
Apple iPhone 5S (Gold, with 16 GB)
Delivered in 2-3 days.
Rs. 49100/-
More sellers selling this product on flipkart.com [?]
View all Sellers (8) »
Rs. 49725
(Free home delivery)
In Stock.
Standard delivery in 4 to 5 business days.
Seller: ElectronicHub
72% positive feedback [?]
(1,352 ratings)
10 Day Replacement Guarantee [?]
BUY NOW
Rs. 51975
(Free home delivery)
In Stock.
Standard delivery in 6 to 7 business days.
Seller: WSSTORE
New Seller
10 Day Replacement Guarantee [?]
BUY NOW
RECOMMENDED COMBOS FOR APPLE IPHONE 5S
Combo 1
Combo 2
Apple iPhone 5S (Silver, with 16 GB)
Rs. 47449
Malbro iPhone 5S Front and Back Screen Guard for iPhone 5S
Rs. 249
More Options
COMBO OFFER
Free Home Delivery
Rs. 47698 Rs. 50 OFF
Rs. 47648
BUY 2 ITEMS



www.google.com
Google
beginner yoga classes
Web
Images
Maps
Videos
News
More
Lauara Yoga Studio (646) 702-4596
www.lauarayoga.com
Great for beginners. Get the first 3 classes free! Call now.
Youth Yoga Classes
www.yogakids.com
Yoga for all ages! We offer modern facilities and reasonable rates
Yoga Kids Inc. – 610 McKenzie Boul. Denver, CO
Hot Yoga Classes
www.yogabears.com/hotyoga
Dynamic, fun and cost effective!
Special: 10 classes for $100
Yoga for beginners
www.vinashiyoga.com
Burn calories and find peace.
Small classes. First week free!
(354) 555-0111 - Directions
Yoga Accessories
www.yogaaccessories.com
Experts or beginners, we have everything you need for yoga.
Lilac Yoga Studio
www.lilacyogadenver.com
Try our popular yoga sessions
Limited time $100 for 10!
Yoga Yoga Denver
www.yogayogadenvers.com
Yoga classes in denver. New to Yoga? Start here! Mommy & baby yoga!
Map & directions to studio · Rent our Space · Energy/Exchange opportunities
Yoga Basics: Your guide to the Practice of Yoga

Verizon
10:19 PM
MACDEMARCO
42 posts
41k followers
79 following
Following
Suggested
lovealienzzz
+ Follow
burgerrecords4life
BURGER RECORDS
+ Follow
wavvescybersquat
WAVVES
+ Follow

Why?

Facts

1. It was and is the **most demanding skill** to have across the globe since 2013.

2. The **salary** of a Data Scientist at Facebook is **$150,000** as compared to a Software Engineer who is paid $100,000.

3. McKinsey Global estimates that there will be a **shortage of 140,000 to 190,000** trained data analysts in the United States by 2018.

4. Companies like **Apple, Microsoft, Nvidia, Google[X], Baidu** are investing Billions of dollars in developing their core data science teams to gear up for the future.

5. The most flexible skills which can help you enter any field or specification with respect to job opportunities.

Google

Bai du 百度

Microsoft

NVIDIA®

Statistical Programming using **R**

1. Expressions

Let's try some simple math. Type the below command.

```
> 1+1
[1] 2
```

Type the string "Big Data Analytics Workshop". (Don't forget the quotes!)

```
> "Arr, matey!"
[1] "Arr, matey!"
```

Now try multiplying 6 times 7 (* is the multiplication operator).

```
> 6*7
[1] 42
```

2. Logical Values

Some expressions return a "logical value": TRUE or FALSE. (Many programming languages refer to these as "boolean" values.) Let's try typing an expression that gives us a logical value:

```
> 3 < 4
[1] TRUE
```

And another logical value (note that you need a double-equals sign to check whether two values are equal - a single-equals sign won't work):

```
> 2 + 2 == 5
[1] FALSE
```

T and F are shorthand for TRUE and FALSE. Try this:

```
> T == TRUE
[1] TRUE
```

3. Variables

1. As in other programming languages, you can store values into a variable to access it later. Type x <- 42 to store a value in x.

```
> x <- 42
```

x can now be used in expressions in place of the original result. Try dividing x by 2 (/ is the division operator).

```
> x/2
[1] 21
```

You can re-assign any value to a variable at any time. Try assigning "Arr, matey!" to x.

```
> x <- "Arr, matey!"
[1] "Arr, matey!"
```

You can print the value of a variable at any time just by typing its name in the console. Try printing the current value of x.

```
> x[1] "Arr, matey!"
```

Now try assigning the TRUE logical value to x.

```
> x <- TRUE
[1] TRUE
```

4. Functions

1. You call a function by typing its name, followed by one or more arguments to that function in parenthesis. Let's try using the sum function, to add up a few numbers. Enter:
2.
```
> sum(1,3,5)
[1] 9
```

3. Some arguments have names. For example, to repeat a value 3 times, you would call the rep function and provide its times argument:
4. `rep("Yo ho!", times = 3)`
5. `[1] "Yo ho!" "Yo ho!" "Yo ho!"`

Try calling the sqrt function to get the square root of 16.

```
> sqrt(16)
[1] 4
```

Help Function

help(functionname) brings up help for the given function. Try displaying help for the sum function:

```
> help(sum)
sum                 package:base                R Documentation

Sum of Vector Elements

Description:

    'sum' returns the sum of all the values present in its arguments.

Usage:

    sum(..., na.rm = FALSE)
...
```

KEEP
CALM
AND
TAKE A
BREAK

Loop Statements

There may be a situation when you need to execute a block of code several number of times. In general, statements are executed sequentially.

R - Repeat Loop

The **Repeat loop** executes the same code again and again until a stop condition is met.

```
repeat {

   commands

   if(condition) {

      break

   }

}
```

R - While Loop

The **While loop** executes the same code again and again until a stop condition is met.

```
while (test_expression) {

   statement

}
```

R - For Loop

A **For loop** is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

```
for (value in vector) {

   statements

}
```

```
Eg : v <- LETTERS[1:4]

      for ( i in v) {

      print(i)

      }
```

Vectors

A vector's values can be numbers, strings, logical values, or any other type, as long as they're all the *same* type. Try creating a vector of numbers, like this:

```
> c(4,7,9)
[1] 4 7 9
```

Now try creating a vector with strings:

```
> c('a', 'b', 'c')
[1] "a" "b" "c"
```

Vectors cannot hold values with different modes (types). Try mixing modes and see what happens:

```
> c(1, TRUE, "three")
[1] "1"     "TRUE"  "three"
```

All the values were converted to a single mode (characters) so that the vector can hold them all.

If you need a vector with a sequence of numbers you can create it with start:end notation. Let's make a vector with values from 5 through 9:

```
> 5:9
[1] 5 6 7 8 9
```

A more versatile way to make sequences is to call the seq function. Let's do the same thing with seq:

```
> seq(5, 9)
[1] 5 6 7 8 9
```

seq also allows you to use increments other than 1. Try it with steps of 0.5:

```
> seq(5, 9, 0.5)
[1] 5.0 5.5 6.0 6.5 7.0 7.5 8.0 8.5 9.0
```

Now try making a vector with integers from 9 down to 5:

```
> 9:5
[1] 9 8 7 6 5
```

You can retrieve an individual value within a vector by providing its numeric index in square brackets. Try getting the third value:

```
> sentence <- c('walk', 'the', 'plank')
> sentence[3]
[1] "plank"
```

You can use a vector within the square brackets to access multiple values. Try getting the first *and* third words:

```
> sentence[c(1,3)]
[1] "walk" "dog"
```

This means you can retrieve ranges of values. Get the second through fourth words:

```
> sentence[2:4]
[1] "the" "dog" "to"
```

You can also set ranges of values; just provide the values in a vector. Add words 5 through 7:

```
> sentence[5:7] <- c('the', 'poop', 'deck')
```

Vectors Names

You can assign names to a vector's elements by passing a second vector filled with names to the names assignment function, like this:

```
> ranks <- 1:3
> names(ranks) <- c("first", "second", "third")
```

Now try passing the vector to the barplot function:

```
> vesselsSunk <- c(4, 5, 1)
> barplot(vesselsSunk)

> names(vesselsSunk) <- c("England", "France", "Norway")
```

Now, if you call barplot with the vector again, you'll see the labels:

```
> barplot(vesselsSunk)
```

```
> barplot(1:100)
```

If you add a scalar (a single value) to a vector, the scalar will be added to each value in the vector, returning a new vector with the results. Try adding 1 to each element in our vector:

```
> a <- c(1, 2, 3)
> a + 1
[1] 2 3 4
```

The same is true of division, multiplication, or any other basic arithmetic. Try dividing our vector by 2:

```
> a / 2
[1] 0.5 1.0 1.5
```

Functions that normally work with scalars can operate on each element of a vector, too. Try getting the sine of each value in our vector:

```
> sin(a)
[1] 0.8414710 0.9092974 0.1411200
```

Scatter Plots

The plot function takes two vectors, one for X values and one for Y values, and draws a graph of them.

```
> x <- seq(1, 20, 0.1)
> y <- sin(x)
```

Then simply call plot with your two vectors:

```
> plot(x, y)
```

We'll create a vector with some negative and positive values for you, and store it in the values variable.
We'll also create a second vector with the absolute values of the first, and store it in the absolutes variable.
Try plotting the vectors, with values on the horizontal axis, and absolutes on the vertical axis.

```
> values <- -10:10
> absolutes <- abs(values)
> plot(values, absolutes)
```

NA Values

Sometimes, when working with sample data, a given value **isn't available**. But it's not a good idea to just throw those values out. R has a value that explicitly indicates a sample was not available: NA. Many functions that work with vectors treat this value specially.

We'll create a vector for you with a missing sample, and store it in the a variable.

Try to get the sum of its values, and see what the result is:

```
> a <- c(1, 3, NA, 7, 9)
> sum(a)
[1] NA
```

Now, try out the help command for sum. You can see that sum function accepts a NA.rm command. It's set to FALSE by default, but if you set it to TRUE, all NA arguments will be removed from the vector before the calculation is performed.

```
> sum(a, na.rm = TRUE)
[1] 20
```

Matrices

Let's make a matrix 3 rows high by 4 columns wide, with all its fields set to 0.

```
> matrix(0, 3, 4)
     [,1] [,2] [,3] [,4]
[1,]    0    0    0    0
[2,]    0    0    0    0
[3,]    0    0    0    0
```

You can also use a vector to initialize a matrix's value. To fill a 3x4 matrix, you'll need a 12-item vector. We'll make that for you now:

```
> a <- 1:12
```

Now call matrix with the vector, the number of rows, and the number of columns:

```
> matrix(a, 3, 4)
```

The vector's values are copied into the new matrix, one by one. You can also re-shape the vector itself into a matrix. Create an 8-item vector:

```
> plank <- 1:8
```

The dim assignment function sets *dim*ensions for a matrix. It accepts a vector with the number of rows and the number of columns to assign.

```
> dim(plank) <- c(2, 4)
```

Now print plank and check the matrix.

Getting values from matrices isn't that different from vectors; you just have to provide two indices instead of one.

```
> plank[2, 3]
[1] 6
```

You can get an entire row of the matrix by omitting the column index (but keep the comma). Try retrieving the second row:

```
> plank[2,]
[1] 2 4 6 8
```

To get an entire column, omit the row index. Retrieve the fourth column:

```
> plank[, 4]
[1] 7 8
```

You can read multiple rows or columns by providing a vector or sequence with their indices. Try retrieving columns 2 through 4:

```
> plank[, 2:4]
     [,1] [,2] [,3]
[1,]    3    5    7
[2,]    4    6    8
```

Plotting Matrix:

Create a matrix ‘elevation’ with (1,10,10).

```
> elevation <- matrix(1, 10, 10)
```

```
> elevation[4, 6] <- 0
```

```
> contour(elevation)
```

Or you can create a 3D perspective plot with the persp function:

```
> persp(elevation)
```

Lists

An ordered collection of data of arbitrary types.

```
> doe = list(name="john",age=28,married=F)
> doe$name
[1] "john“
> doe$age
[1] 28
```

Typically, vector elements are accessed by their index (an integer), list elements by their name (a character string). But both types support both access methods.

Data Frames

Data frame: is supposed to represent the typical data table that researchers come up with – like a spreadsheet.
It is a rectangular table with rows and columns; data within each column has the same type (e.g. number, text, logical), but different columns may have different types.

```
# Create the data frame.
emp.data <- data.frame(
   emp_id = c (1:5),
   emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
   salary = c(623.3,515.2,611.0,729.0,843.25),

   start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
      "2015-03-27")),
   stringsAsFactors = FALSE
)
# Print the data frame.
print(emp.data)
```

When we execute the above code, it produces the following result –

```
emp_id    emp_name     salary     start_date
1      1      Rick        623.30     2012-01-01
2      2      Dan         515.20     2013-09-23
3      3      Michelle    611.00     2014-11-15
4      4      Ryan        729.00     2014-05-11
5      5      Gary        843.25     2015-03-27
```

The structure of the data frame can be seen by using **str()** function.

```
# Get the structure of the data frame.
str(emp.data)
```

```
'data.frame':   5 obs. of  4 variables:
 $ emp_id    : int  1 2 3 4 5
 $ emp_name  : chr  "Rick" "Dan" "Michelle" "Ryan" ...
 $ salary    : num  623 515 611 729 843
 $ start_date: Date, format: "2012-01-01" "2013-09-23" "2014-11-15" "2014-05-11" ...
```

Extract specific column from a data frame using column name.

```
# Extract Specific columns.
result <- data.frame(emp.data$emp_name,emp.data$salary)
print(result)
```

```
 emp.data.emp_name emp.data.salary
1              Rick          623.30
2               Dan          515.20
3          Michelle          611.00
4              Ryan          729.00
5              Gary          843.25
```

```
# Extract first two rows.
result <- emp.data[1:2,]
print(result)
```

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

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

```
 emp_name start_date

3 Michelle 2014-11-15

5     Gary 2015-03-27
```

Add the "dept" column.

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

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.

```r
# Create the second data frame
emp.newdata <-   data.frame(
   emp_id = c (6:8),
   emp_name = c("Rasmi","Pranab","Tusar"),
   salary = c(578.0,722.5,632.8),
   start_date = as.Date(c("2013-05-21","2013-07-30","2014-06-17")),
   dept = c("IT","Operations","Fianance"),
   stringsAsFactors = FALSE
)
# Bind the two data frames.
emp.finaldata <- rbind(emp.data,emp.newdata)
print(emp.finaldata)
```

```
emp_id     emp_name    salary     start_date       dept
1      1     Rick        623.30     2012-01-01       IT
2      2     Dan         515.20     2013-09-23       Operations
3      3     Michelle    611.00     2014-11-15       IT
4      4     Ryan        729.00     2014-05-11       HR
5      5     Gary        843.25     2015-03-27       Finance
6      6     Rasmi       578.00     2013-05-21       IT
7      7     Pranab      722.50     2013-07-30       Operations
8      8     Tusar       632.80     2014-06-17       Fianance
```

Factors

Factors are the data objects which are used to categorize the data and store it as levels. They can store both strings and integers. They are useful in the columns which have a limited number of unique values. Like "Male, "Female" and True, False etc. They are useful in data analysis for statistical modeling.

Factors are created using the **factor ()** function by taking a vector as input.

```
# Create a vector as input.
data <- c("East","West","East","North","North","East","West","West","West","East","North")

print(data)
print(is.factor(data))

# Apply the factor function.
factor_data <- factor(data)

print(factor_data)
print(is.factor(factor_data))
```

```
[1] "East"  "West"  "East"  "North" "North" "East"  "West"  "West"  "West"  "East" "North"

[1] FALSE

 [1] East  West  East  North North East  West  West  West  East  North

Levels: East North West

[1] TRUE
```

```
# Create the vectors for data frame.
height <- c(132,151,162,139,166,147,122)
weight <- c(48,49,66,53,67,52,40)
gender <- c("male","male","female","female","male","female","male")

# Create the data frame.
input_data <- data.frame(height,weight,gender)
print(input_data)

# Test if the gender column is a factor.
print(is.factor(input_data$gender))

# Print the gender column so see the levels.
print(input_data$gender)
```

R - Packages

R packages are a collection of R functions, complied code and sample data. They are stored under a directory called **"library"** in the R environment. By default, R installs a set of packages during installation.

Get the list of all the packages installed

```
> library()
```

Install directly from CRAN

The following command gets the packages directly from CRAN webpage and installs the package in the R environment.

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

R - CSV Files

In R, we can read data from files stored outside the R environment. We can also write data into files which will be stored and accessed by the operating system.

The csv file is a text file in which the values in the columns are separated by a comma. Let's consider the following data present in the file named **input.csv**.

```
data <- read.csv("input.csv")
print(data)
```

KEEP
CALM
AND
TAKE A
BREAK

Machine Learning

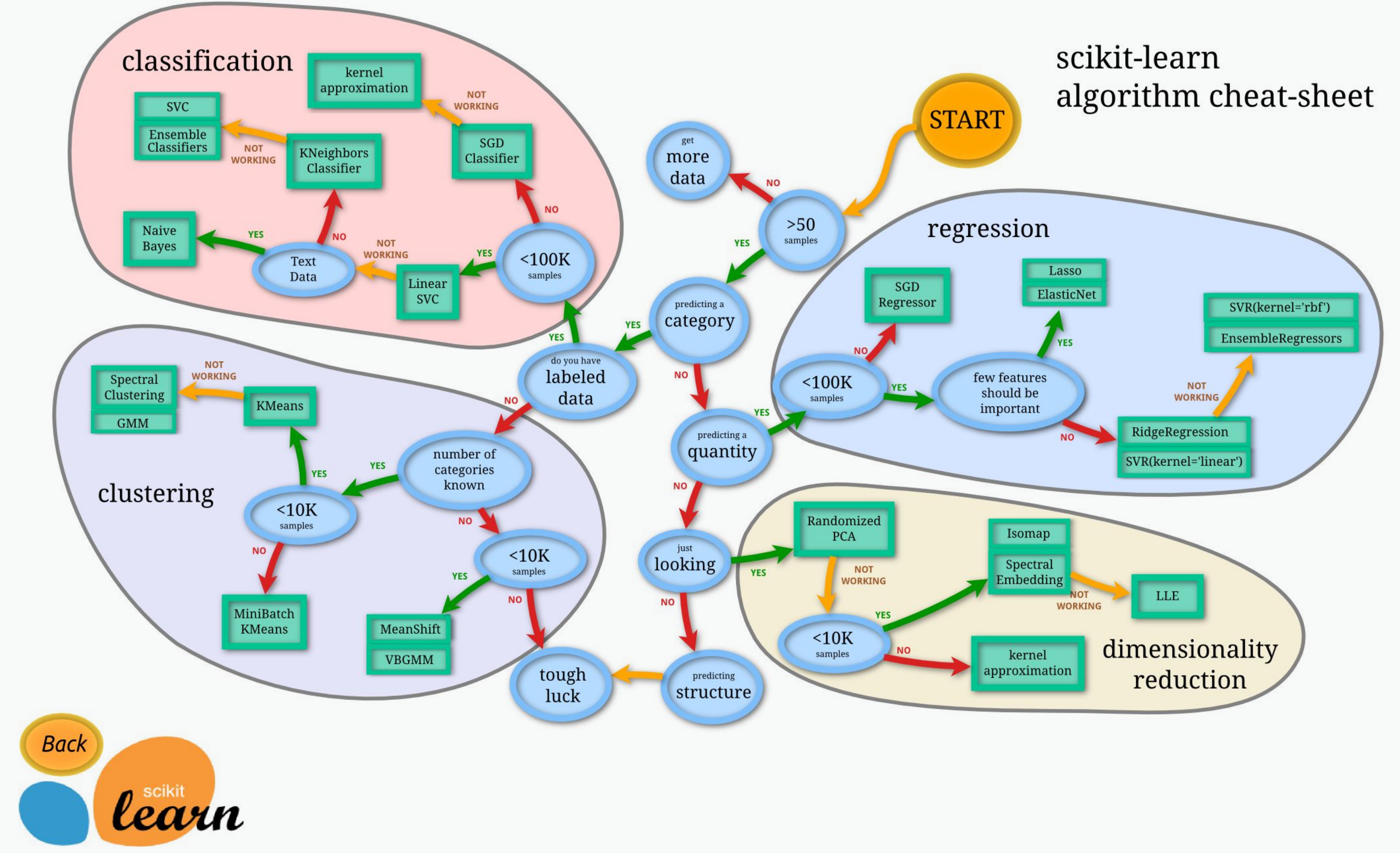
Machine Learning

Supervised Learning

Supervised learning is the machine **learning** task of inferring a function from labeled training data. The training data consist of a set of training examples.

Unsupervised Learning

Unsupervised learning is a type of machine **learning** algorithm used to draw inferences from datasets consisting of input data without labeled responses.



scikit-learn
algorithm cheat-sheet
START
>50 samples
NO
get more data
YES
predicting a category
YES
do you have labeled data
YES
<100K samples
YES
Linear SVC
NOT WORKING
Text Data
YES
Naive Bayes
NO
KNeighbors Classifier
NOT WORKING
SVC
Ensemble Classifiers
NO
SGD Classifier
NOT WORKING
kernel approximation
classification
NO
number of categories known
YES
<10K samples
YES
KMeans
NOT WORKING
Spectral Clustering
GMM
NO
MiniBatch KMeans
NO
<10K samples
YES
MeanShift
VBGMM
NO
tough luck
clustering
NO
predicting a quantity
YES
<100K samples
NO
SGD Regressor
YES
few features should be important
YES
Lasso
ElasticNet
NO
RidgeRegression
SVR(kernel='linear')
NOT WORKING
SVR(kernel='rbf')
EnsembleRegressors
regression
NO
just looking
YES
Randomized PCA
NOT WORKING
<10K samples
YES
Isomap
Spectral Embedding
NOT WORKING
LLE
NO
kernel approximation
dimensionality reduction
NO
predicting structure
Back
scikit learn

Linear regression

One of the most common modeling approaches in statistical learning is linear regression.

In R, use the **lm function** to generate these models. The general form of a linear regression model is
$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_k X_k + \varepsilon$, where ε is normally distributed with mean 0 and some variance σ^2. Let y be a vector of dependent variables, and x1 and x2 be vectors of independent variables. We want to find the coefficients of the linear regression model $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \varepsilon$. The following commands generate the linear regression model and give a summary of it.

```
> lm_model <- lm(y ~ x1 + x2, data=as.data.frame(cbind(y,x1,x2)))
> summary(lm_model)
```

The vector of coefficients for the model is contained in lm model$coefficients.

Prediction

For most of the following algorithms (as well as linear regression), we would in practice first generate the model using training data, and then predict values for test data. To make predictions, we use the predict function.

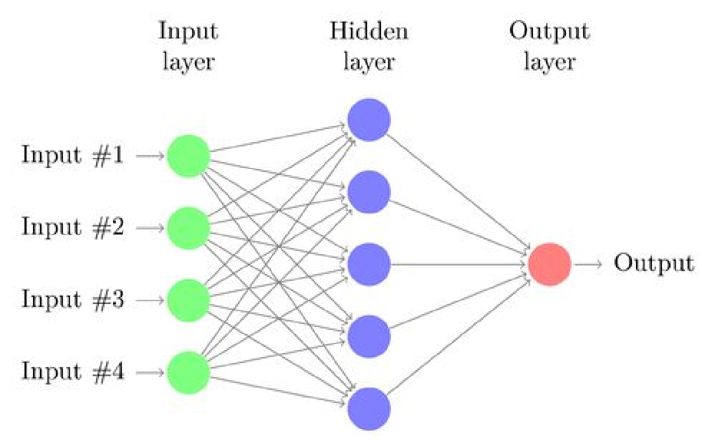
```
> predicted_values <- predict(lm_model, newdata=as.data.frame(cbind(x1_test, x2_test)))
```

Live Project

Neural Networks

In machine learning and cognitive science, **artificial neural networks (ANNs)** are a family of models inspired by biological neural networks (the central nervous systems of animals, in particular the brain) and are used to estimate or approximate functions that can depend on a large number of inputs and are generally unknown.

Artificial neural networks are generally presented as systems of interconnected "neurons" which exchange messages between each other. The connections have numeric weights that can be tuned based on experience, making neural nets adaptive to inputs and capable of learning.



We are trying to predict the workload level using the Artificial Neural Network.

1. Divide the given dataset in a 2:1 ratio as training and testing dataset.
2. Name the training dataset df1 with having 20 values.
3. Use the training methods to train the network using the df1.
4. Then, once the network is trained, use the predict method to test the data on the remaining 10 values.
5. Now, compare the values of the predicted and the actual values of the test dataset.

install.packages(“neuralnet”)

```
net <- neuralnet
(df1$BL+df1$LWL+df1$HWL~AF3+F7+F3+FC5+T7+P7+O1+O2+P8+T8+FC6+F4+F8+AF4, df1, hidden =
8, rep = 10, algorithm = "rprop+", linear.output = FALSE)

> net

> c <- df2[1,]
> j <- compute(net, c)$net.result
```

Thank You