DATA SCIENCE USING R

Data Science

"……THE SEXIEST JOB OF THE 21st Century"

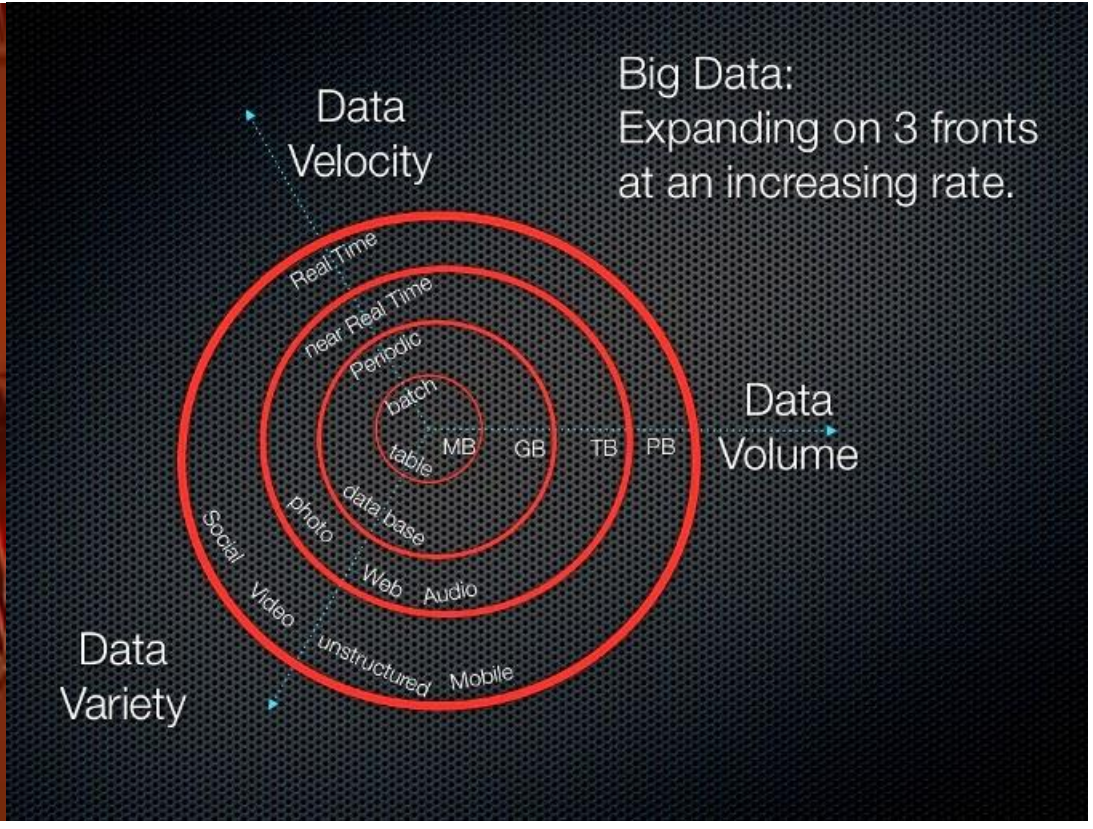
– Harvard Business Review

Big Data

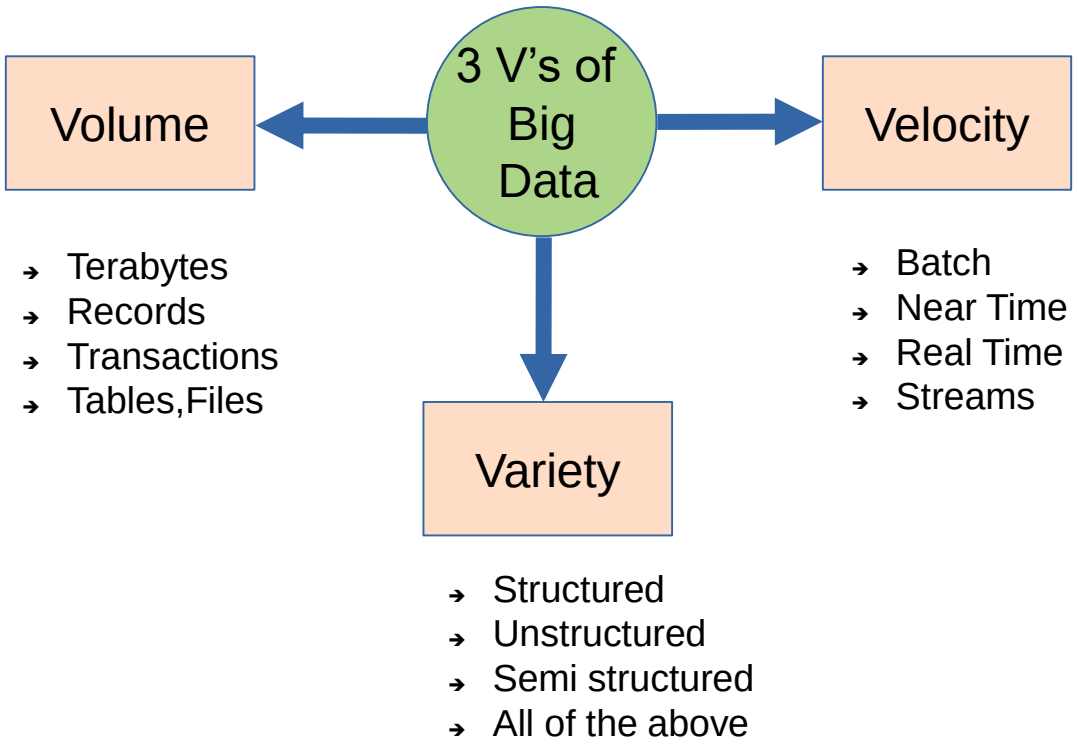
Sources of Big Data

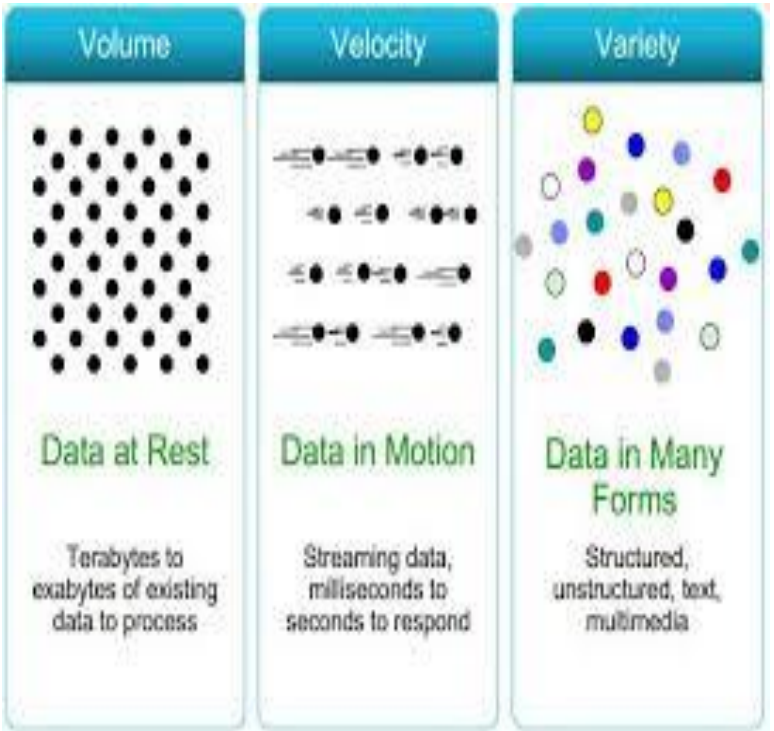
What is Big Data?

Used to describe large collections of data sets that can be unstructured and growing so large that can be difficult to manage by traditional databases.



3 V’s of Big Data





Big Data Scenarios

Big Data Scenarios: Online Generated data

Generates stream of data that can be analyzed.

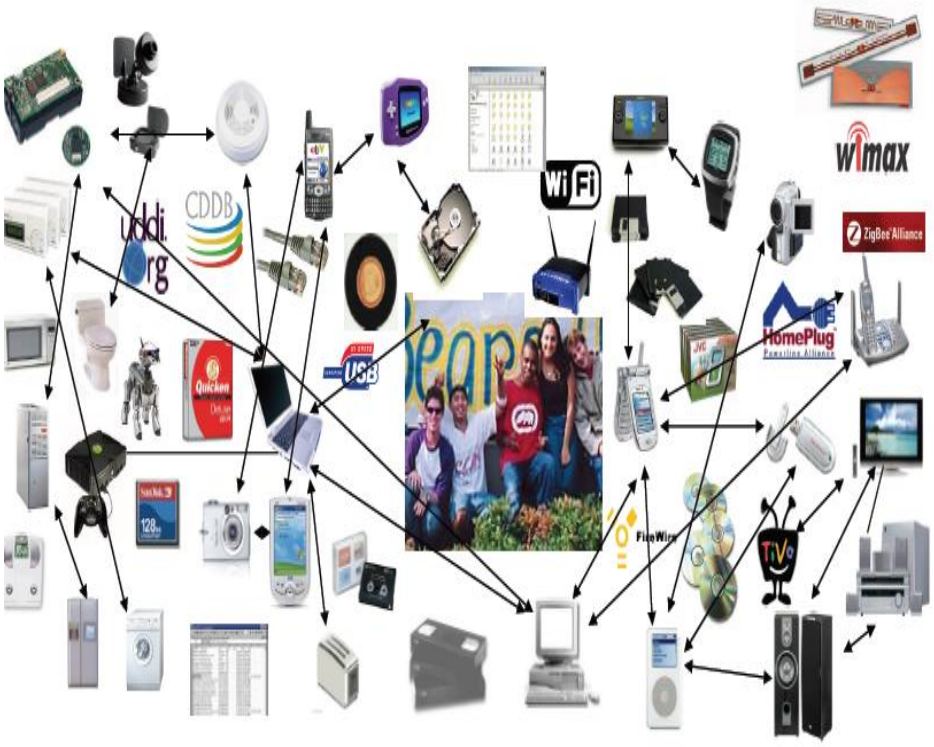
Examples

- Clicks
- Transactions
- Network messages
- Play, fast forward, pause
- Tweets, wall posts, friending
- Mails, chats

Big Data Scenarios: Social Media

- Emails
- Photos
- Videos
- Chats
- Phone Calls
- Social Media

Big Data Scenarios: Machine Generated Data



Machines Operational data from sensors

Examples

- 1TB data/hour of flight
- Terabytes from smart grids
- Smart phones usage data

Big Data Scenarios: Machine Generated Data

- Click Stream Data (navigation from product pages to placing order)
- Order data (products, features, prices, billing)
- Fulfillment data/ Services data

Big Data Scenarios: Graph Data

- Graph analytics, also known as network analysis, is an exciting new area for analytics workloads.
- Examples
  - Social Networks
  - Communication networks
  - Spread of infections
  - Collaborations & Relationships
  - Citations
  - Road networks

Measures of Data Volume

Multiples of bytes					V · T · E
SI decimal prefixes		Binary usage	IEC binary prefixes		
Name (Symbol)	Value		Name (Symbol)	Value	
kilobyte (kB)	10^{3}	2^{10}	kibibyte (KiB)	2^{10}	
megabyte (MB)	10^{6}	2^{20}	mebibyte (MiB)	2^{20}	
gigabyte (GB)	10^{9}	2^{30}	gibibyte (GiB)	2^{30}	
terabyte (TB)	10^{12}	2^{40}	tebibyte (TiB)	2^{40}	
petabyte (PB)	10^{15}	2^{50}	pebibyte (PiB)	2^{50}	
exabyte (EB)	10^{18}	2^{60}	exbibyte (EiB)	2^{60}	
zettabyte (ZB)	10^{21}	2^{70}	zebibyte (ZiB)	2^{70}	
yottabyte (YB)	10^{24}	2^{80}	yobibyte (YiB)	2^{80}	
See also: Multiples of bits · Orders of magnitude of data					

How much Data is generated?

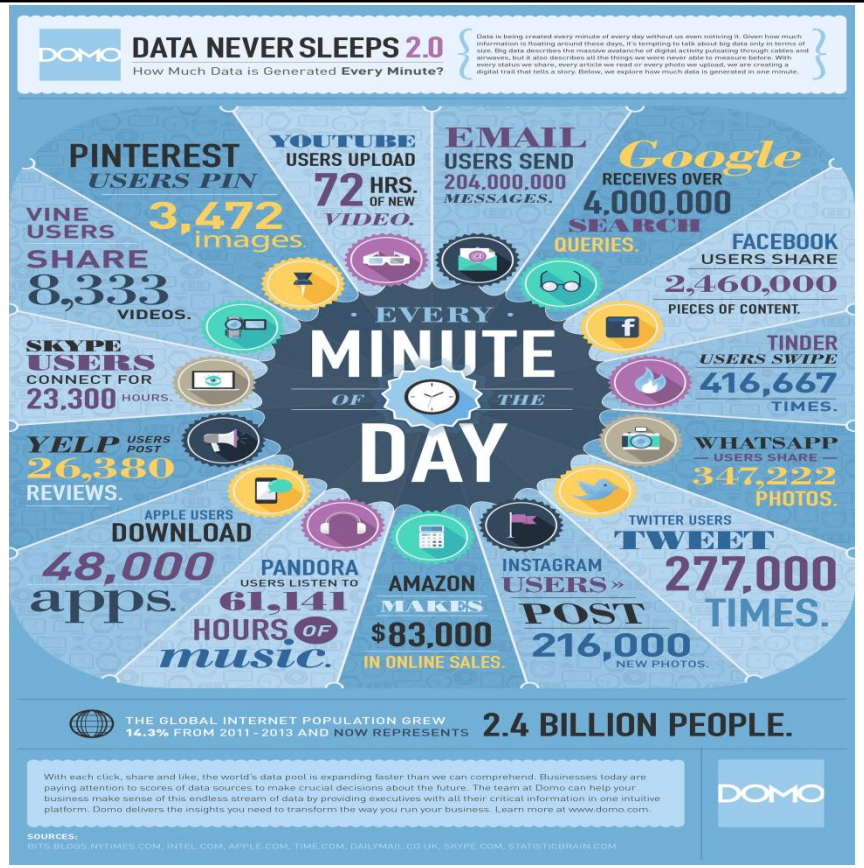
- Google processes 1000's of PB/day, 48 hours of video uploaded every minute
- Facebook has 100 PB and collects 500 TB/day

http://gigaom.com/data/facebook-is-collecting-your-data-500-terabytes-a-day/

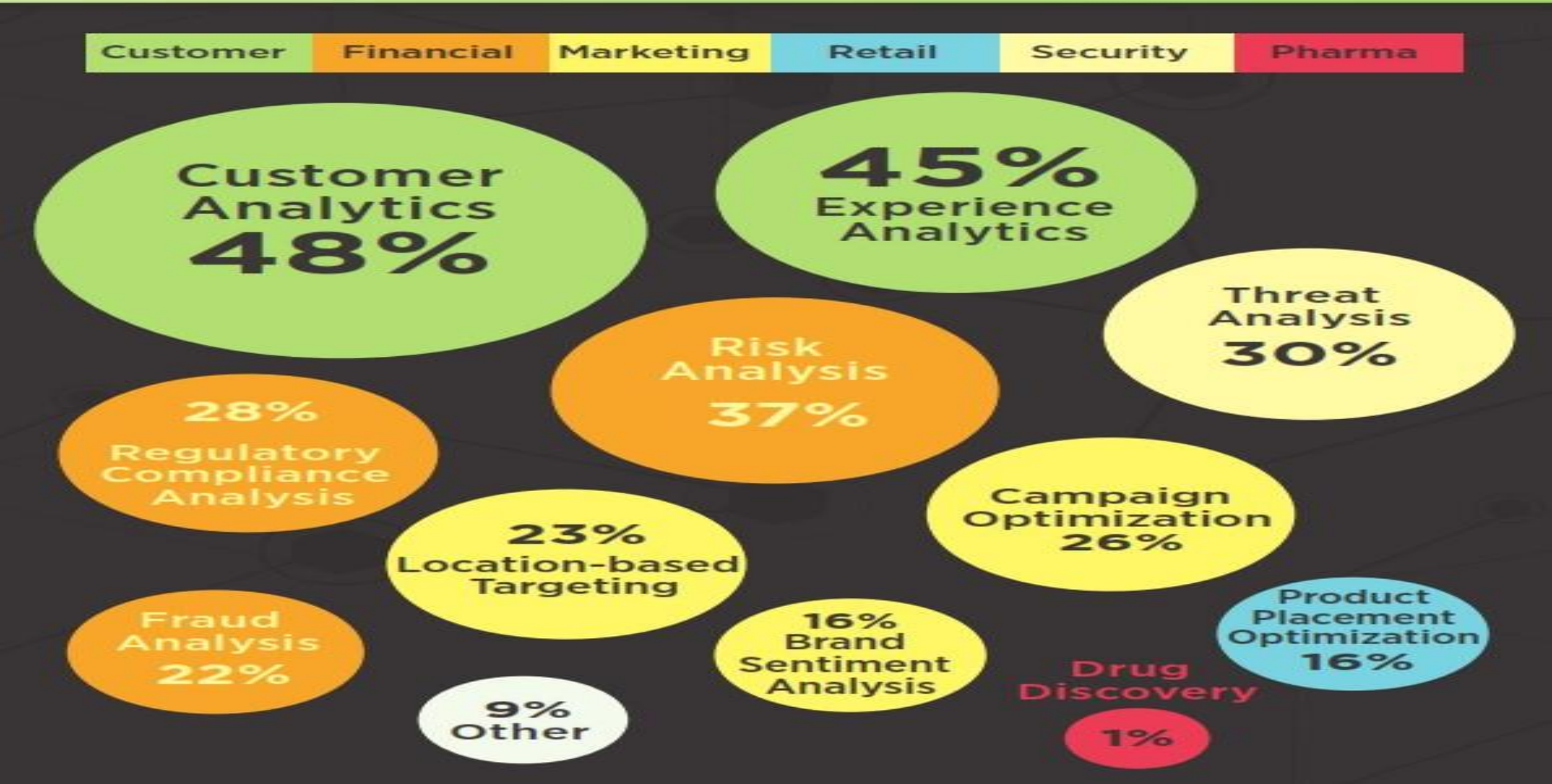
- eBay has 10 PB of user data + 100 TB/day

http://www.zdnet.com/au/how-ebay-uses-big-data-to-make-you-buy-more-7000006498/

- CERN's Large Hadron Collider (LHC) generates 25 PB a year



TOP BIG DATA USE CASES



How is Big Data used?

Linked In

➢LinkedIn Uses Big Data to increase profile visibility and enhance professional brands

Amazon

➢Buyer preferences and recommendations from clicks data

Dunnhumby

➢Understand the product substitutes from petabytes of loyalty card data

Facebook

➢Who are the influencer's and who are the followers

Boeing

➢Sensors data for likelihood of failure, real time vs. batch processing

Twitter

➢Customer Sentiment, Politicians gauging the interests of populations

How is Big Data used?...contd

Reports (E.g. Tracking business performance)

Diagnosis

- Why is user engagement dropping
- Why is system slow
- Detect viruses, spam, worms etc.

Decision

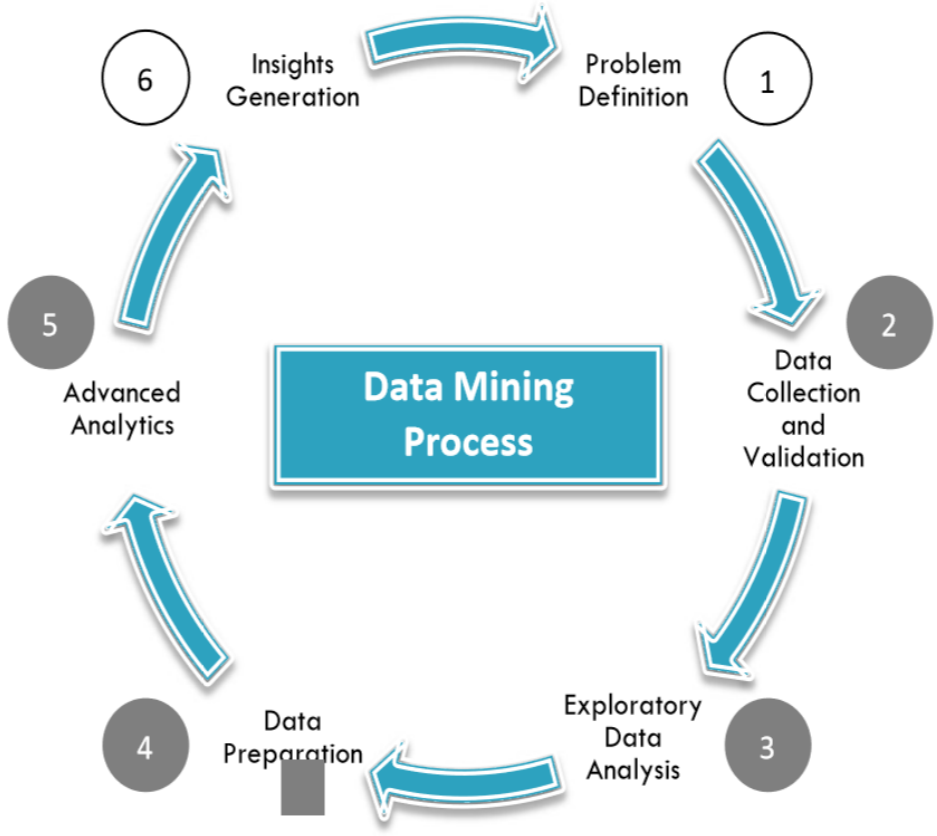
- Decide what features to add
- Decide what ads to show
- Block viruses, worms and spam

Introduction to Data Analytics

Introduction to Data Mining

Data mining

- ✓ Data mining is the process of analyzing data and identifying patterns from different perspectives, that help summarizing the data into useful information.



Data Collection

Objective

- ✓ Data collection is the process of identifying and assembling data required for analysis.
- ✓ Since an analysis may require a variety of data residing in different locations (servers) and in different formats, data needs to be combined such that all the information can be used effectively for analysis.

Approach

Identify data that can be used to analyze a given business problem

Once the data accesses are secured, move data across platforms and integrate metadata from different sources

Identify joining keys for various tables, if there is a need to combine information from multiple tables in a single view.

Data Validation

Objective

- ✓ Data validation is needed to identify data inconsistencies. Data quality is a very important factor for ensuring quality analysis.
- ✓ Real data in its raw format is often unclean (noisy and inconsistent).It could also be incomplete and redundant.
- ✓ Initial screening is essential to identify such data issues.

Approach

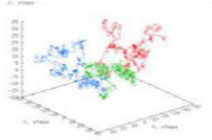
- General data checks
- Measured parameter checks
- Data completeness checks
- Missing values checks
- Data redundancies checks

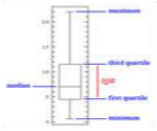
Exploratory Data Analysis (EDA)

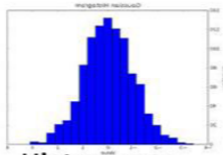
Objective

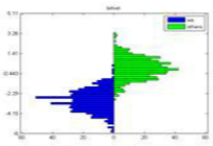
- ✓ Exploratory Data Analysis is an approach of allowing the data to reveal its underlying structure and model, without making any initial assumptions. It is a philosophy as to how we dissect a dataset, what we look for, how we look and how we interpret data.

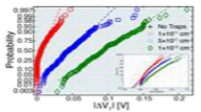
Some Common Techniques

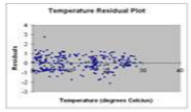

Scatter plots

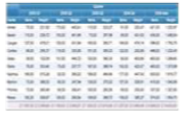

Box plots

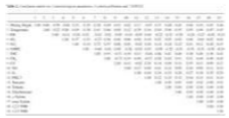

Histograms


Bi-histograms


Probability plots


Residual plots


Crosstab


Correlation Matrix

Methods for EDA

Univariate non graphical

Measures of Central tendency, Measures of dispersion, Measures of symmetry

Univariate graphical

Line Chart, Box and whisker plot, Histogram, QQ Plot, Steam and Leaf plot

Multivariate non graphical

Cross tabulation, Correlation Matrix, Category wise Univariate statistics

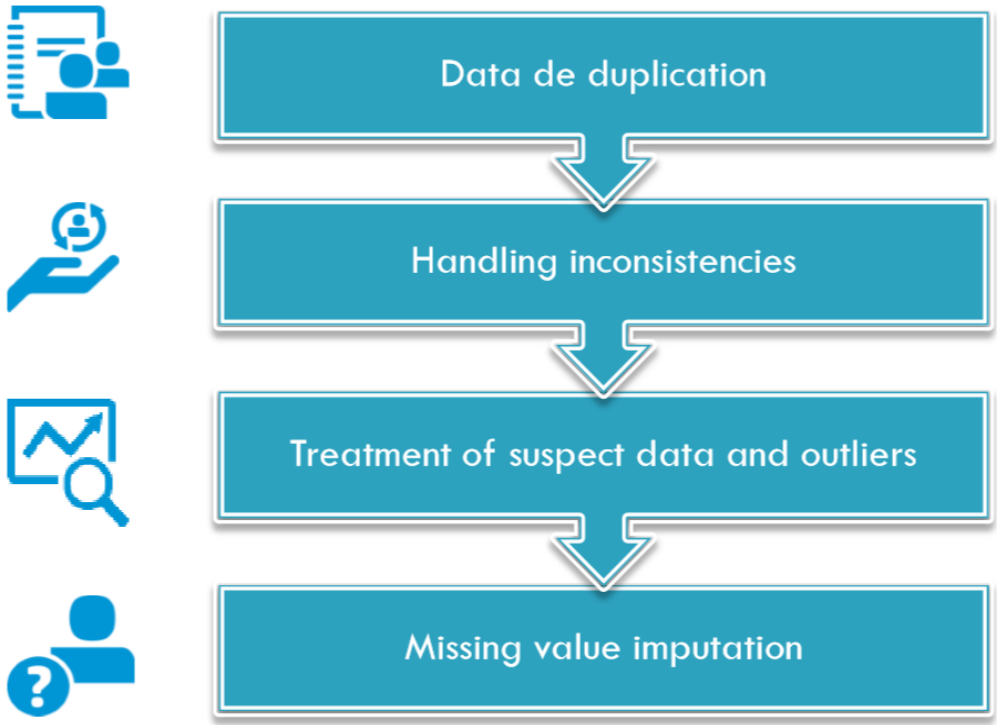
Multivariate graphical

Scatter plots, Univariate graphs by category, Area charts

Data Cleaning

Objective

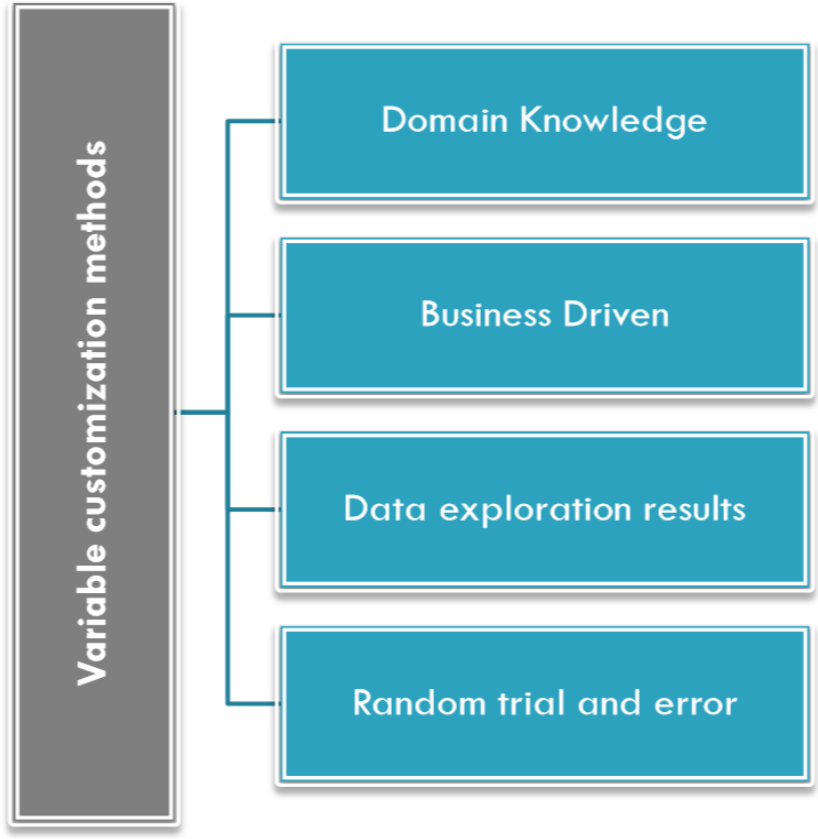
- ✓ Data Cleansing is the process of applying treatments to improve data quality and meet analysis requirements.
- ✓ Implementation of most statistical or machine learning algorithms require that the data should be in a particular format.
- ✓ Data redundancies need to be handled with care to avoid loss of information.
- ✓ Anomalies and exceptions almost always occur in real data, but care has to be taken so that they don't have significant impact on the analysis.



Variable Customization

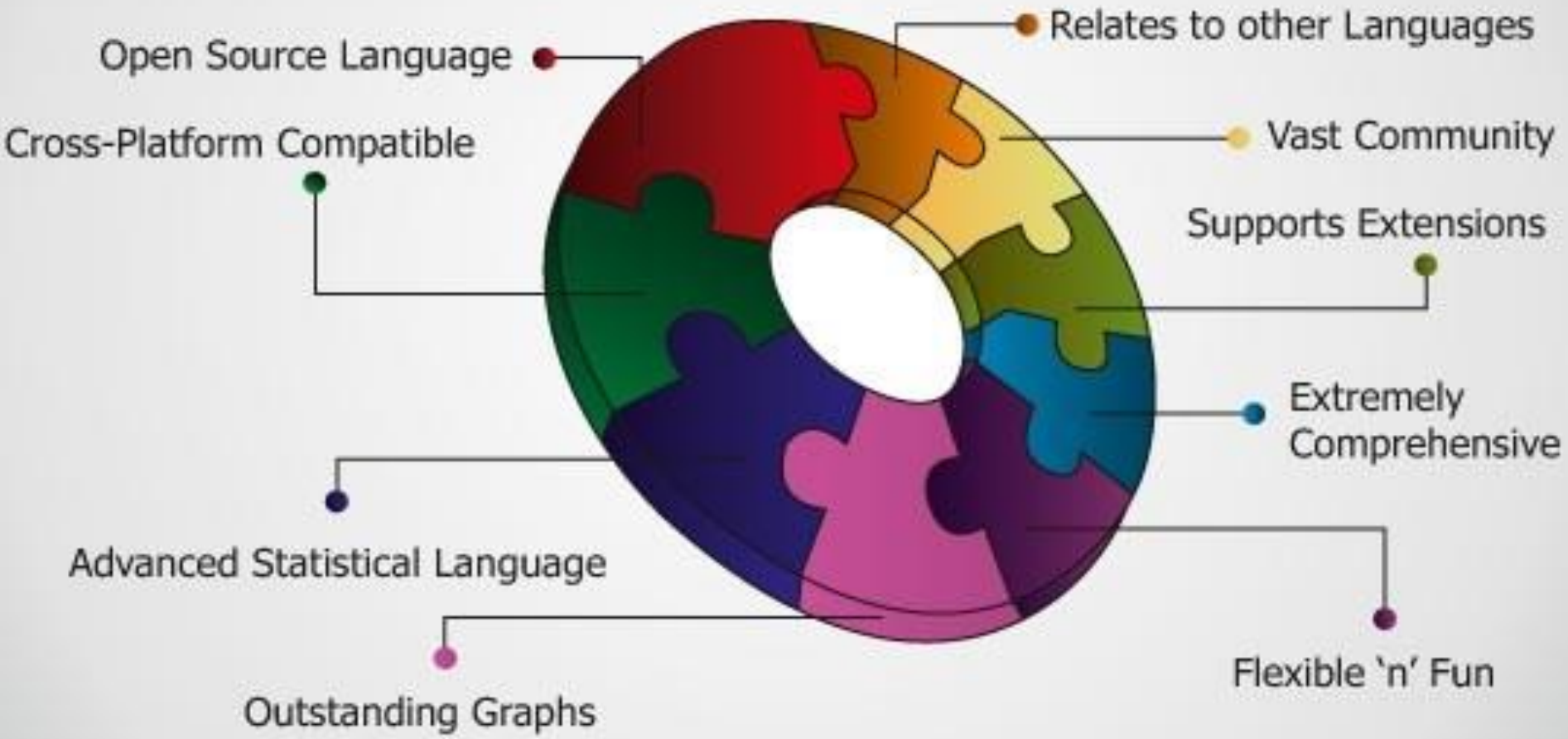
Objective

- ✓ Sometimes variables in their raw form may not be very effective if directly used as inputs to the analysis algorithm. A way of strengthening the analysis is to derive variables that are custom made for the analysis, using data and business domain knowledge.
- ✓ Portfolio of transformations, usually applied to predictors in modeling, using domain knowledge and business context is called variable customization.



Introduction to R-Statistical Programming

Why Learn R?


Open Source Language
Relates to other Languages
Cross-Platform Compatible
Vast Community
Supports Extensions
Extremely Comprehensive
Advanced Statistical Language
Flexible 'n' Fun
Outstanding Graphs

What is R?

R is one of the most powerful machine learning platforms and is used by the top data scientists in the world.

R is a language, an interpreter and a platform.

- R is used by the best data scientists in the world.
- R is powerful because of the breadth of techniques it offers in third-party packages.
- R is state-of-the-art because it is used by academics.
- R is free because it is open source software.
- R is a lot of fun.

What is R?

R is a computer language . It can be difficult to learn but is familiar and you will figure it out quickly if you have used other scripting languages like Python, Ruby or BASH.

R is an interpreter . You can write scripts and save them as files. Like other scripting languages, you can then use the interpreter to run those scripts any time. R also provides a REPL environment where you can type in commands and see the output immediately.

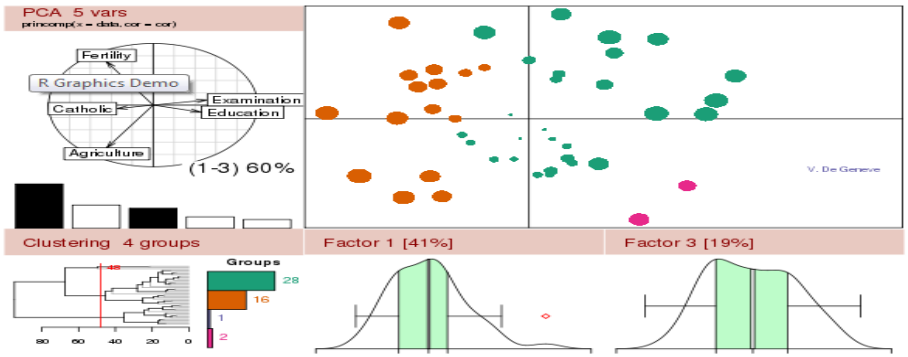
R is also a platform . You can use it to create and display graphics, to save and load state and to interface with other systems. You can do all of your exploration and development in the REPL (read-evaluate-print loop) environment if you so wish.

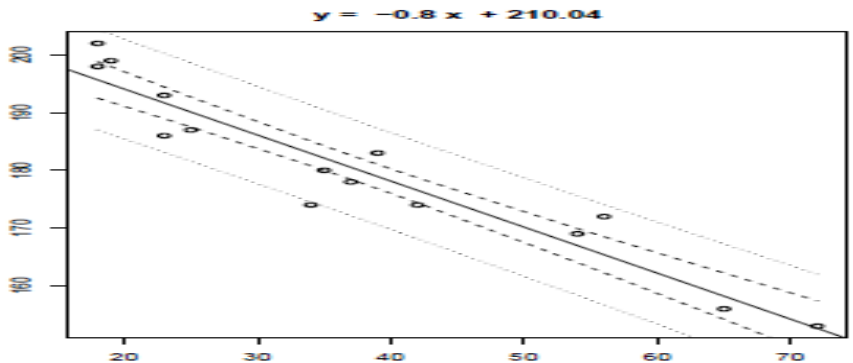
What is R – Statistics?

Hypothesis Testing

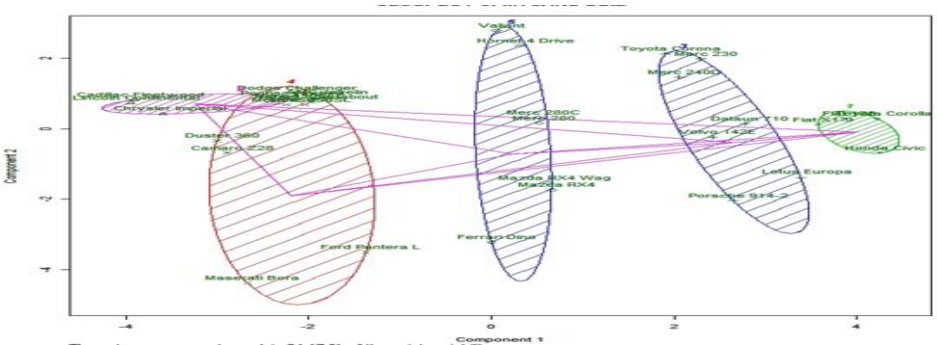
```
        One Sample t-test

data:  x
t = 283.8161, df = 9, p-value = < 2.2e-16
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 173.3076 176.0924
sample estimates:
mean of x
    174.7
```

Powerful graphs using R





Linear Regression



Clustering

R Programming – Reserved keywords

- Reserved words in R programming are a set of words that have special meaning and cannot be used as an identifier (variable name, function name etc).
- Here is a list of reserved words in the R's parser.

Reserved words in R

if	else	repeat	while	function
for	in	next	break	TRUE
FALSE	NULL	Inf	NaN	NA
NA_integer_	NA_real_	NA_complex_	NA_character_	

R Programming – Reserved keywords

- Among these words, **if**, **else**, **repeat**, **while**, **function**, **for**, **in**, **next** and **break** are used for conditions, loops and user defined functions. They form the basic building blocks of programming in R.
- **TRUE** and **FALSE** are the logical constants in R.
- **NULL** represents the absence of a value or an undefined value.
- **Inf** is for "Infinity", for example when 1 is divided by 0 whereas **NaN** is for "Not a Number", for example when 0 is divided by 0.

R Programming – Reserved keywords

- **NA** stands for "Not Available" and is used to represent missing values.
- R is a case sensitive language. Which mean that **TRUE** and **True** are not the same.
- While the first one is a reserved word denoting a logical constant in R, the latter can be used a variable name.

```
> TRUE <- 1
Error in TRUE <- 1 : invalid (do_set) left-hand side to assignment

> True <- 1

> TRUE
[1] TRUE

> True
[1] 1
```

R Programming – Variables in R

- Variables are used to store data, whose value can be changed according to our need. Unique name given to variable (function and objects as well) is identifier.
- Rules for writing Identifiers in R.
  - Identifiers can be a combination of letters, digits, period (.) and underscore (_).
  - It must start with a letter or a period. If it starts with a period, it cannot be followed by a digit.
  - Reserved words in R cannot be used as identifiers.
- Valid identifiers in R
  - **total, Sum, .fine.with.dot, this_is_acceptable, Number5**
- Invalid identifiers in R
  - **tot@l, 5um, _fine, TRUE, .0ne**

R Programming – R Operators

- R has many operators to carry out different mathematical and logical operations.
- Operators in R can mainly be classified into the following categories.
- Type of operators in R
  - Arithmetic operators
  - Relational operators
  - Logical operators
  - Assignment operators
- **R Arithmetic Operators**
- These operators are used to carry out mathematical operations like addition and multiplication. Here is a list of arithmetic operators available in R.

R Programming – R Operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
^	Exponent
%%	Modulus (Remainder from division)
%/%	Integer Division

```
> x <- 5
> y <- 16

> x+y
[1] 21

> x-y
[1] -11

> x*y
[1] 80

> y/x
[1] 3.2

> y%/%x
[1] 3

> y%%x
[1] 1

> y^x
[1] 1048576
```

R Programming – R Relational Operators

- Relational operators are used to compare between values. Here is a list of relational operators available in R.

Relational Operators in R

Operator	Description
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
!=	Not equal to

R Programming – R Relational Operators

Example

```
> x <- 5
> y <- 16

> x<y
[1] TRUE

> x>y
[1] FALSE

> x<=5
[1] TRUE

> y>=20
[1] FALSE

> y == 16
[1] TRUE

> x != 5
[1] FALSE
```

R Programming – R Vectors

- Vector is a basic data structure in R. It contains element of the same type. The data types can be logical, integer, double, character, complex or raw.
- A vector's type can be checked with the **typeof()** function.
- Another important property of a vector is its length. This is the number of elements in the vector and can be checked with the function **length()**.

How to create a Vector in R programming?

- Vectors are generally created using the **c()** function.
- Since, a vector must have elements of the same type, this function will try and coerce elements to the same type, if they are different.

R Programming – Operation on Vectors

Coercion is from lower to higher types from logical to integer to double to character.

```
> x <- c(1, 5, 4, 9, 0)
> typeof(x)
[1] "double"
> length(x)
[1] 5

> x <- c(1, 5.4, TRUE, "hello")
> x
[1] "1"     "5.4"   "TRUE"  "hello"
> typeof(x)
[1] "character"
```

If we want to create a vector of consecutive numbers, the : operator is very helpful.

Example 1: Creating a vector using : operator

```
> x <- 1:7; x
[1] 1 2 3 4 5 6 7

> y <- 2:-2; y
[1]  2  1  0 -1 -2
```

More complex sequences can be created using the seq() function, like defining number of points in an interval, or the step size.

R Programming – Operation on Vectors

Example 2: Creating a vector using seq() function

```
> seq(1, 3, by=0.2)           # specify step size
[1] 1.0 1.2 1.4 1.6 1.8 2.0 2.2 2.4 2.6 2.8 3.0

> seq(1, 5, length.out=4)     # specify length of the vector
[1] 1.000000 2.333333 3.666667 5.000000
```

How to access Elements of a Vector?

Elements of a vector can be accessed using vector indexing. The vector used for indexing can be logical, integer or character vector.

R Programming – Operation on Vectors

Using integer vector as index

- Vector index in R starts from 1, unlike most programming languages where index start from 0.
- We can use a vector of integers as index to access specific elements.
- We can also use negative integers to return all elements except that those specified.
- But we cannot mix positive and negative integers while indexing and real numbers, if used, are truncated to integers.

```
> x
[1]  0  2  4  6  8 10

> x[3]            # access 3rd element
[1] 4

> x[c(2, 4)]      # access 2nd and 4th element
[1] 2 6

> x[-1]           # access all but 1st element
[1]  2  4  6  8 10

> x[c(2, -4)]     # cannot mix positive and negative integers
Error in x[c(2, -4)] : only 0's may be mixed with negative subscripts

> x[c(2.4, 3.54)]    # real numbers are truncated to integers
[1] 2 4
```

R Programming – Operation on Vectors

How to modify a vector in R?

- ✓ We can modify a vector using the assignment operator.
- ✓ We can use the techniques discussed above to access specific elements and modify them.
- ✓ If we want to truncate the elements, we can use reassignments.

```
> x
[1] -3 -2 -1  0  1  2

> x[2] <- 0; x         # modify 2nd element
[1] -3  0 -1  0  1  2

> x[x<0] <- 5; x   # modify elements less than 0
[1] 5 0 5 0 1 2

> x <- x[1:4]; x       # truncate x to first 4 elements
[1] 5 0 5 0
```

How to delete a Vector?

We can delete a vector by simply assigning a NULL to it.

```
> x
[1] -3 -2 -1  0  1  2
> x <- NULL

> x
NULL
> x[4]
NULL
```

R Programming – Operation on Vectors

- When there is a mismatch in length (number of elements) of operand vectors, the elements in shorter one is recycled in a cyclic manner to match the length of the longer one.
- R will issue a warning if the length of the longer vector is not an integral multiple of the shorter vector.

```
> x <- c(2,1,8,3)
> y <- c(9,4)

> x+y # Element of y is recycled to 9,4,9,4
[1] 11  5 17  7

> x-1 # Scalar 1 is recycled to 1,1,1,1
[1] 1 0 7 2

> x+c(1,2,3)
[1]  3  3 11  4
Warning message:
In x + c(1, 2, 3) :
 longer object length is not a multiple of shorter object length
```

R Programming – R Matrix

- Matrix is a two dimensional data structure in R programming.
- Matrix is similar to vectors but additionally contains the dimension attribute. All attributes of an object can be checked with the attributes() function (dimension can be checked directly with the dim() function).
- We can check if a variable is a matrix or not with the class() function.
- Let us consider a matrix as follows.

```
> a
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9

> class(a)
[1] "matrix"

> attributes(a)
$dim
[1] 3 3

> dim(a)
[1] 3 3
```

R Programming – R Matrix

How to create a matrix in R programming?

- Matrix can be created using the **matrix()** function.
- Dimension of the matrix can be defined by passing appropriate value for arguments **nrow** and **ncol**.
- Providing value for both dimension is not necessary. If one of the dimension is provided, the

```
> matrix(1:9, nrow = 3, ncol = 3)
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9

> # same result is obtained by providing only one dimension
> matrix(1:9, nrow = 3)
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

R Programming – R Matrix

We can see that the matrix is filled column-wise. This can be reversed to row-wise filling by passing **TRUE** to the argument **byrow**.

```
> matrix(1:9, nrow=3, byrow=TRUE)    # fill matrix row-wise
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

We can see that the matrix is filled column-wise. This can be reversed to row-wise filling by passing **TRUE** to the argument **byrow**.

```
> matrix(1:9, nrow=3, byrow=TRUE)    # fill matrix row-wise
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

R Programming – R Matrix

- In all cases, however, a matrix is stored in column-major order internally as we will see in the subsequent sections.
- It is possible to name the rows and columns of matrix during creation by passing a 2 element list to the argument **dimnames**.

```
> x <- matrix(1:9, nrow = 3, dimnames = list(c("X","Y","Z"), c("A","B","C")))
> x
 A B C
X 1 4 7
Y 2 5 8
Z 3 6 9
```

These names can be accessed or changed with two helpful functions **colnames()** and **rownames()**.

```
> colnames(x)
[1] "A" "B" "C"
> rownames(x)
[1] "X" "Y" "Z"

> # It is also possible to change names
> colnames(x) <- c("C1","C2","C3")
> rownames(x) <- c("R1","R2","R3")

> x
   C1 C2 C3
R1  1  4  7
R2  2  5  8
R3  3  6  9
```

R Programming – R Matrix

- Another way of creating a matrix is by using functions **cbind()** and **rbind()** as in column bind and row bind.

```
> cbind(c(1,2,3),c(4,5,6))
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6

> rbind(c(1,2,3),c(4,5,6))
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

Finally, you can also create a matrix from a vector by setting its dimension using **dim()**.

```
> x <- c(1,2,3,4,5,6)
> x
[1] 1 2 3 4 5 6
> class(x)
[1] "numeric"

> dim(x) <- c(2,3)
> x
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> class(x)
[1] "matrix"
```

R Programming – R Matrix

How to access Elements of a matrix?

We can access elements of a matrix using the square bracket [indexing method. Elements can be accessed as var[row, column]. Here rows and columns are vectors.

How to modify a matrix in R?

We can combine assignment operator with the above learned methods for accessing elements of a matrix to modify it.

```
> x
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9

> x[2,2] <- 10; x    # modify a single element
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2   10    8
[3,]    3    6    9

> x[x<5] <- 0; x    # modify elements less than 5
     [,1] [,2] [,3]
[1,]    0    0    7
[2,]    0   10    8
[3,]    0    6    9
```

R Programming – R Matrix

A common operation with matrix is to transpose it. This can be done with the function **t().**

```
> t(x)    # transpose a matrix
     [,1] [,2] [,3]
[1,]    0    0    0
[2,]    0   10    6
[3,]    7    8    9
```

We can add row or column using **rbind()** and **cbind()** function respectively. Similarly, it can be removed through reassignment.

```
> cbind(x, c(1, 2, 3))    # add column
     [,1] [,2] [,3] [,4]
[1,]    0    0    7    1
[2,]    0   10    8    2
[3,]    0    6    9    3

> rbind(x,c(1,2,3))    # add row
     [,1] [,2] [,3]
[1,]    0    0    7
[2,]    0   10    8
[3,]    0    6    9
[4,]    1    2    3

> x <- x[1:2,]; x    # remove last row
     [,1] [,2] [,3]
[1,]    0    0    7
[2,]    0   10    8
```

R Programming – R Matrix

Dimension of matrix can be modified as well, using the **dim()** function.

```
> x
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

> dim(x) <- c(3,2); x     # change to 3X2 matrix
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6

> dim(x) <- c(1,6); x     # change to 1X6 matrix
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    2    3    4    5    6
```

R Programming – R Data Frame

Data frame is a two dimensional data structure in R. It is a special case of a list which has each component of equal length.

Each component form the column and contents of the component form the rows.

Check if a variable is a data frame using class()

```
> x
  SN Age Name
1  1  21 John
2  2  15 Dora

> typeof(x)     # data frame is a special case of  list
[1] "list"

> class(x)
[1] "data.frame"
```

In this example, x can be considered as a list of 3 components with each component having a two element vector. Some useful functions to know more about a data frame are given below.

R Programming – R Data Frame

Functions of data frame

```
> names(x)
[1] "SN"   "Age"  "Name"

> ncol(x)
[1] 3

> nrow(x)
[1] 2

> length(x)    # returns length of the list, same as ncol()
[1] 3
```

How to create a Data Frame in R?

We can create a data frame using the **data.frame()** function.

For example, the above shown data frame can be created as follows.

```
> x <- data.frame("SN" = 1:2, "Age" = c(21,15), "Name" = c("John","Dora"))

> str(x)    # structure of x
'data.frame':   2 obs. of  3 variables:
$ SN  : int  1 2
$ Age : num  21 15
$ Name: Factor w/ 2 levels "Dora","John": 2 1
```

R Programming – R Data Frame

Notice above that the third column, **Name** is of type factor, instead of a character vector.

By default, **data.frame()** function converts character vector into factor.

To suppress this behavior, we can pass the argument **stringsAsFactors=FALSE**.

```
> x <- data.frame("SN" = 1:2, "Age" = c(21,15), "Name" = c("John", "Dora"), stringsAsFactors =

> str(x)    # now the third column is a character vector
'data.frame':   2 obs. of  3 variables:
$ SN  : int  1 2
$ Age : num  21 15
$ Name: chr  "John" "Dora"
```

Many data input functions of R like, **read.table(), read.csv(), read.delim(), read.fwf()** also read data into a data frame.

R Programming – R Data Frame

How to modify a Data Frame in R?

Data frames can be modified like we modified matrices through reassignment.

```
> x
 SN Age Name
1  1  21 John
2  2  15 Dora

> x[1,"Age"] <- 20; x
 SN Age Name
1  1  20 John
2  2  15 Dora
```

Adding Components

Rows can be added to a data frame using the **rbind()** function.

```
> rbind(x,list(1,16,"Paul"))
 SN Age Name
1  1  20 John
2  2  15 Dora
3  1  16 Paul
```

R Programming – R Data Frame

Similarly, we can add columns using **cbind()**.

```
> cbind(x,State=c("NY","FL"))
 SN Age Name State
1  1  20 John    NY
2  2  15 Dora    FL
```

Since data frames are implemented as list, we can also add new columns through simple list-like assignments.

```
> x
 SN Age Name
1  1  20 John
2  2  15 Dora

> x$State <- c("NY","FL"); x
 SN Age Name State
1  1  20 John    NY
2  2  15 Dora    FL
```

R Programming – R Data Frame

Deleting Component

Data frame columns can be deleted by assigning **NULL** to it.

```
> x$State <- NULL
> x
 SN Age Name
1  1  20 John
2  2  15 Dora
```

Similarly, rows can be deleted through reassignments.

```
> x <- x[-1,]
> x
 SN Age Name
2  2  15 Dora
```

R Programming – R Factors

- Factor is a data structure used for fields that takes only predefined, finite number of values (categorical data).
- For example, a data field such as marital status may contain only values from single, married, separated, divorced, or widowed.
- In such case, we know the possible values beforehand and these predefined, distinct values are

```
> x
[1] single  married married single
Levels: married single
```

Here, we can see that factor x has four elements and two levels. We can check if a variable is a factor or not using **class()** function.
Similarly, levels of a factor can be checked using the **levels()** function.

```
> class(x)
[1] "factor"

> levels(x)
[1] "married" "single"
```

R Programming – R Factors

How to create a factor in R?

We can create a factor using the function **factor().** Levels of a factor are inferred from the data if not provided.

```
> x <- factor(c("single", "married", "married", "single"));
> x
[1] single  married married single
Levels: married single


> x <- factor(c("single", "married", "married", "single"), levels =
c("single", "married", "divorced"));
> x
[1] single  married married single
Levels: single married divorced
```

We can see from the above example that levels may be predefined even if not used.

Factors are closely related with vectors. In fact, factors are stored as integer vectors. This is clearly seen from its structure.

```
> x <- factor(c("single","married","married","single"))
> str(x)
Factor w/ 2 levels "married","single": 2 1 1 2
```

R Programming – R Factors

- Factor is a data structure used for fields that takes only predefined, finite number of values (categorical data).
- For example, a data field such as marital status may contain only values from single, married, separated, divorced, or widowed.
- In such case, we know the possible values beforehand and these predefined, distinct values are called levels. Following is an example of factor in R.

```
> x
[1] single  married married single
Levels: married single
```

Here, we can see that factor x has four elements and two levels. We can check if a variable is a factor or not using **class()** function.

Similarly, levels of a factor can be checked using the **levels()** function.

R Programming – R Factors

```
> x <- factor(c("single","married","married","single"))
> str(x)
Factor w/ 2 levels "married","single": 2 1 1 2
```

- We see that levels are stored in a character vector and the individual elements are actually stored as indices.
- Factors are also created when we read non-numerical columns into a data frame.
- By default, **data.frame()** function converts character vector into factor. To suppress this behavior, we have to pass the argument **stringsAsFactors = FALSE**.

R Programming – R Factors

How to access components of a factor?

Accessing components of a factor is very much similar to that of vectors.

```
> x
[1] single  married married single
Levels: married single

> x[3]            # access 3rd element
[1] married
Levels: married single

>  x[c(2, 4)]     # access 2nd and 4th element
[1] married single
Levels: married single

> x[-1]           # access all but 1st element
[1] married married single
Levels: married single

> x[c(TRUE, FALSE, FALSE, TRUE)]  # using logical vector
[1] single single
Levels: married single
```

R Programming – R Factors

How to modify a factor?

Components of a factor can be modified using simple assignments. However, we cannot choose values outside of its predefined levels.

```
> x
[1] single  married married single
Levels: single married divorced

> x[2] <- "divorced"    # modify second element;  x
[1] single   divorced married  single
Levels: single married divorced

> x[3] <- "widowed"    # cannot assign values outside levels
Warning message:
In `[<-.factor`(`*tmp*`, 3, value = "widowed") :
 invalid factor level, NA generated

> x
[1] single   divorced <NA>     single
Levels: single married divorced
```

R Programming – R Lists

- List is a data structure having components of mixed data types.
- A vector having all elements of the same type is called atomic vector but a vector having elements of different type is called list.
- We can check if it's a list with **typeof()** function and find its length using **length().**
- Following is an example of a list having three components each of different data type.

```
> x
$a
[1] 2.5

$b
[1] TRUE

$c
[1] 1 2 3

> typeof(x)
[1] "list"

> length(x)
[1] 3
```

R Programming – R Lists

How to access components of a list?

Lists can be accessed in similar fashion to vectors. Integer, logical or character vectors can be used for indexing. Let us consider a list as follows.

```
> x
$name
[1] "John"

$age
[1] 19

$speaks
[1] "English" "French"

> x[c(1:2)]     # index using integer vector
$name
[1] "John"

$age
[1] 19

> x[-2]         # using negative integer to exclude second component
$name
[1] "John"

$speaks
[1] "English" "French"

> x[c(T,F,F)]   # index using logical vector
$name
[1] "John"

> x[c("age","speaks")]     # index using character vector
$age
[1] 19

$speaks
[1] "English" "French"
```

R Programming – R Lists

How to modify a list in R?

We can change components of a list through reassignment. We can choose any of the component accessing techniques discussed above to modify it.

Notice below that modification causes reordering of components.

```
> x[["name"]] <- "Clair"; x
$age
[1] 19

$speaks
[1] "English" "French"

$name
[1] "Clair"
```

R Programming – R Lists

How to add components to a list?

Adding new components is easy. We simply assign values using new tags and it will pop into action.

```
> x[["married"]] <- FALSE
> x
$age
[1] 19

$speaks
[1] "English" "French"

$name
[1] "Clair"

$married
[1] FALSE
```

R Programming – Useful functions

Function	Description
ls()	List of Data objects in memory
rm(object)	Remove an object from memory
length(object)	Number of elements in a vector
str(object)	Details about structure of an object
class(object)	What type of object
dim(object)	Number of rows and columns in a data frame or a matrix
data()	List of data sets available in different loaded packages

R Programming – Useful functions

Function	Description
mean (x)	Average of vector of numbers
var(x), sd(x)	Variance and Standard Deviation
median(x)	Median
range(x)	Minimum and Maximum numbers
sum(x)	Sum of a vector of numbers
min(x), max(x)	Minimum, Maximum
quantile(x, prob)	Values of x at different percentiles

R Programming – Loop functions

Looping on the Command Line

Writing for and while loops is useful when programming but not particularly easy when working interactively on the command line.

Multi-line expressions with curly braces are just not that easy to sort through when working on the command line.

R has some functions which implement looping in a compact form to make your life easier.

- lapply(): Loop over a list and evaluate a function on each element
- sapply(): Same as lapply but try to simplify the result
- apply(): Apply a function over the margins of an array
- tapply(): Apply a function over subsets of a vector
- mapply(): Multivariate version of lapply

An auxiliary function split is also useful, particularly in conjunction with lapply.

R Programming – apply()

The apply() Family

The apply() family pertains to the R base package and is populated with functions to manipulate slices of data from matrices, arrays, lists and dataframes in a repetitive way.
These functions allow crossing the data in a number of ways and avoid explicit use of loop constructs.
They act on an input list, matrix or array and apply a named function with one or several optional arguments.
The called function could be:

- An aggregating function, like for example the mean, or the sum (that return a number or scalar);
- Other transforming or subsetting functions; and
- Other vectorized functions, which return more complex structures like lists, vectors, matrices and arrays.

R Programming – apply()

The apply() functions form the basis of more complex combinations and helps to perform operations with very few lines of code. More specifically, the family is made up of the **apply(), lapply() , sapply(), vapply(), mapply(), rapply(),** and **tapply()** functions.

How To Use apply() in R
apply(X, MARGIN, FUN, ...)
where:

- X is an array or a matrix if the dimension of the array is 2;
- MARGIN is a variable defining how the function is applied: when MARGIN=1, it applies over rows, whereas with MARGIN=2, it works over columns. Note that when you use the construct MARGIN=c(1,2), it applies to both rows and columns; and
- FUN, which is the function that you want to apply to the data. It can be any R function, including a User Defined Function (UDF).

R Programming – lapply()

Looping on the Command Line

Writing for and while loops is useful when programming but not particularly easy when working interactively on the command line.

Multi-line expressions with curly braces are just not that easy to sort through when working on the command line.

R has some functions which implement looping in a compact form to make your life easier.

- lapply(): Loop over a list and evaluate a function on each element
- sapply(): Same as lapply but try to simplify the result
- apply(): Apply a function over the margins of an array
- tapply(): Apply a function over subsets of a vector
- mapply(): Multivariate version of lapply

An auxiliary function split is also useful, particularly in conjunction with lapply.

R Programming – sapply()

The sapply() function behaves similarly to lapply(); the only real difference is in the return value.

sapply() will try to simplify the result of lapply() if possible. Essentially, sapply() calls lapply()

on its input and then applies the following algorithm:

- If the result is a list where every element is length 1, then a vector is returned
- If the result is a list where every element is a vector of the same length (> 1), a matrix is returned.
- If it can’t figure things out, a list is returned

R Programming – split()

The split() function takes a vector or other objects and splits it into groups determined by a factor or list of factors.

The arguments to split() are

```
> str(split)
```

```
function (x, f, drop = FALSE, ...)
```

where

- x is a vector (or list) or data frame.
- f is a factor (or coerced to one) or a list of factors
- drop indicates whether empty factors levels should be dropped

R Programming – tapply()

tapply() is used to apply a function over subsets of a vector. It can be thought of as a combination of split() and sapply() for vectors only. I've been told that the "t" in tapply() refers to "table", but that is unconfirmed.

> str(tapply)

function (X, INDEX, FUN = NULL, ..., simplify = TRUE)

The arguments to tapply() are as follows:

- X is a vector
- INDEX is a factor or a list of factors (or else they are coerced to factors)
- FUN is a function to be applied
- … contains other arguments to be passed FUN
- simplify, should we simplify the result?

R Programming – Col/Row Sums and Means

For the special case of column/row sums and column/row means of matrices, we have some useful shortcuts.

- rowSums = apply(x, 1, sum)
- rowMeans = apply(x, 1, mean)
- colSums = apply(x, 2, sum)
- colMeans = apply(x, 2, mean)

The shortcut functions are heavily optimized and hence are much faster, but you probably won’t notice unless you’re using a large matrix. Another nice aspect of these functions is that they are a bit more descriptive. It’s arguably more clear to write colMeans(x) in your code than apply(x, 2, mean).

R Programming – mapply()

The mapply() function is a multivariate apply of sorts which applies a function in parallel over a set of arguments. Recall that lapply() and friends only iterate over a single R object. What if you want to iterate over multiple R objects in parallel? This is what mapply() is for.

```
> str(mapply)
```

```
function (FUN, ..., MoreArgs = NULL, SIMPLIFY = TRUE, USE.NAMES = TRUE)
```

The arguments to mapply() are

- FUN is a function to apply
- ... contains R objects to apply over
- MoreArgs is a list of other arguments to FUN.
- SIMPLIFY indicates whether the result should be simplified

Exploratory Data Analysis

Creating standard data summaries

1. Read the data from auto-mpg.csv and convert cylinders to factor:

- auto <- read.csv("auto-mpg.csv", header = TRUE, stringsAsFactors = FALSE)
- # Convert cylinders to factor
- auto$cylinders <- factor(auto$cylinders, levels = c(3,4,5,6,8), labels = c("3cyl", "4cyl", "5cyl", "6cyl", "8cyl"))

2. Get the summary statistics:

- **summary(auto)**
- The summary() function gives a "six number" summary for numerical variables—minimum, first quartile, median, mean, third quartile, and maximum. For factors (or categorical variables), the function shows the counts for each level; for character variables, it just shows the total number of available values.

Creating standard data summaries

3.Using the str() function for an overview of a data frame :

- **str(auto)**
- The str() function gives a concise view into a data frame. In fact, we can use it to see the underlying structure of any arbitrary R object.

4. Computing the summary for a single variable :

- **summary(auto$cylinders)**
- **summary(auto$mpg)**
- The summary you get for numerical variables remains as before, but for factors, you get counts for many more levels.

5.Finding the mean and standard deviation :

Use the functions mean() and sd() as follows:

- **mean(auto$mpg)**
- **sd(auto$mpg)**

Extracting a subset of a dataset

The following steps extract a subset of a dataset:

1.Index by position.

- Get model_year and car_name for the first three cars:
- auto[1:3, 8:9]
- auto[1:3, c(8,9)]

2. Index by name.

- Get model_year and car_name for the first three cars:
- auto[1:3,c("model_year", "car_name")]

3.Retrieve all details for cars with the highest or lowest mpg, using the following code:

- auto[auto$mpg == max(auto$mpg) | auto$mpg == min(auto$mpg),]

Extracting a subset of a dataset

3. Excluding columns

Use the minus sign for variable positions that you want to exclude from the subset. Also, you cannot mix both positive and negative indexes in the list. Both of the following approaches are correct:

- **auto[,c(-1,-9)]**
- **auto[,-c(1,9)]**

However, this subsetting approach does not work while specifying variables using names. For example, we cannot use –c("No", "car_name"). Instead, use %in% with ! (negation) to exclude variables:

- **auto[, !names(auto) %in% c("No", "car_name")]**

Extracting a subset of a dataset

4. Selecting based on multiple values

Select all cars with mpg = 15 or mpg = 20:

- **auto[auto$mpg %in% c(15,20),c("car_name","mpg")]**

5. Selecting using logical vector

You can specify the cases (rows) and variables you want to retrieve using boolean vectors. In the following example, R returns the first and second cases, and for each, we get the third variable alone. R returns the elements corresponding to TRUE:

- **auto[1:2,c(FALSE,FALSE,TRUE)]**

Splitting a dataset

- The split function divides data into groups based on a factor or vector. The unsplit() function reverses the effect of split.
  - **carslist <- split(auto, auto$cylinders)**
- The split(auto, auto$cylinders) function returns a list of data frames with each data frame corresponding to the cases for a particular level of cylinders. To reference a data frame from the list, use the [notation.

Visualization

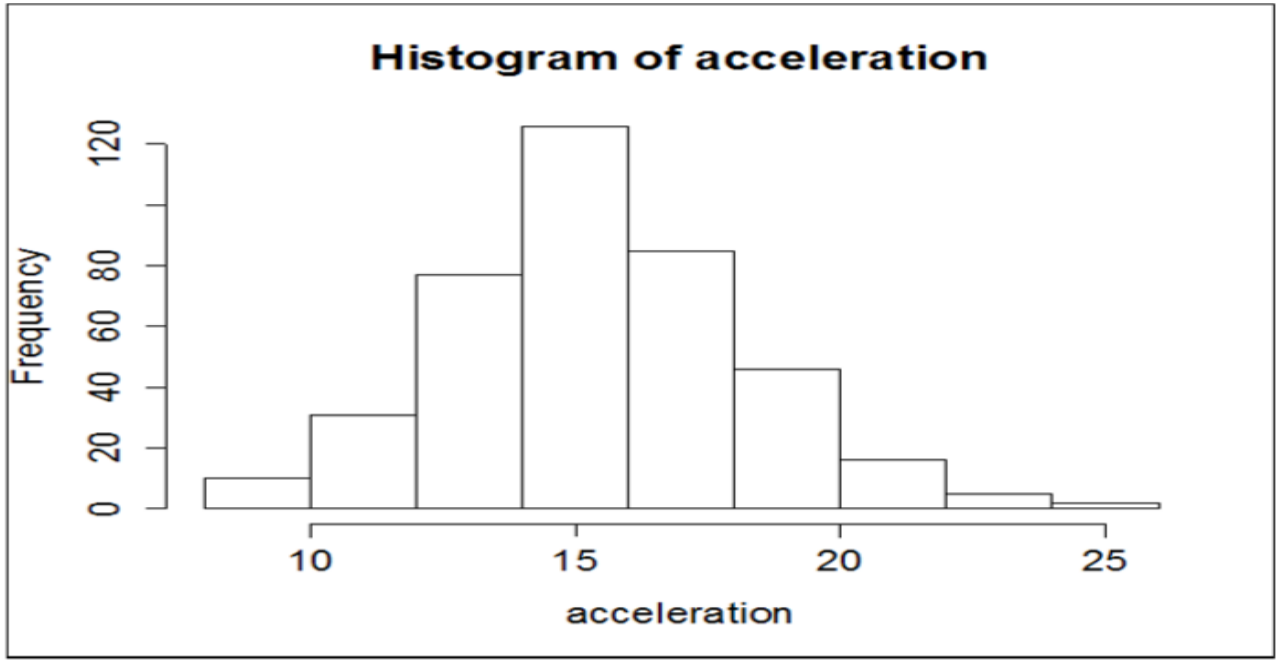
Visualize Your Data

- Visualization means creating charts and plots from the raw data.
- Plots of the distribution or spread of attributes can help you spot outliers, strange or invalid data and give you an idea of possible data transformations you could apply.
  - ✓**Visualization Packages:** A quick note about your options when it comes to R packages for visualization.
  - ✓**Univariate Visualization:** Plots you can use to understand each attribute standalone.
  - ✓**Multivariate Visualization:** Plots that can help you to better understand the interactions between attributes.

Histogram

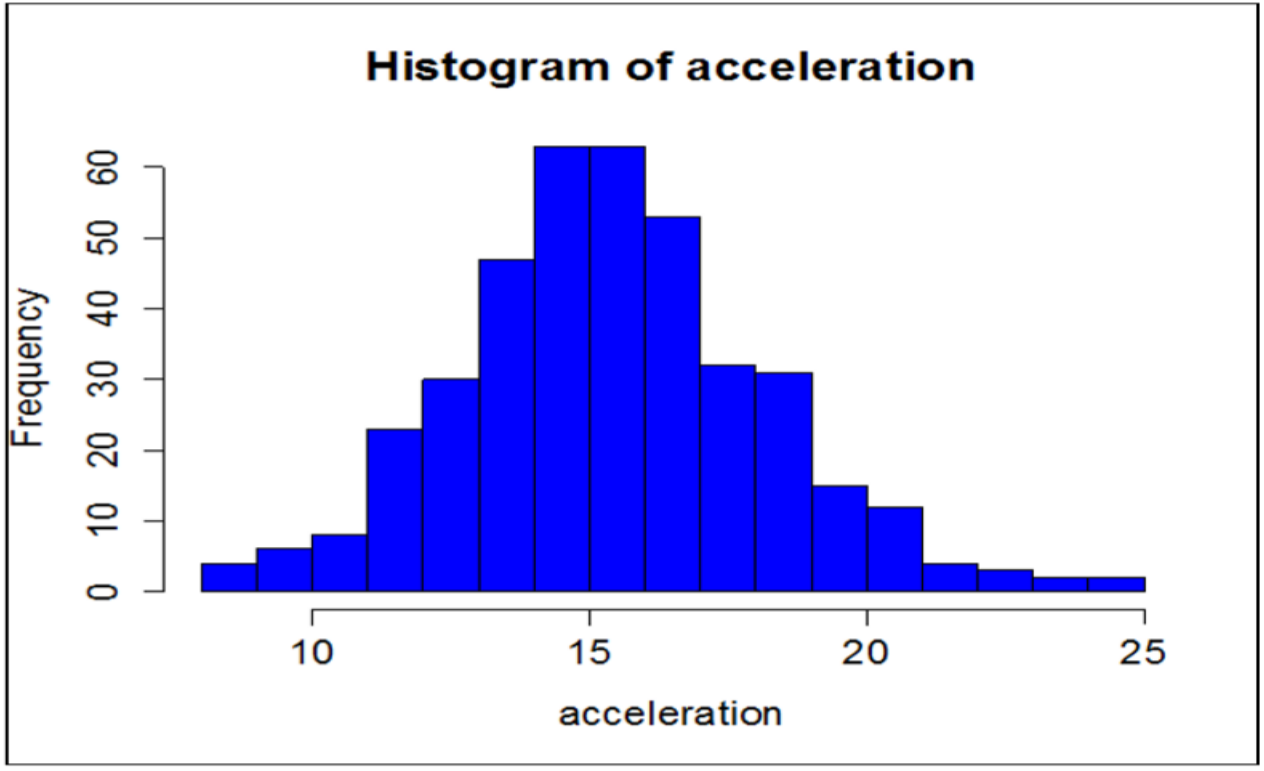
✓ **hist(acceleration)**

- R determines the various properties of the generated graph (like bin sizes, axes scales, axes titles, chart title, bar colors,…) automatically. The following diagram shows the output of the preceding command:



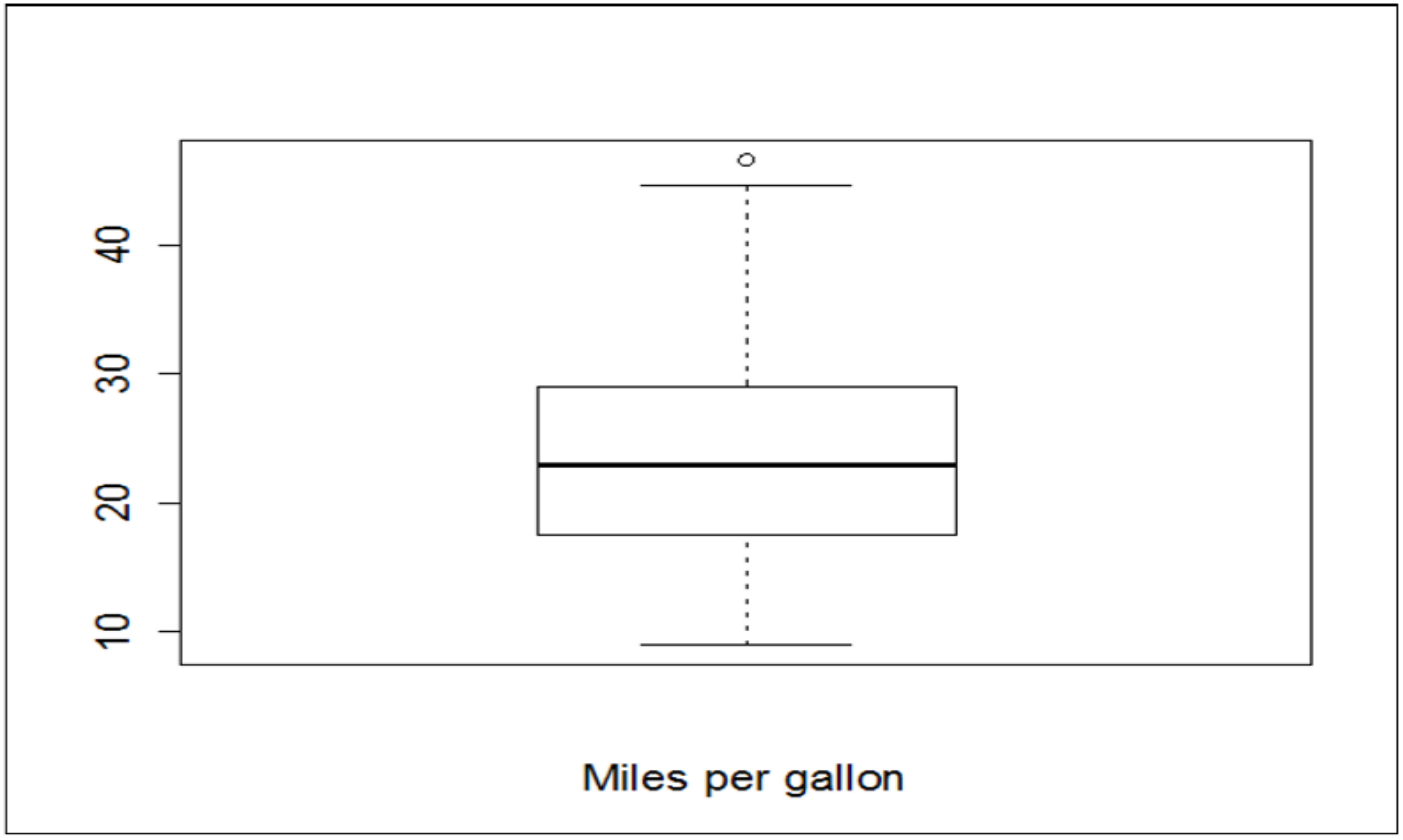
Histogram

- hist(acceleration, col="blue", xlab = "acceleration", main = "Histogram of acceleration", breaks = 15)



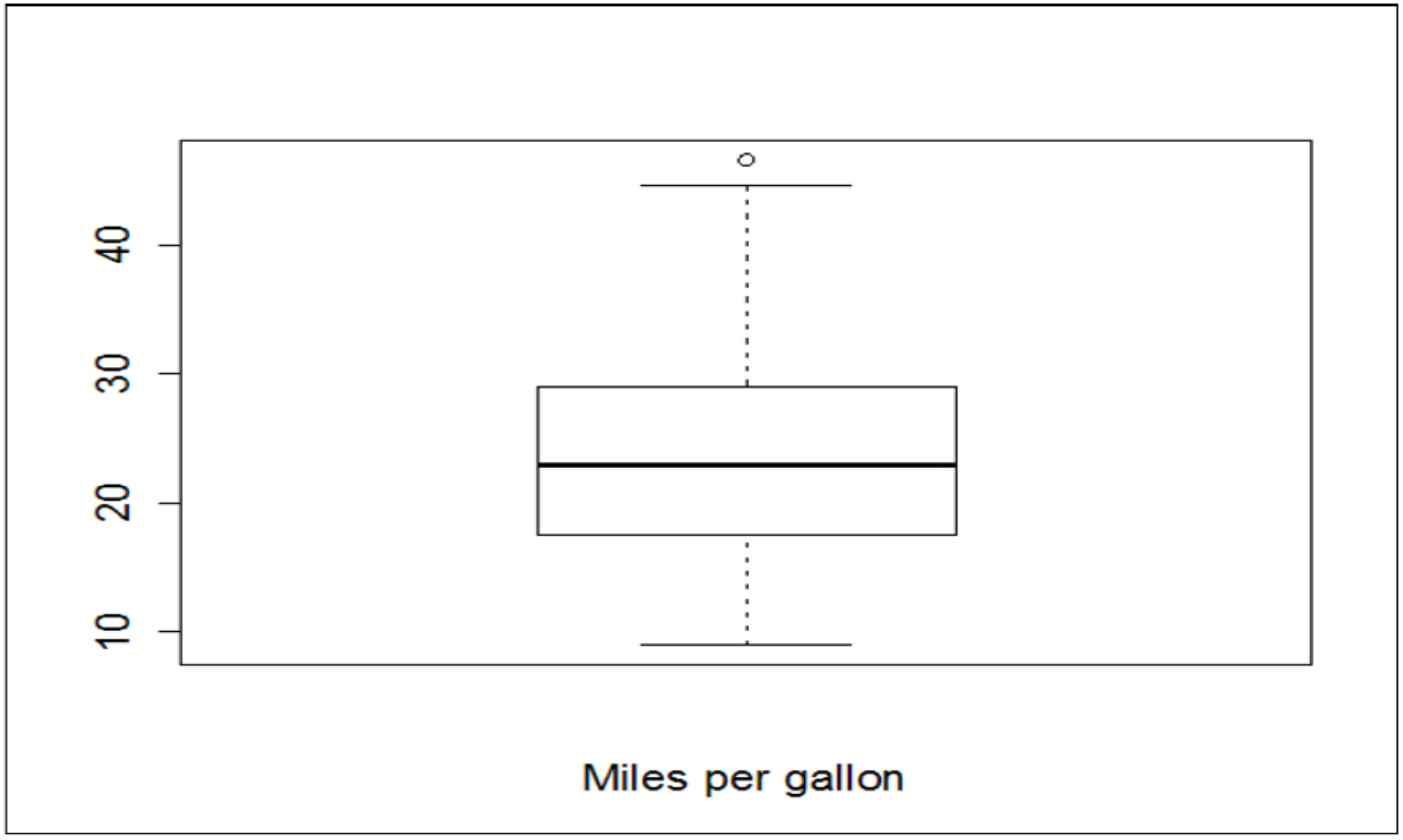
Boxplots

- ✓ **boxplot(mpg, xlab = "Miles per gallon")**



Boxplots

- ✓ **boxplot(mpg, xlab = "Miles per gallon")**



Machine Learning Vs. Statistics

ML vs Statistics

	MACHINE LEARNERS	STATISTICIANS
Network/Graphs vs. Models	Network/Graphs to train and test data	Models to create predictive power
Weights vs. Parameters	Weights used to maximize accuracy scoring and hand tuning	Parameters used to interpret real-world phenomena – stress on magnitude
Confidence Interval	There is no notion of uncertainty	Capturing the variability and uncertainty of parameters
Assumptions	No prior assumption (we learn from the data)	Explicit a-priori assumptions
Distribution	Unknown a priori	A-priori well-defined distribution
Fit	Best fit to learning models (generalization)	Fit to the distribution

Feature Selection

Common Data Problems

- **Noise:** random error or variance in a measured variable
  - Extremely high quantity, rate and amount were eliminated from the data.
  - Negative discount values were eliminated from the data.
- Missing data may be due to
  - Equipment malfunction
  - Consistent with other recorded data and thus deleted
  - Data not entered due to misunderstanding
  - Certain data may not be considered important at the time of entry

Not register history or changes of the data

- **Unified date format:**
  - binary, nominal (categorical), ordinal, numeric
  - e.g. “Sep 24, 2003” , 9/24/03, 24.09.03, etc
- Containing discrepancies in names
- Other data problems which requires data cleaning
  - Duplicate records
  - Incomplete data
  - Inconsistent data

Understand Your Data Using Descriptive Statistics

Any machine learning models that you build are only as good as the data that you provide them. The first step in understanding your data is to actually look at some raw values and calculate some basic statistics.

- **Data Cleaning :** You may discover missing or corrupt data and think of various data cleaning operations to perform such as marking or removing bad data and imputing missing data.

- **Data Transforms :** You may discover that some attributes have familiar distributions such as Gaussian or exponential giving you ideas of scaling or other transforms you could apply.

- **Data Modeling :** You may notice properties of the data such as distributions or data types that suggest the use (or to not use) specific machine learning algorithms.

Feature Selection

- Feature selection is also called variable selection or attribute selection.
- It is the automatic selection of attributes in your data (such as columns in tabular data) that are most relevant to the predictive modeling problem you are working on.
- **Feature Selection Algorithms**
  - **Filter Methods**
    - Some examples of some filter methods include the Chi squared test, information gain and correlation coefficient scores.
  - **Wrapper Methods**
    - A predictive model us used to evaluate a combination of features and assign a score based on model accuracy.
  - **Embedded Methods**
    - Examples of regularization algorithms are the LASSO, Elastic Net and Ridge Regression.

Class Distribution

- A frequency distribution is a table or graph that displays the frequency of various outcomes in a sample.
- Each entry in the table contains the frequency or count of the occurrences of values within a particular group or interval, and in this way, the table summarizes the distribution of values in the sample.

Calculate class breakdown

```
# load the packages
library(mlbench)
# load the dataset
data(PimaIndiansDiabetes)
# distribution of class variable
y <- PimaIndiansDiabetes$diabetes
cbind(freq=table(y), percentage=prop.table(table(y))*100)
```

Output class breakdown

```
    freq percentage
neg  500   65.10417
pos  268   34.89583
```

Data Summary

- In descriptive statistics, summary statistics are used to summarize a set of observations, in order to communicate the largest amount of information as simply as possible.
- Numerical attributes are described using the properties:
  - Min
  - 25th percentile
  - Median
  - Mean
  - 75th percentile
  - Max
- The breakdown also includes an indication of the number of missing values for an attribute (marked N/A).

Data Summary

```
# load the iris dataset
data(iris)
# summarize the dataset
summary(iris)
```

Calculate the summary of the attributes.

```
  Sepal.Length   Sepal.Width    Petal.Length   Petal.Width          Species
 Min.   :4.300  Min.   :2.000  Min.   :1.000  Min.   :0.100  setosa    :50
 1st Qu.:5.100  1st Qu.:2.800  1st Qu.:1.600  1st Qu.:0.300  versicolor:50
 Median :5.800  Median :3.000  Median :4.350  Median :1.300  virginica :50
 Mean   :5.843  Mean   :3.057  Mean   :3.758  Mean   :1.199
 3rd Qu.:6.400  3rd Qu.:3.300  3rd Qu.:5.100  3rd Qu.:1.800
 Max.   :7.900  Max.   :4.400  Max.   :6.900  Max.   :2.500
```

Output of the summary of attributes.

Standard Deviation

- The standard deviation(SD) is a measure that is used to quantify the amount of variation or dispersion of a set of data values.
- A low standard deviation indicates that the data points tend to be close to the mean (also called the expected value) of the set, while a high standard deviation indicates that the data points are spread out over a wider range of values.

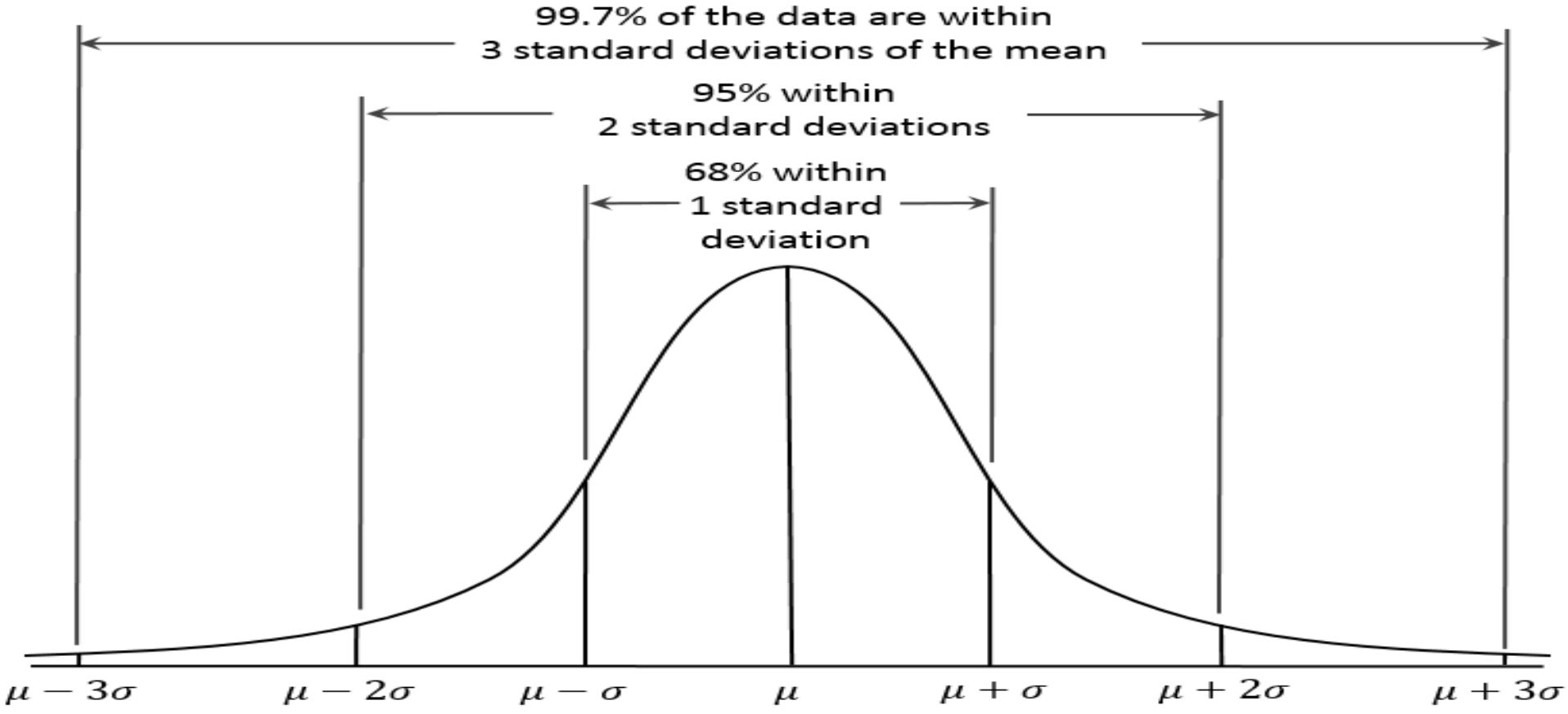
$$s_x = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n - 1}}$$

n = The number of data points

$\bar{x}$ = The mean of the x_i

x_i = Each of the values of the data

Standard Deviation



Standard Deviation

- The standard deviation along with the mean are useful to know if the data has a Gaussian (or nearly Gaussian) distribution.
- It can be useful for a quick and dirty outlier removal tool, where any values that are more than three times the standard deviation from the mean are outside of 99.7% of the data.

```
# load the packages
library(mlbench)
# load the dataset
data(PimaIndiansDiabetes)
# calculate standard deviation for all attributes
sapply(PimaIndiansDiabetes[,1:8], sd)
```

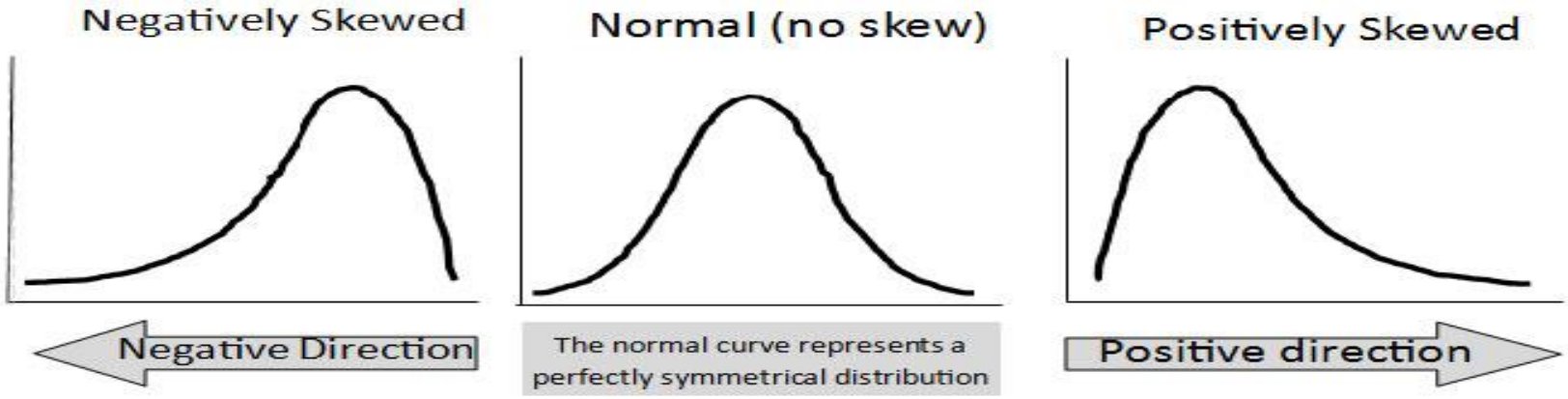
Calculate standard deviations.

This calculates the standard deviation for each numeric attribute in the dataset.

```
  pregnant    glucose   pressure    triceps    insulin       mass   pedigree        age
 3.3695781 31.9726182 19.3558072 15.9522176 115.2440024 7.8841603 0.3313286 11.7602315
```

Skewness

- Skewness is a measure that refers to the extent of symmetry or asymmetry in a distribution.
- Skewness is asymmetry in a statistical distribution, in which the curve appears distorted or skewed, either to the left or to the right.
- Skewness can be quantified to define the extent to which a distribution differs from a normal distribution.



Skewness

- If a distribution looks nearly-Gaussian but is pushed far left or right it is useful to know the skew.
- Getting a feeling for the skew is much easier with plots of the data, such as a histogram or density plot. It is harder to tell from looking at means, standard deviations and quartiles.

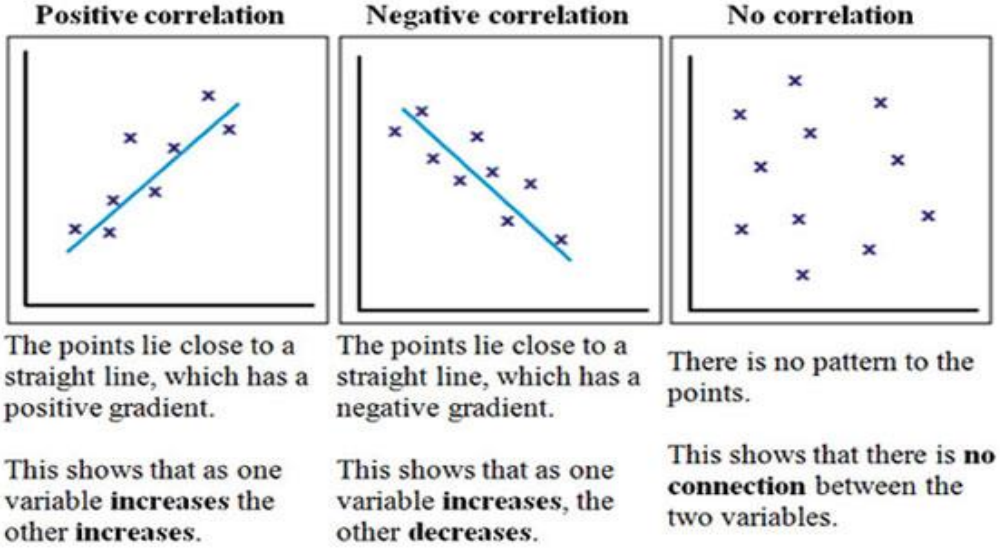
```
# load packages
library(mlbench)
library(e1071)
# load the dataset
data(PimaIndiansDiabetes)
# calculate skewness for each variable
skew <- apply(PimaIndiansDiabetes[,1:8], 2, skewness)
# display skewness, larger/smaller deviations from 0 show more skew
print(skew)
```

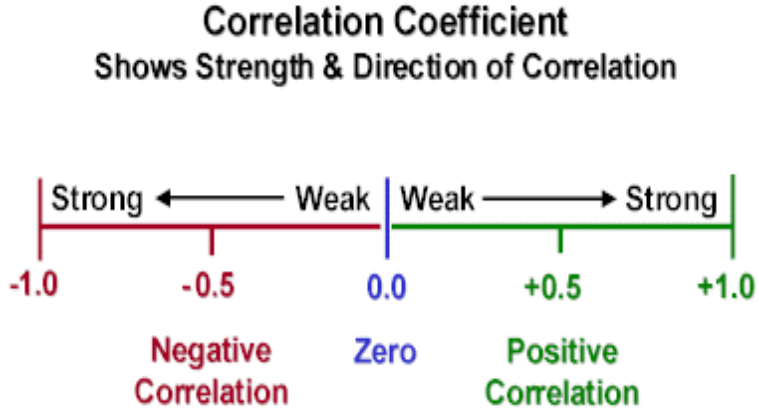
The further the distribution of the skew value from zero, the larger the skew to the left (negative skew value) or right (positive skew value).

```
  pregnant    glucose   pressure    triceps    insulin       mass   pedigree        age
 0.8981549  0.1730754 -1.8364126  0.1089456  2.2633826 -0.4273073  1.9124179  1.1251880
```

Correlations

- Correlation is a statistical technique that can show whether and how strongly pairs of variables are related.
- The main result of a correlation is called the correlation coefficient (or "r"). It ranges from -1.0 to +1.0. The closer r is to +1 or -1, the more closely the two variables are related.





Correlations

- For numeric attributes, an excellent way to think about attribute-to-attribute interactions is to calculate correlations for each pair of attributes.

```
# load the packages
library(mlbench)
# load the dataset
data(PimaIndiansDiabetes)
# calculate a correlation matrix for numeric variables
correlations <- cor(PimaIndiansDiabetes[,1:8])
# display the correlation matrix
print(correlations)
```

Calculate Correlations

Correlations contd...

- This creates a symmetrical table of all pairs of attribute correlations for numerical data.
- Deviations from zero show more positive or negative correlation. Values above 0.75 or below -0.75 are perhaps more interesting as they show a high correlation or high negative correlation.
- Values of 1 and -1 show full positive or negative correlation.

	pregnant	glucose	pressure	triceps	insulin	mass	pedigree	age
pregnant	1.00000000	0.12945867	0.14128198	-0.08167177	-0.07353461	0.01768309	-0.03352267	0.54434123
glucose	0.12945867	1.00000000	0.15258959	0.05732789	0.33135711	0.22107107	0.13733730	0.26351432
pressure	0.14128198	0.15258959	1.00000000	0.20737054	0.08893338	0.28180529	0.04126495	0.23952795
triceps	-0.08167177	0.05732789	0.20737054	1.00000000	0.43678257	0.39257320	0.18392757	-0.11397026
insulin	-0.07353461	0.33135711	0.08893338	0.43678257	1.00000000	0.19785906	0.18507093	-0.04216295
mass	0.01768309	0.22107107	0.28180529	0.39257320	0.19785906	1.00000000	0.14064695	0.03624187
pedigree	-0.03352267	0.13733730	0.04126495	0.18392757	0.18507093	0.14064695	1.00000000	0.03356131
age	0.54434123	0.26351432	0.23952795	-0.11397026	-0.04216295	0.03624187	0.03356131	1.00000000

Output of Correlations

Data Pre-Processing

Data Pre-Processing in R

The caret package in R provides a number of useful data transforms. These transforms can be used in two ways:

- **Standalone :**
  - Transforms can be modeled from training data and applied to multiple datasets. The model of the transform is prepared using the preProcess() function and applied to a dataset using the predict() function.
- **Training :**
  - Transforms can be prepared and applied automatically during model evaluation. Transforms applied during training are prepared using the preProcess() and passed to the train() function via the preProcess argument.

Data Pre-Processing in R contd...

Below is a quick summary of all of the transform methods supported in the argument to the preProcess() function in caret.

BoxCox	apply a Box-Cox transform, values must be non-zero and positive.
YeoJohnson	apply a Yeo-Johnson transform, like a BoxCox, but values can be negative.
expoTrans	apply a power transform like BoxCox and YeoJohnson.
zv	remove attributes with a zero variance (all the same value).
nzv	remove attributes with a near zero variance (close to the same value).
center	divide values by standard deviation.
scale	subtract mean from values.
range	normalize values.
pca	transform data to the principal components.

Scale Data

The scale transform calculates the standard deviation for an attribute and divides each value by that standard deviation.

```
# load packages
library(caret)
# load the dataset
data(iris)
# summarize data
summary(iris[,1:4])
# calculate the pre-process parameters from the dataset
preprocessParams <- preProcess(iris[,1:4], method=c("scale"))
# summarize transform parameters
print(preprocessParams)
# transform the dataset using the parameters
transformed <- predict(preprocessParams, iris[,1:4])
# summarize the transformed dataset
summary(transformed)
```

Calculate scale data transform.

Scale Data contd...

The scale transform calculates the standard deviation for an attribute and divides each value by that standard deviation.

```
 Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
Median :5.800   Median :3.000   Median :4.350   Median :1.300
Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500

Created from 150 samples and 4 variables

Pre-processing:
  - centered (4)
  - ignored (0)
 Sepal.Length      Sepal.Width       Petal.Length      Petal.Width
Min.   :-1.54333   Min.   :-1.05733   Min.   :-2.758   Min.   :-1.0993
1st Qu.:-0.74333   1st Qu.:-0.25733   1st Qu.:-2.158   1st Qu.:-0.8993
Median :-0.04333   Median :-0.05733   Median : 0.592   Median : 0.1007
Mean   : 0.00000   Mean   : 0.00000   Mean   : 0.000   Mean   : 0.0000
3rd Qu.: 0.55667   3rd Qu.: 0.24267   3rd Qu.: 1.342   3rd Qu.: 0.6007
Max.   : 2.05667   Max.   : 1.34267   Max.   : 3.142   Max.   : 1.3007
```

Output of scale data transform.

Center Data

The center transform calculates the mean for an attribute and subtracts it from each value.

```
# load packages
library(caret)
# load the dataset
data(iris)
# summarize data
summary(iris[,1:4])
# calculate the pre-process parameters from the dataset
preprocessParams <- preProcess(iris[,1:4], method=c("center"))
# summarize transform parameters
print(preprocessParams)
# transform the dataset using the parameters
transformed <- predict(preprocessParams, iris[,1:4])
# summarize the transformed dataset
summary(transformed)
```

Calculate center data transform.

Center Data contd...

```
  Sepal.Length   Sepal.Width    Petal.Length   Petal.Width
 Min.   :4.300  Min.   :2.000  Min.   :1.000  Min.   :0.100
 1st Qu.:5.100  1st Qu.:2.800  1st Qu.:1.600  1st Qu.:0.300
 Median :5.800  Median :3.000  Median :4.350  Median :1.300
 Mean   :5.843  Mean   :3.057  Mean   :3.758  Mean   :1.199
 3rd Qu.:6.400  3rd Qu.:3.300  3rd Qu.:5.100  3rd Qu.:1.800
 Max.   :7.900  Max.   :4.400  Max.   :6.900  Max.   :2.500

Created from 150 samples and 4 variables

Pre-processing:
  - centered (4)
  - ignored (0)

 Sepal.Length      Sepal.Width       Petal.Length    Petal.Width
 Min.   :-1.54333  Min.   :-1.05733  Min.   :-2.758  Min.   :-1.0993
 1st Qu.:-0.74333  1st Qu.:-0.25733  1st Qu.:-2.158  1st Qu.:-0.8993
 Median :-0.04333  Median :-0.05733  Median : 0.592  Median : 0.1007
 Mean   : 0.00000  Mean   : 0.00000  Mean   : 0.000  Mean   : 0.0000
 3rd Qu.: 0.55667  3rd Qu.: 0.24267  3rd Qu.: 1.342  3rd Qu.: 0.6007
 Max.   : 2.05667  Max.   : 1.34267  Max.   : 3.142  Max.   : 1.3007
```

Output of center data transform.

Standardize Data

➔A standardized variable (sometimes called a z-score or a standard score) is a variable that has been re scaled to have a mean of zero and a standard deviation of one.

➔A standardized variable (Z) redefines each observation in terms of the number of standard deviations from the mean.

Standardization formula for a population:

$$z_i = \frac{x_i - \mu}{\sigma}$$

Standardization formula for a sample:

$$z_i = \frac{x_i - \overline{x}}{s}$$

- z_i tells how far away the observation is from the mean (in terms of σ).
- A negative z value means the observation is below the mean.
- Positive z means the observation is above the mean.

Standardize Data

Combining the scale and center transforms will standardize your data. Attributes will have a mean value of 0 and a standard deviation of 1.

```
# load packages
library(caret)
# load the dataset
data(iris)
# summarize data
summary(iris[,1:4])
# calculate the pre-process parameters from the dataset
preprocessParams <- preProcess(iris[,1:4], method=c("center", "scale"))
# summarize transform parameters
print(preprocessParams)
# transform the dataset using the parameters
transformed <- predict(preprocessParams, iris[,1:4])
# summarize the transformed dataset
summary(transformed)
```

Calculate standardize data transform.

Standardize Data contd...

```
  Sepal.Length   Sepal.Width    Petal.Length   Petal.Width
 Min.   :4.300  Min.   :2.000  Min.   :1.000  Min.   :0.100
 1st Qu.:5.100  1st Qu.:2.800  1st Qu.:1.600  1st Qu.:0.300
 Median :5.800  Median :3.000  Median :4.350  Median :1.300
 Mean   :5.843  Mean   :3.057  Mean   :3.758  Mean   :1.199
 3rd Qu.:6.400  3rd Qu.:3.300  3rd Qu.:5.100  3rd Qu.:1.800
 Max.   :7.900  Max.   :4.400  Max.   :6.900  Max.   :2.500

Created from 150 samples and 4 variables

Pre-processing:
  - centered (4)
  - ignored (0)
  - scaled (4)

 Sepal.Length      Sepal.Width      Petal.Length     Petal.Width
 Min.   :-1.86378  Min.   :-2.4258  Min.   :-1.5623  Min.   :-1.4422
 1st Qu.:-0.89767  1st Qu.:-0.5904  1st Qu.:-1.2225  1st Qu.:-1.1799
 Median :-0.05233  Median :-0.1315  Median : 0.3354  Median : 0.1321
 Mean   : 0.00000  Mean   : 0.0000  Mean   : 0.0000  Mean   : 0.0000
 3rd Qu.: 0.67225  3rd Qu.: 0.5567  3rd Qu.: 0.7602  3rd Qu.: 0.7880
 Max.   : 2.48370  Max.   : 3.0805  Max.   : 1.7799  Max.   : 1.7064
```

Output standardize data transform.

Normalize Data

Data values can be scaled into the range of [0, 1] which is called normalization.

```
library(caret)
# load the dataset
data(iris)
# summarize data
summary(iris[,1:4])
# calculate the pre-process parameters from the dataset
preprocessParams <- preProcess(iris[,1:4], method=c("range"))
# summarize transform parameters
print(preprocessParams)
# transform the dataset using the parameters
transformed <- predict(preprocessParams, iris[,1:4])
# summarize the transformed dataset
summary(transformed)
```

Calculate normalize data transform.

Normalize Data

- Eliminates the unit of measurement by transforming the data into new scores with a mean of 0 and a standard deviation of 1.
- Data values can be scaled into the range of [0, 1] which is called normalization.
- To normalize data means to fit the data within unity, so all the data will take on a value between 0 and 1. Many formulas are available:
- Ex: $$X_{i,\,0\text{ to }1} = \frac{X_i - X_{Min}}{X_{Max} - X_{Min}}$$

Normalize Data

```
library(caret)
# load the dataset
data(iris)
# summarize data
summary(iris[,1:4])
# calculate the pre-process parameters from the dataset
preprocessParams <- preProcess(iris[,1:4], method=c("range"))
# summarize transform parameters
print(preprocessParams)
# transform the dataset using the parameters
transformed <- predict(preprocessParams, iris[,1:4])
# summarize the transformed dataset
summary(transformed)
```

Calculate normalize data transform.

Normalize Data contd...

```
  Sepal.Length   Sepal.Width    Petal.Length   Petal.Width
 Min.   :4.300  Min.   :2.000  Min.   :1.000  Min.   :0.100
 1st Qu.:5.100  1st Qu.:2.800  1st Qu.:1.600  1st Qu.:0.300
 Median :5.800  Median :3.000  Median :4.350  Median :1.300
 Mean   :5.843  Mean   :3.057  Mean   :3.758  Mean   :1.199
 3rd Qu.:6.400  3rd Qu.:3.300  3rd Qu.:5.100  3rd Qu.:1.800
 Max.   :7.900  Max.   :4.400  Max.   :6.900  Max.   :2.500

Created from 150 samples and 4 variables

Pre-processing:
  - ignored (0)
  - re-scaling to [0, 1] (4)


  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
 Min.   :0.0000  Min.   :0.0000  Min.   :0.0000  Min.   :0.00000
 1st Qu.:0.2222  1st Qu.:0.3333  1st Qu.:0.1017  1st Qu.:0.08333
 Median :0.4167  Median :0.4167  Median :0.5678  Median :0.50000
 Mean   :0.4287  Mean   :0.4406  Mean   :0.4675  Mean   :0.45806
 3rd Qu.:0.5833  3rd Qu.:0.5417  3rd Qu.:0.6949  3rd Qu.:0.70833
 Max.   :1.0000  Max.   :1.0000  Max.   :1.0000  Max.   :1.00000
```

Output normalize data transform.

Box-Cox Transform

When an attribute has a Gaussian-like distribution but is shifted, this is called a skew. The distribution of an attribute can be shifted to reduce the skew and make it more Gaussian. The BoxCox transform can perform this operation (assumes all values are positive).

```
# load packages
library(mlbench)
library(caret)
# load the dataset
data(PimaIndiansDiabetes)
# summarize pedigree and age
summary(PimaIndiansDiabetes[,7:8])
# calculate the pre-process parameters from the dataset
preprocessParams <- preProcess(PimaIndiansDiabetes[,7:8], method=c("BoxCox"))
# summarize transform parameters
print(preprocessParams)
# transform the dataset using the parameters
transformed <- predict(preprocessParams, PimaIndiansDiabetes[,7:8])
# summarize the transformed dataset (note pedigree and age)
summary(transformed)
```

Calculate Box-Cox data transform.

Box-Cox Transform contd...

```
    pedigree          age
 Min.   :0.0780  Min.   :21.00
 1st Qu.:0.2437  1st Qu.:24.00
 Median :0.3725  Median :29.00
 Mean   :0.4719  Mean   :33.24
 3rd Qu.:0.6262  3rd Qu.:41.00
 Max.   :2.4200  Max.   :81.00

Created from 768 samples and 2 variables

Pre-processing:
  - Box-Cox transformation (2)
  - ignored (0)

Lambda estimates for Box-Cox transformation:
-0.1, -1.1

    pedigree           age
 Min.   :-2.5510  Min.   :0.8772
 1st Qu.:-1.4116  1st Qu.:0.8815
 Median :-0.9875  Median :0.8867
 Mean   :-0.9599  Mean   :0.8874
 3rd Qu.:-0.4680  3rd Qu.:0.8938
 Max.   : 0.8838  Max.   :0.9019
```

Output Box-Cox data transform.

Yeo-Johnson Transform

The YeoJohnson transform another power-transform like Box-Cox, but it supports raw values that are equal to zero and negative.

```
# load packages
library(mlbench)
library(caret)
# load the dataset
data(PimaIndiansDiabetes)
# summarize pedigree and age
summary(PimaIndiansDiabetes[,7:8])
# calculate the pre-process parameters from the dataset
preprocessParams <- preProcess(PimaIndiansDiabetes[,7:8], method=c("YeoJohnson"))
# summarize transform parameters
print(preprocessParams)
# transform the dataset using the parameters
transformed <- predict(preprocessParams, PimaIndiansDiabetes[,7:8])
# summarize the transformed dataset (note pedigree and age)
summary(transformed)
```

Calculate Yeo-Johnson data transform.

Yeo-Johnson Transform contd...

```
   pedigree          age
 Min.   :0.0780  Min.   :21.00
 1st Qu.:0.2437  1st Qu.:24.00
 Median :0.3725  Median :29.00
 Mean   :0.4719  Mean   :33.24
 3rd Qu.:0.6262  3rd Qu.:41.00
 Max.   :2.4200  Max.   :81.00

Created from 768 samples and 2 variables

Pre-processing:
  - ignored (0)
  - Yeo-Johnson transformation (2)

Lambda estimates for Yeo-Johnson transformation:
-2.25, -1.15

   pedigree          age
 Min.   :0.0691  Min.   :0.8450
 1st Qu.:0.1724  1st Qu.:0.8484
 Median :0.2265  Median :0.8524
 Mean   :0.2317  Mean   :0.8530
 3rd Qu.:0.2956  3rd Qu.:0.8580
 Max.   :0.4164  Max.   :0.8644
```

Output Yeo-Johnson data transform.

Principal Component Analysis Transform

- The PCA transforms the data to return only the principal components, a technique form multivariate statistics and linear algebra.
- The transform keeps those components above the variance threshold (default=0.95) or the number of components can be specified (pcaComp).
- The result is attributes that are uncorrelated, useful for algorithms like linear and generalized linear regression.

```
# load the packages
library(mlbench)
# load the dataset
data(iris)
# summarize dataset
summary(iris)
# calculate the pre-process parameters from the dataset
preprocessParams <- preProcess(iris, method=c("center", "scale", "pca"))
# summarize transform parameters
print(preprocessParams)
# transform the dataset using the parameters
transformed <- predict(preprocessParams, iris)
# summarize the transformed dataset
summary(transformed)
```

Calculate PCA data transform.

Principal Component Analysis Transform

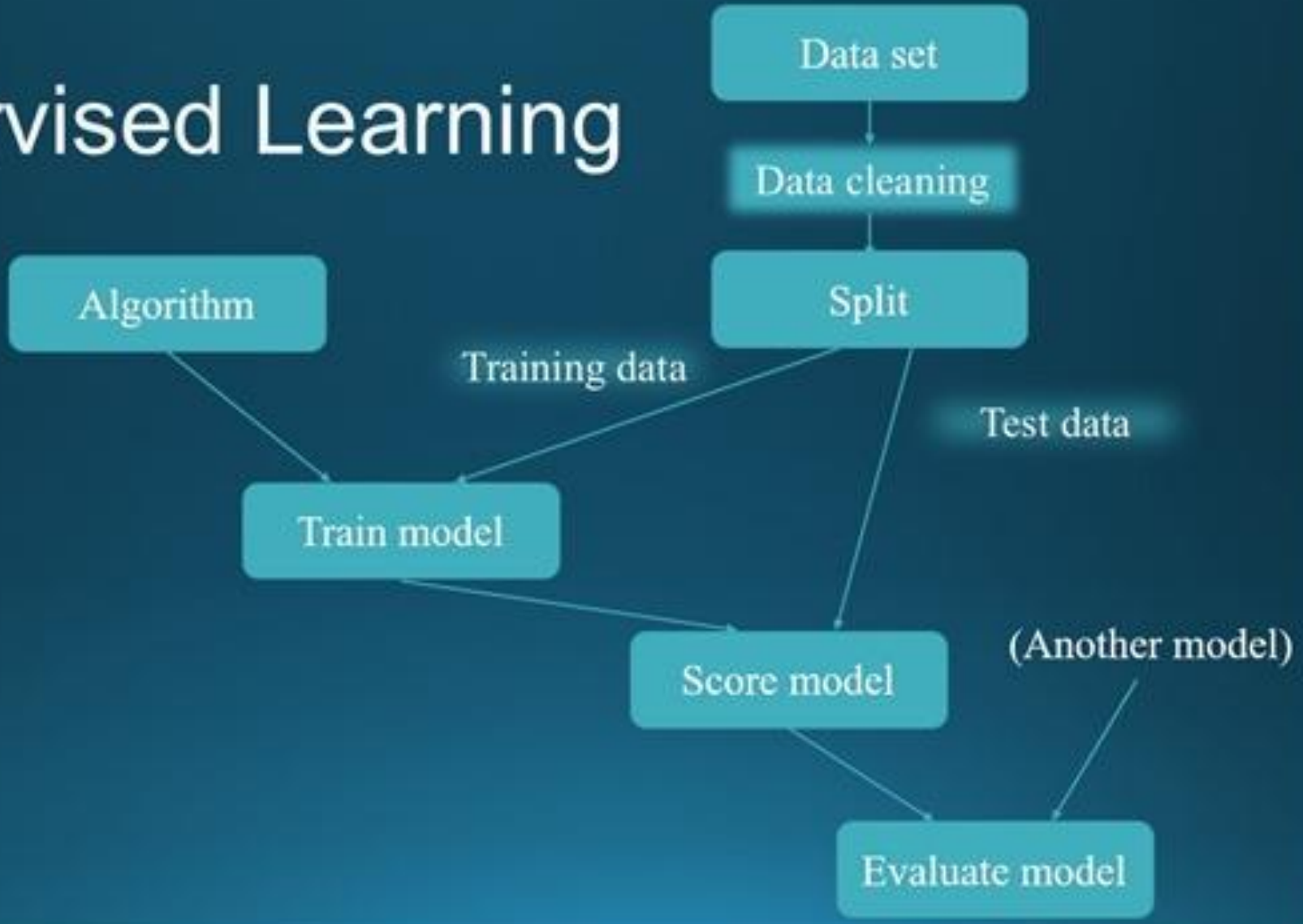
- Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components.
- The PCA transforms the data to return only the principal components, a technique form multivariate statistics and linear algebra.
- The transform keeps those components above the variance threshold (default=0.95) or the number of components can be specified (pcaComp).
- The result is attributes that are uncorrelated, useful for algorithms like linear and generalized linear regression.

Principal Component Analysis Transform contd...

```
  Sepal.Length   Sepal.Width    Petal.Length   Petal.Width          Species
 Min.   :4.300  Min.   :2.000  Min.   :1.000  Min.   :0.100  setosa    :50
 1st Qu.:5.100  1st Qu.:2.800  1st Qu.:1.600  1st Qu.:0.300  versicolor:50
 Median :5.800  Median :3.000  Median :4.350  Median :1.300  virginica :50
 Mean   :5.843  Mean   :3.057  Mean   :3.758  Mean   :1.199
 3rd Qu.:6.400  3rd Qu.:3.300  3rd Qu.:5.100  3rd Qu.:1.800
 Max.   :7.900  Max.   :4.400  Max.   :6.900  Max.   :2.500
Created from 150 samples and 5 variables

Pre-processing:
  - centered (4)
  - ignored (1)
  - principal component signal extraction (4)
  - scaled (4)

PCA needed 2 components to capture 95 percent of the variance

      Species        PC1                PC2
 setosa    :50   Min.   :-2.7651  Min.   :-2.67732
 versicolor:50   1st Qu.:-2.0957  1st Qu.:-0.59205
 virginica :50   Median : 0.4169  Median :-0.01744
                 Mean   : 0.0000  Mean   : 0.00000
                 3rd Qu.: 1.3385  3rd Qu.: 0.59649
                 Max.   : 3.2996  Max.   : 2.64521
```

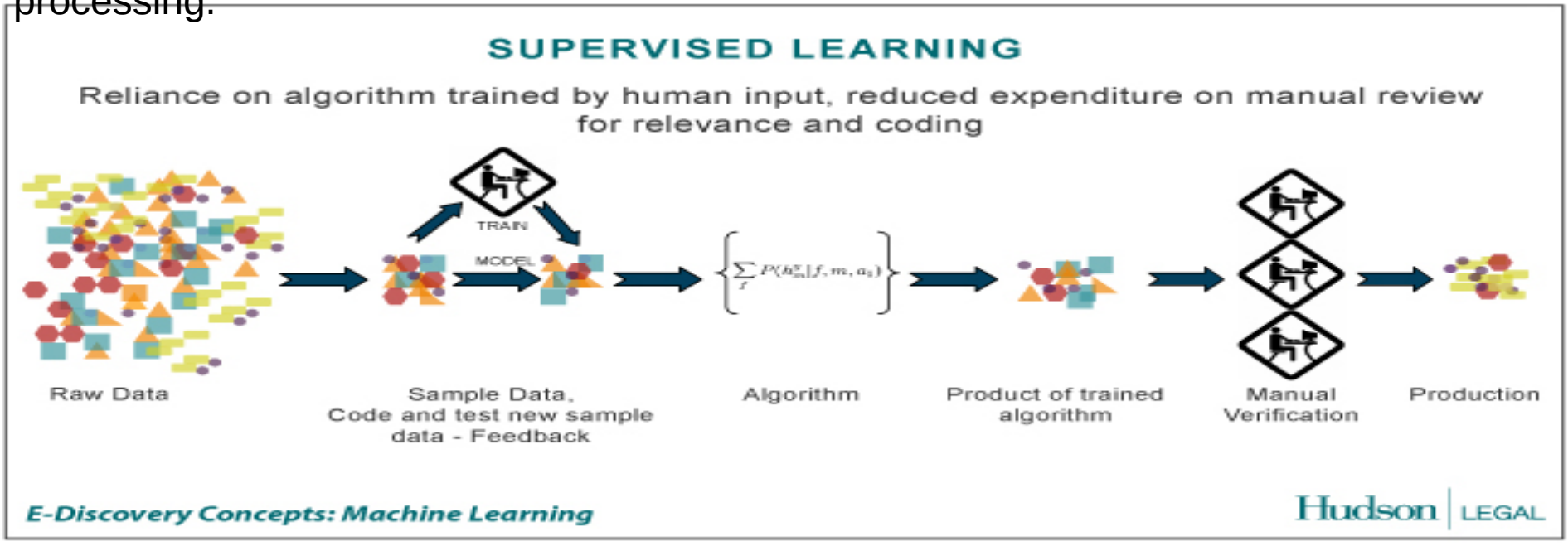
Output PCA data transform.

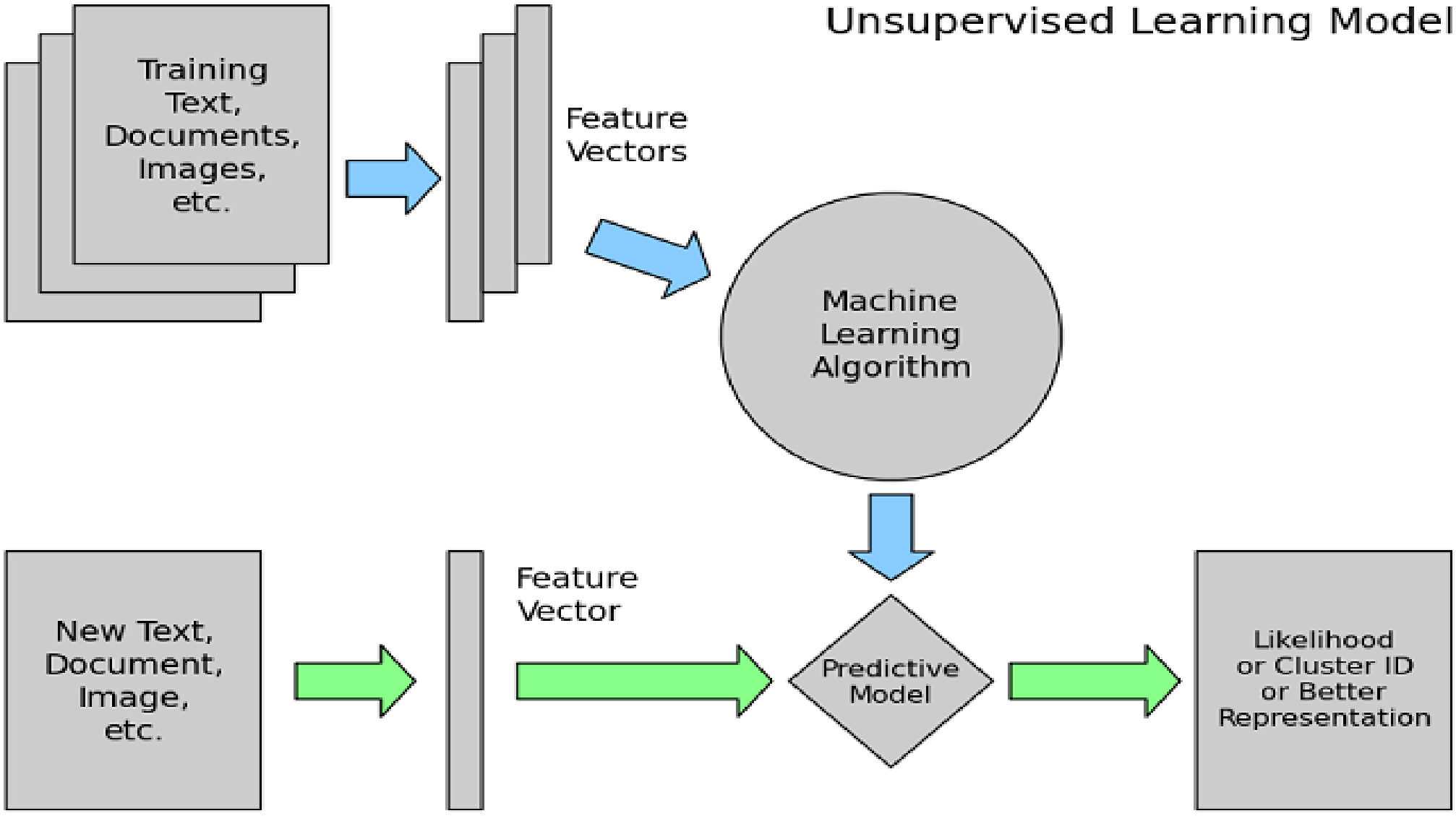
Supervised Learning



Supervised Learning

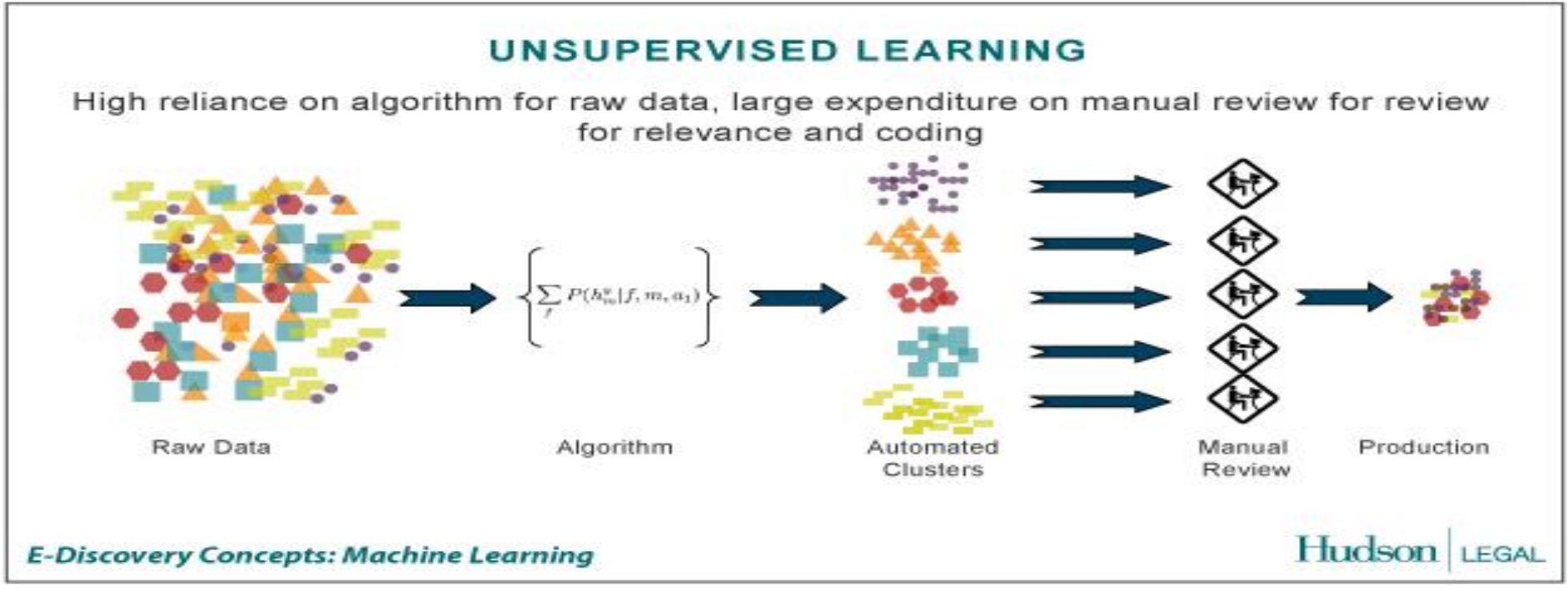
Supervised learning, in the context of artificial intelligence (AI) and machine learning, is a type of system in which both input and desired output data are provided. Input and output data are labelled for classification to provide a learning basis for future data processing.




Unsupervised Learning Model
Training Text, Documents, Images, etc.
Feature Vectors
Machine Learning Algorithm
New Text, Document, Image, etc.
Feature Vector
Predictive Model
Likelihood or Cluster ID or Better Representation

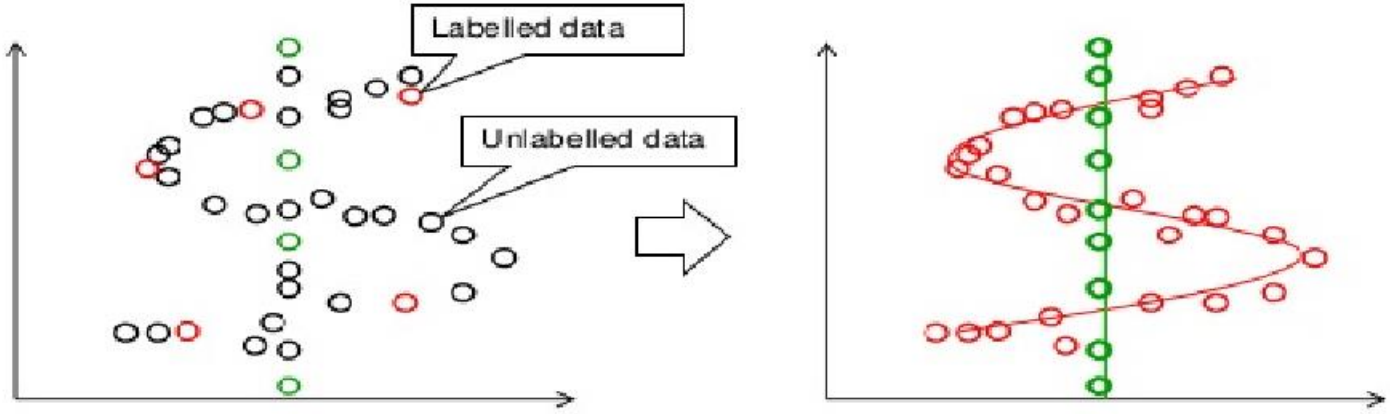
Unsupervised Learning

Unsupervised learning is the training of an artificial intelligence (AI) algorithm using information that is neither classified nor labeled and allowing the algorithm to act on that information without guidance.

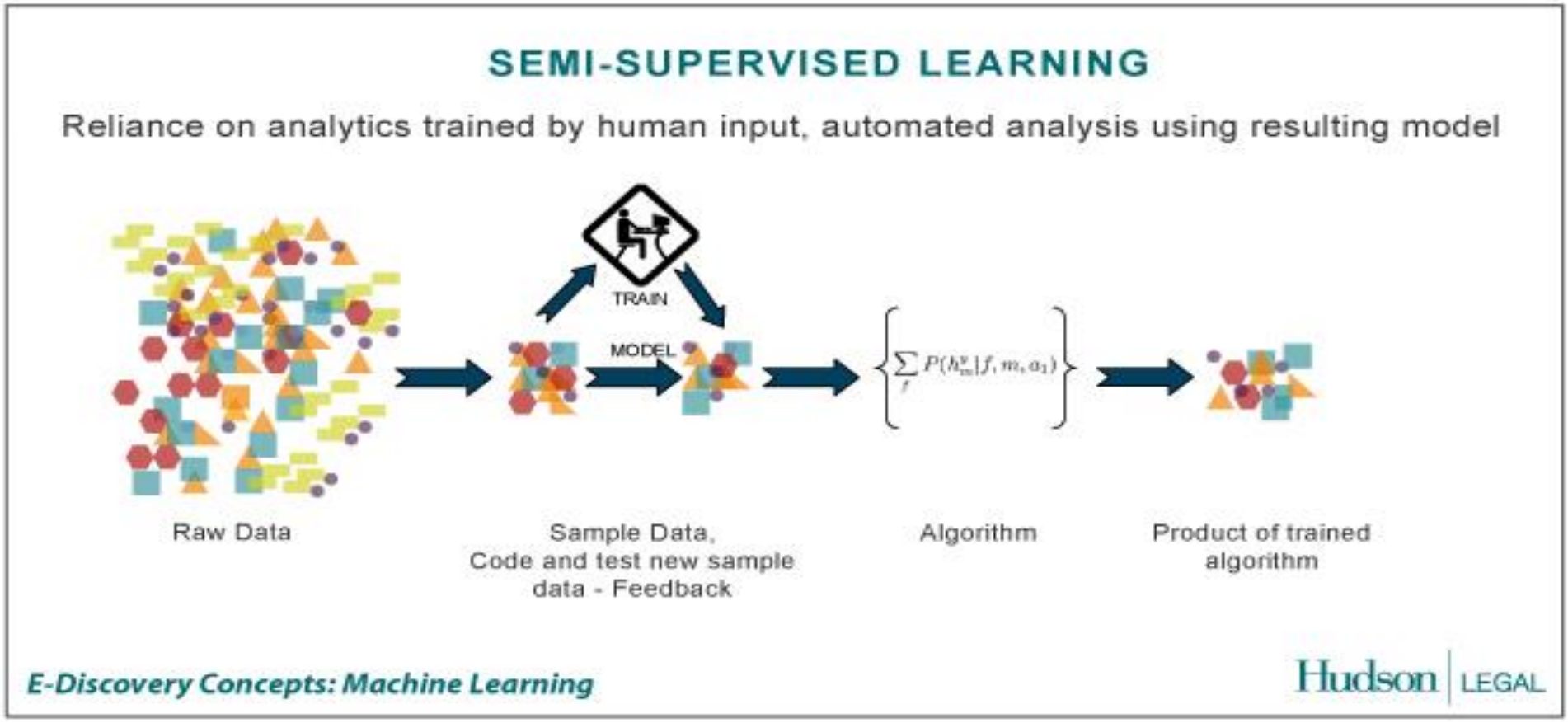


Semi - supervised Learning

Semi-supervised learning is a class of supervised learning tasks and techniques that also make use of unlabeled data for training – typically a small amount of labeled data with a large amount of unlabeled data.

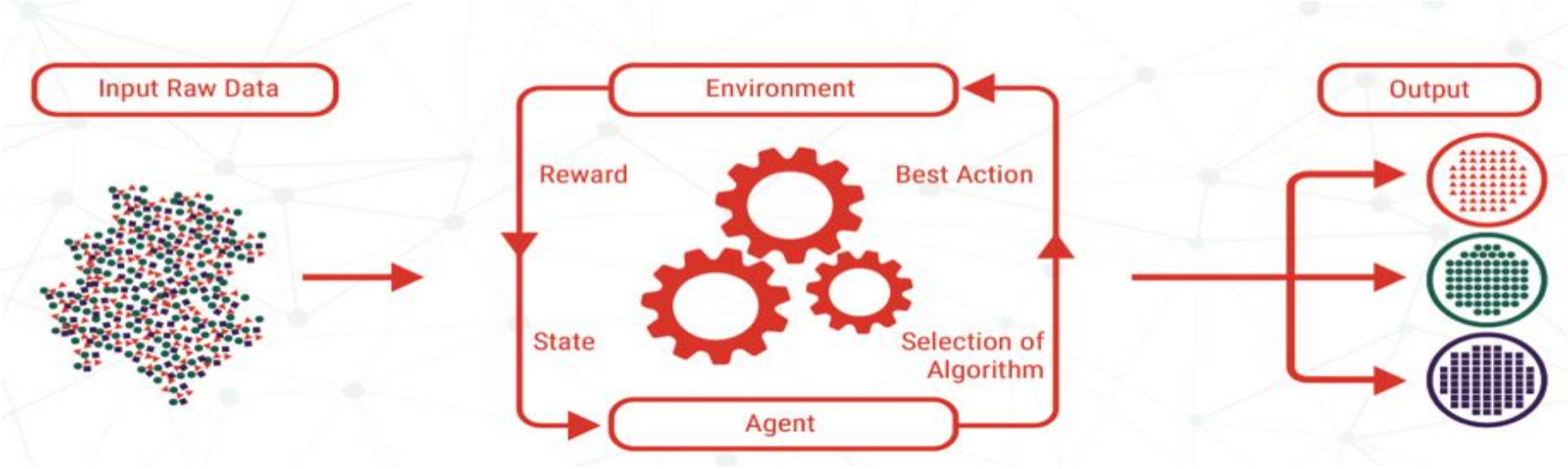


Semi - supervised Learning

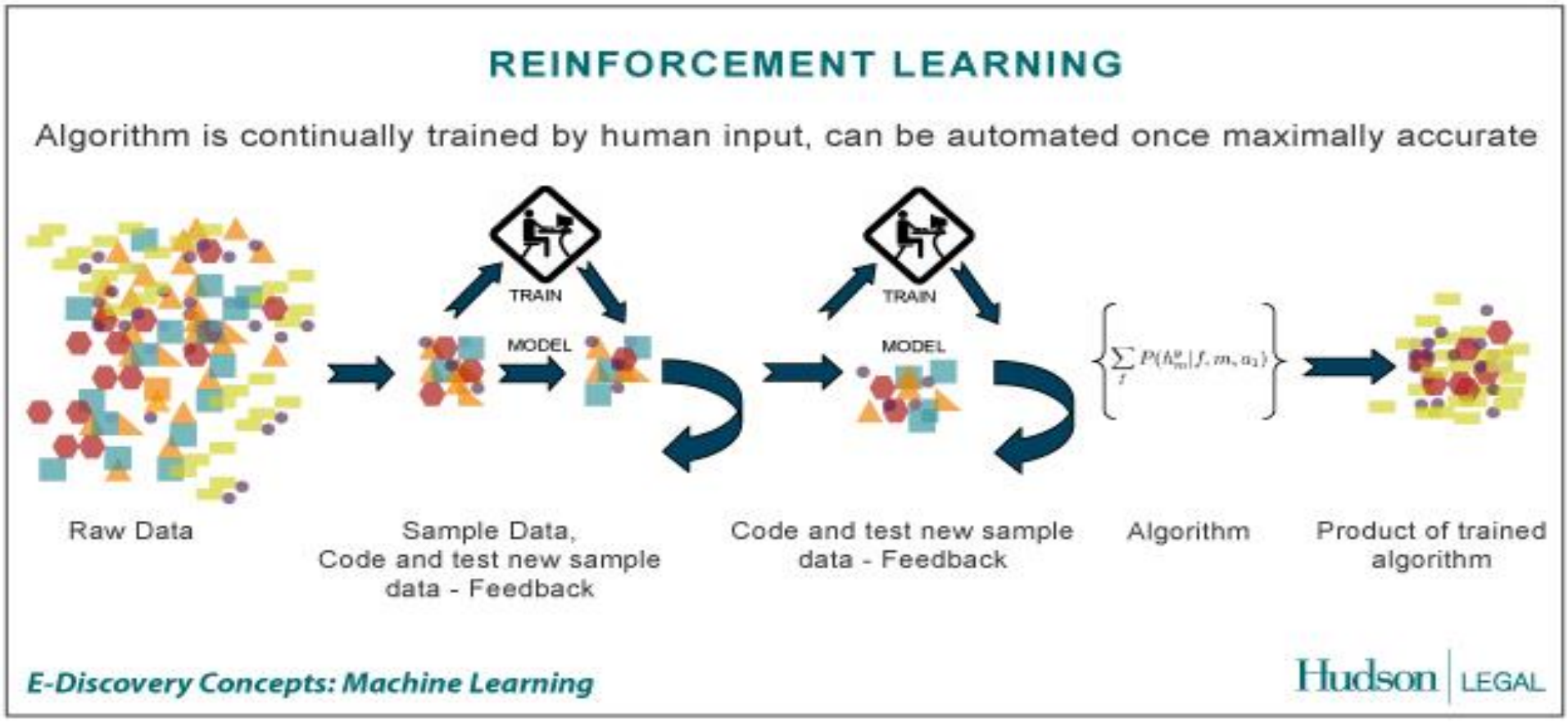


Reinforcement Learning

Reinforcement Learning is a type of Machine Learning, and thereby also a branch of Artificial Intelligence. It allows machines and software agents to automatically determine the ideal behavior within a specific context, in order to maximize its performance.



Reinforcement Learning



Data Split

- Data splitting involves partitioning the data into an explicit training dataset used to prepare the model and an unseen test dataset used to evaluate the models performance on unseen data.
- It is useful when you have a very large dataset so that the test dataset can provide a meaningful estimation of performance, or for when you are using slow methods and need a quick approximation of performance.

```
# load the packages
library(caret)
library(klaR)
# load the iris dataset
data(iris)
# define an 80%/20% train/test split of the dataset
trainIndex <- createDataPartition(iris$Species, p=0.80, list=FALSE)
dataTrain <- iris[ trainIndex,]
dataTest <- iris[-trainIndex,]
```

Model Evaluation Metrics

Accuracy and Kappa

- Accuracy and Kappa are the default metrics used to evaluate algorithms on binary and multiclass classification datasets in caret.
  - **Accuracy** is the percentage of correctly classified instances out of all instances. It is more useful on a binary classification than multi-class classification problem because it can be less clear exactly how the accuracy breaks down across those classes (e.g. you need to go deeper with a confusion matrix).
  - **Kappa** or Cohen's Kappa is like classification accuracy, except that it is normalized at the baseline of random chance on your data set
  - Kappa is a more useful measure to use on problems that have an imbalance in the classes (e.g. a 70% to 30% split for classes 0 and 1 and you can achieve 70% accuracy by predicting all instances are for class 0).

Accuracy and Kappa contd...

```
# load packages
library(caret)
library(mlbench)
# load the dataset
data(PimaIndiansDiabetes)
# prepare resampling method
trainControl <- trainControl(method="cv", number=5)
set.seed(7)
fit <- train(diabetes~., data=PimaIndiansDiabetes, method="glm", metric="Accuracy",
    trControl=trainControl)
# display results
print(fit)
```

Calculate Accuracy and Kappa metrics.

```
Generalized Linear Model

768 samples
  8 predictor
  2 classes: 'neg', 'pos'

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 614, 614, 615, 615, 614
Resampling results

  Accuracy  Kappa     Accuracy SD Kappa SD
  0.7695442 0.4656824 0.02692468  0.0616666
```

Output of Accuracy and Kappa metrics.

Metric Description Formula
Accuracy : what % of predictions were correct? (TP+TN)/(TP+TN+FP+FN)

Misclassification Rate : what % of prediction is wrong? (FP+FN)/(TP+TN+FP+FN)

True Positive Rate OR Sensitivity OR Recall (completeness) : what % of positive cases did model catch? TP/(FN+TP)

False Positive Rate : what % of 'No' were predicted as 'Yes'? FP/(FP+TN)

Specificity : what % of 'No' were predicted as 'No‘? TN/(TN+FP)

Precision (exactness) : what % of positive predictions were correct? TP/(TP+FP)

F1 score : Weighted average of precision and recall : 2*((precision * recall) / (precision + recall))

RMSE and R2

- RMSE and R2 are the default metrics used to evaluate algorithms on regression datasets in caret.
    - **RMSE or Root Mean Squared Error** is the average deviation of the predictions from the observations. It is useful to get a gross idea of how well (or not) an algorithm is doing, in the units of the output variable.
    - **R2** spoken as **R Squared** or also called the coefficient of determination provides a goodness of-fit measure for the predictions to the observations. This is a value between 0 and 1 for no-fit and perfect fit respectively.

RMSE and R2 contd...

```
# load packages
library(caret)
# load data
data(longley)
# prepare resampling method
trainControl <- trainControl(method="cv", number=5)
set.seed(7)
fit <- train(Employed~., data=longley, method="lm", metric="RMSE", trControl=trainControl)
# display results
print(fit)
```

Calculate RMSE and RSquared metrics.

```
Linear Regression

16 samples
 6 predictor

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 12, 12, 14, 13, 13
Resampling results

  RMSE      Rsquared  RMSE SD    Rsquared SD
  0.3868618 0.9883114 0.1025042 0.01581824
```

Output of RMSE and RSquared metrics.

Area Under ROC Curve

- ROC metrics are only suitable for binary classification problems (e.g. two classes).
- ROC is actually the area under the ROC curve or AUC. The AUC represents a models ability to discriminate between positive and negative classes.
  - **Sensitivity** is the true positive rate also called the recall. It is the number of instances from the positive (first) class that actually predicted correctly.
  - **Specificity** is also called the true negative rate. Is the number of instances from the negative class (second class) that were actually predicted correctly.

Area Under ROC Curve contd...

Calculate ROC metrics.

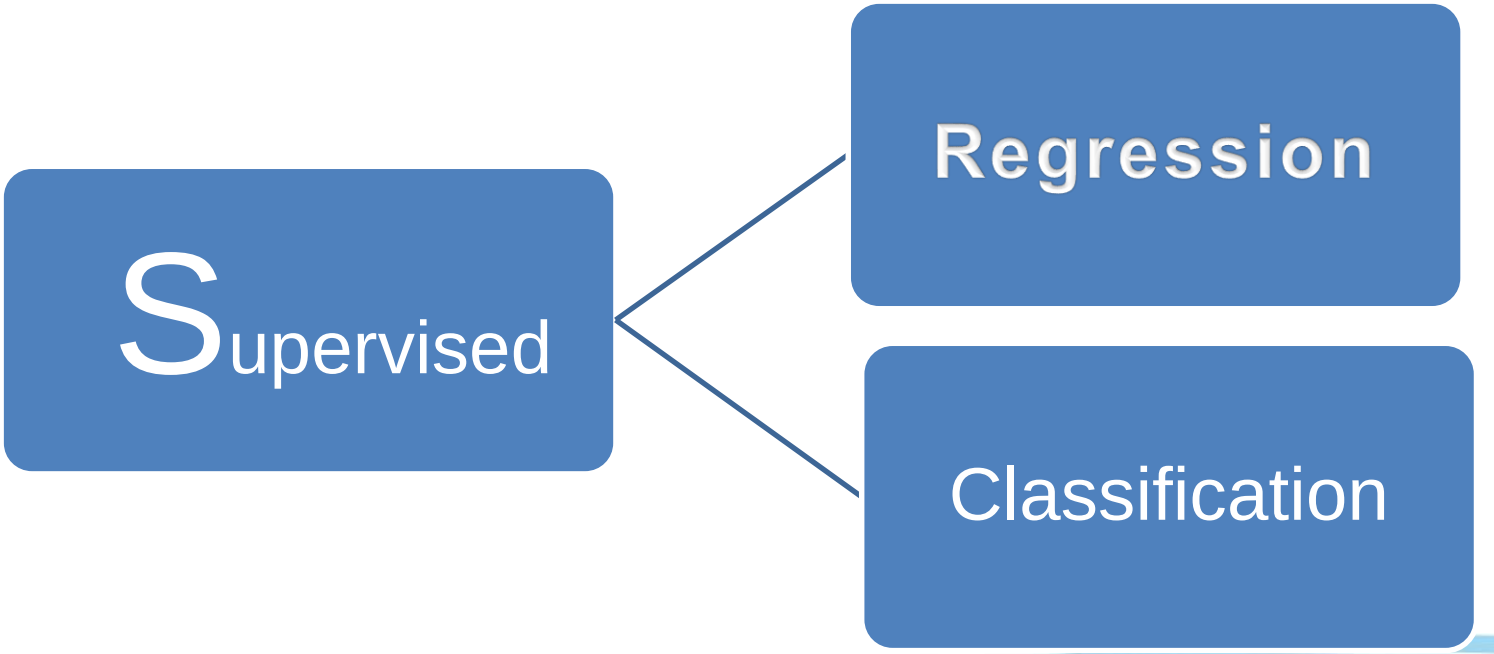
```
# load packages
library(caret)
library(mlbench)
# load the dataset
data(PimaIndiansDiabetes)
# prepare resampling method
trainControl <- trainControl(method="cv", number=5, classProbs=TRUE,
    summaryFunction=twoClassSummary)
set.seed(7)
fit <- train(diabetes~., data=PimaIndiansDiabetes, method="glm", metric="ROC",
    trControl=trainControl)
# display results
print(fit)
```

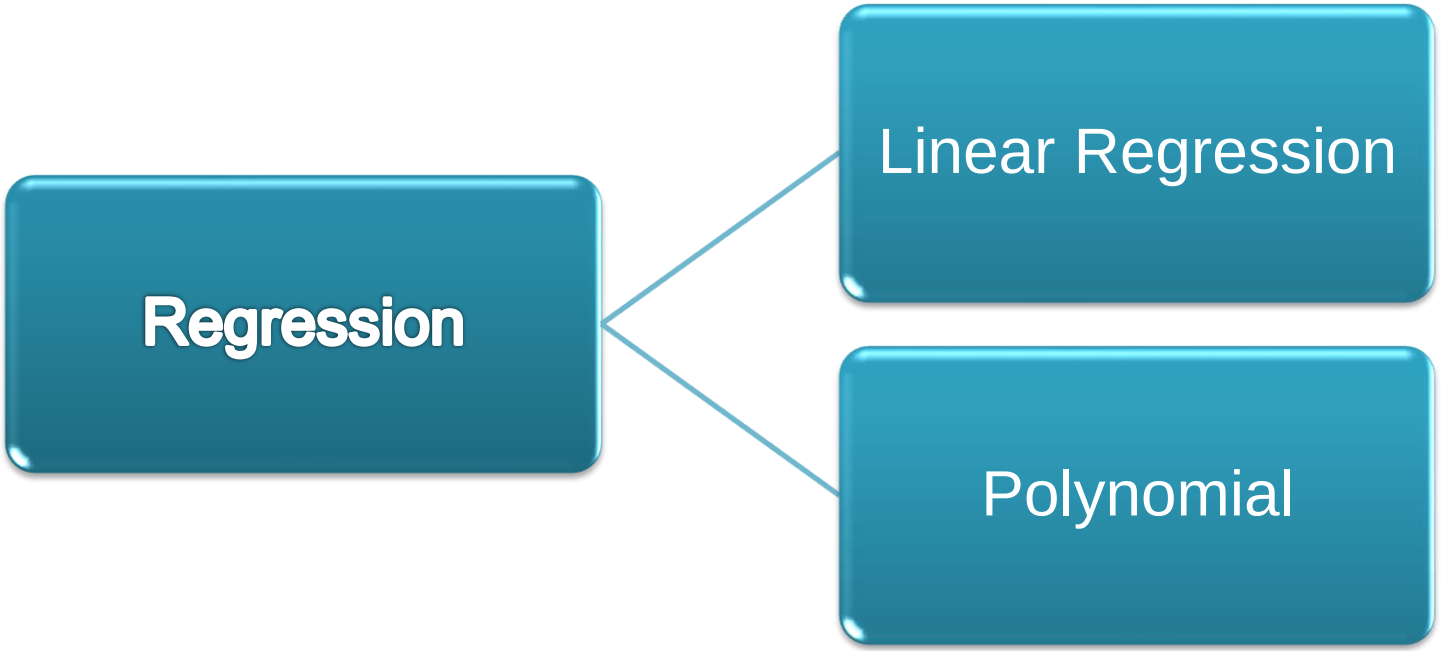
Output ROC metrics.

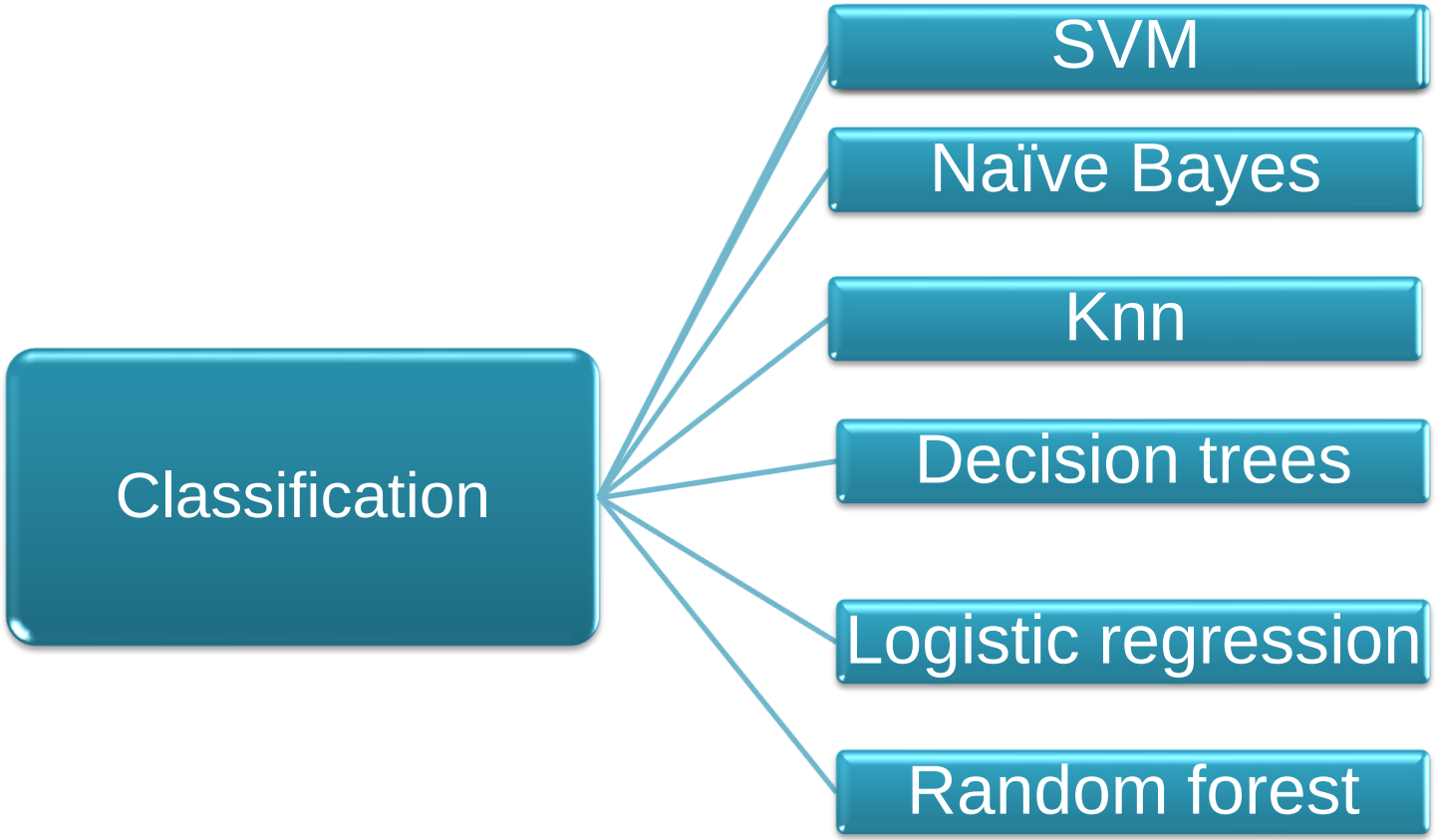
```
Generalized Linear Model

768 samples
  8 predictor
  2 classes: 'neg', 'pos'

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 614, 614, 615, 615, 614
Resampling results

  ROC       Sens   Spec       ROC SD      Sens SD     Spec SD
  0.8336003 0.882  0.5600978 0.02111279 0.03563706 0.0560184
```

Machine Learning Algorithms

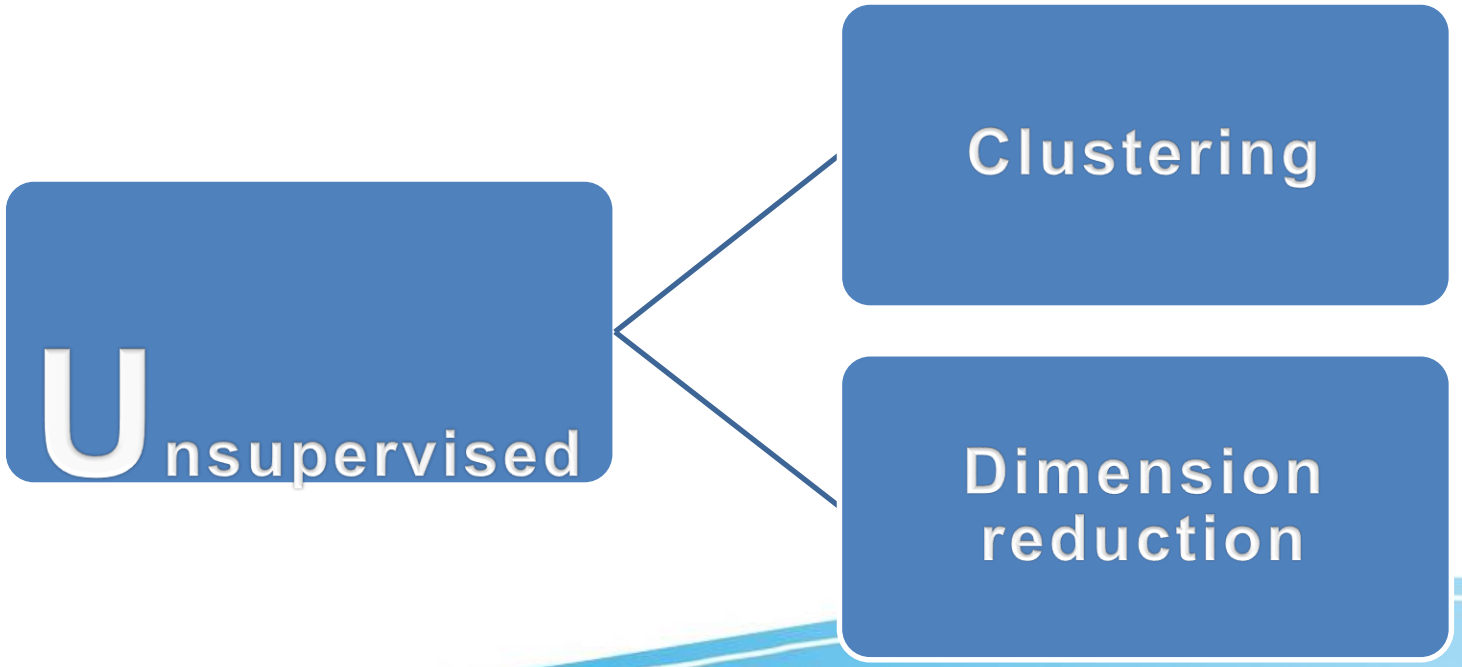
supervised learning algorithm

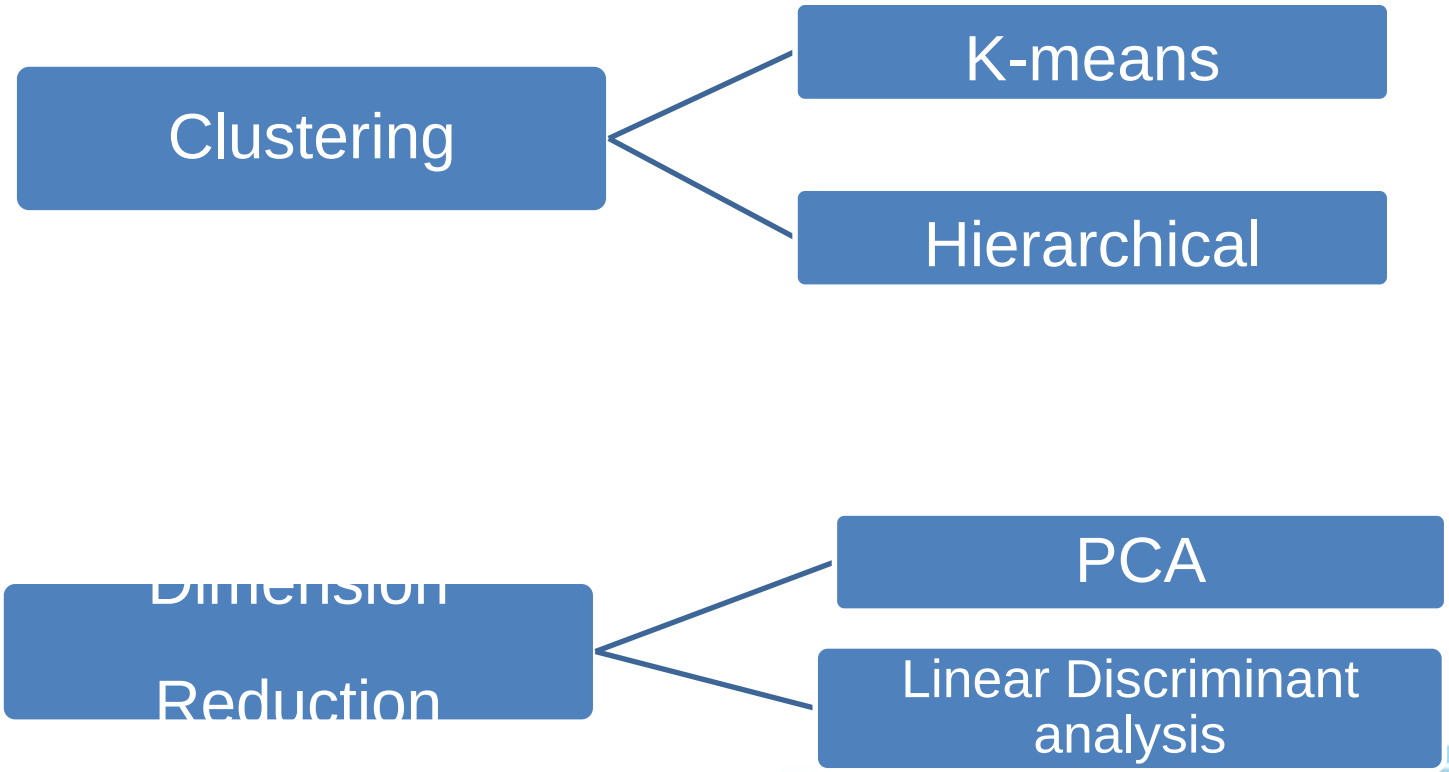



Regression
Linear Regression
Polynomial


Classification
SVM
Naïve Bayes
Knn
Decision trees
Logistic regression
Random forest

Unsupervised Learning algorithm



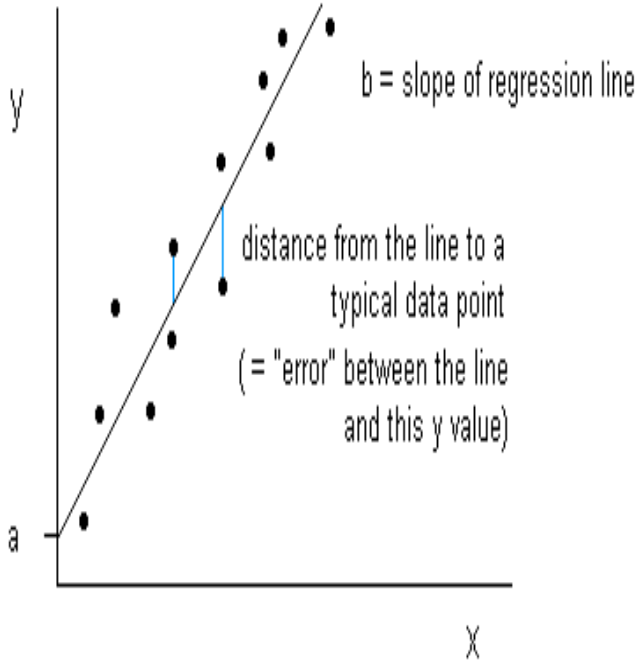

Clustering
K-means
Hierarchical
Dimension
Reduction
PCA
Linear Discriminant analysis

Linear Regression

- A learning algorithm to learn relation between dependent variable y with one or more explanatory variable x which are connected by linear relation.
- Given multiple dimensional data, the hypothesis for linear regression looks as below

  $h(x) = c0 + c1x1 + c2x2 + ...$

- x1, x2 are the explanatory variables and y is the dependent variable. h(x) is the hypothesis.
- Linear regression goal is to learn c0,c1,c2



Linear Regression

The lm() function is in the stats package and creates a linear regression model using ordinary least squares.

```
# load the package
library(mlbench)
# load data
data(BostonHousing)
# fit model
fit <- lm(medv~., BostonHousing)
# summarize the fit
print(fit)
# make predictions
predictions <- predict(fit, BostonHousing)
# summarize accuracy
mse <- mean((BostonHousing$medv - predictions)^2)
print(mse)
```

Example of linear regression algorithm.

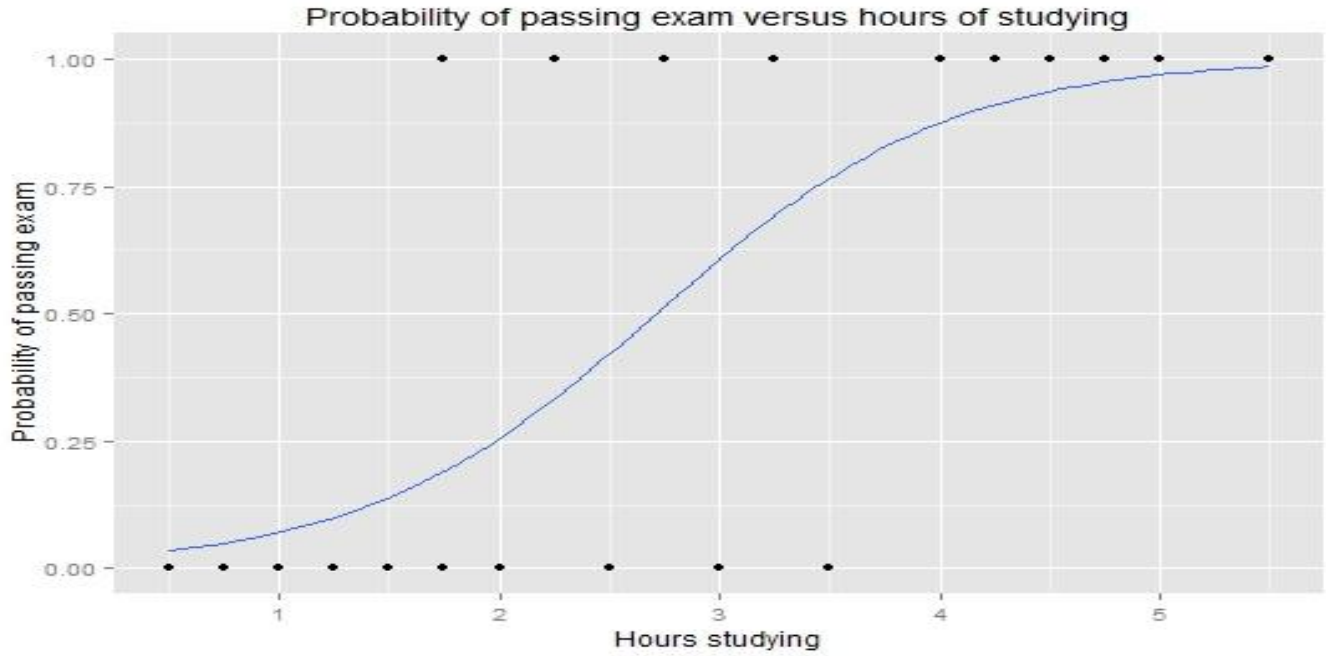
Linear Regression contd...

```
# load packages
library(caret)
library(mlbench)
# load dataset
data(BostonHousing)
# train
set.seed(7)
trainControl <- trainControl(method="cv", number=5)
fit.lm <- train(medv~., data=BostonHousing, method="lm", metric="RMSE", preProc=c("center",
    "scale"), trControl=trainControl)
# summarize fit
print(fit.lm)
```

Example of linear regression algorithm in caret.

Logistic Regression

- Logistic regression, or logit regression, or logit model is a regression model where the dependent variable (DV) is categorical.



Logistic Regression

The glm() function is in the stats package and creates a generalized linear model for regression or classification. It can be configured to perform a logistic regression suitable for binary classification problems.

Example of logistic regression algorithm.

```
# load the package
library(mlbench)
# Load the dataset
data(PimaIndiansDiabetes)
# fit model
fit <- glm(diabetes~., data=PimaIndiansDiabetes, family=binomial(link='logit'))
# summarize the fit
print(fit)
# make predictions
probabilities <- predict(fit, PimaIndiansDiabetes[,1:8], type='response')
predictions <- ifelse(probabilities > 0.5,'pos','neg')
# summarize accuracy
table(predictions, PimaIndiansDiabetes$diabetes)
```

Logistic Regression contd...

The glm algorithm can be used in caret as follows:

```
# load packages
library(caret)
library(mlbench)
# Load the dataset
data(PimaIndiansDiabetes)
# train
set.seed(7)
trainControl <- trainControl(method="cv", number=5)
fit.glm <- train(diabetes~., data=PimaIndiansDiabetes, method="lm", metric="Accuracy",
    preProc=c("center", "scale"), trControl=trainControl)
# summarize fit
print(fit.glm)
```

Example of logistic regression algorithm in caret.

Support Vector Machine

The ksvm() function is in the kernlab package and can be used for classification or regression. It is a wrapper for the LIBSVM software package and provides a suite of kernel types and configuration options. These examples uses a Radial Basis kernel.

Classification Example:

```
# load the packages
library(kernlab)
library(mlbench)
# Load the dataset
data(PimaIndiansDiabetes)
# fit model
fit <- ksvm(diabetes~., data=PimaIndiansDiabetes, kernel="rbfdot")
# summarize the fit
print(fit)
# make predictions
predictions <- predict(fit, PimaIndiansDiabetes[,1:8], type="response")
# summarize accuracy
table(predictions, PimaIndiansDiabetes$diabetes)
```

Example of SVM algorithm for classification.

Support Vector Machine contd...

Regression Example:

```
# load the packages
library(kernlab)
library(mlbench)
# load data
data(BostonHousing)
# fit model
fit <- ksvm(medv~., BostonHousing, kernel="rbfdot")
# summarize the fit
print(fit)
# make predictions
predictions <- predict(fit, BostonHousing)
# summarize accuracy
mse <- mean((BostonHousing$medv - predictions)^2)
print(mse)
```

Example of SVM algorithm for regression.

K-Means Clustering

- Clustering refers to grouping of data points based on similarity criteria, e.g. group customers based on similar purchase behavior
- Similarity can be measured in multiple dimensions, e.g. for customers, dimensions can items purchased, total spend etc.
- Goal of clustering is to assign a right cluster to each data point.
- The objective is to minimize sum squared error of all data points – error of a data point is distance w.r.t center of its cluster

$$J = Min \sum_{i=1}^{k} \sum_{x \in c_i} \|x - \mu_i\|_2^2$$

K-Means Clustering Algorithm

- Choose a number of clusters k – iterative method or standard algorithms.
- Initialize cluster centers $\mu 1, \ldots \mu k$
  - Could pick k data points and set cluster centers to these points.
  - Or could randomly assign points to clusters and take means of clusters.
- For each data point, compute the cluster center it is closest to (using some distance measure) and assign the data point to this cluster.
- Re-compute cluster centers (mean of data points in cluster).
- Stop when there are no new re-assignments.

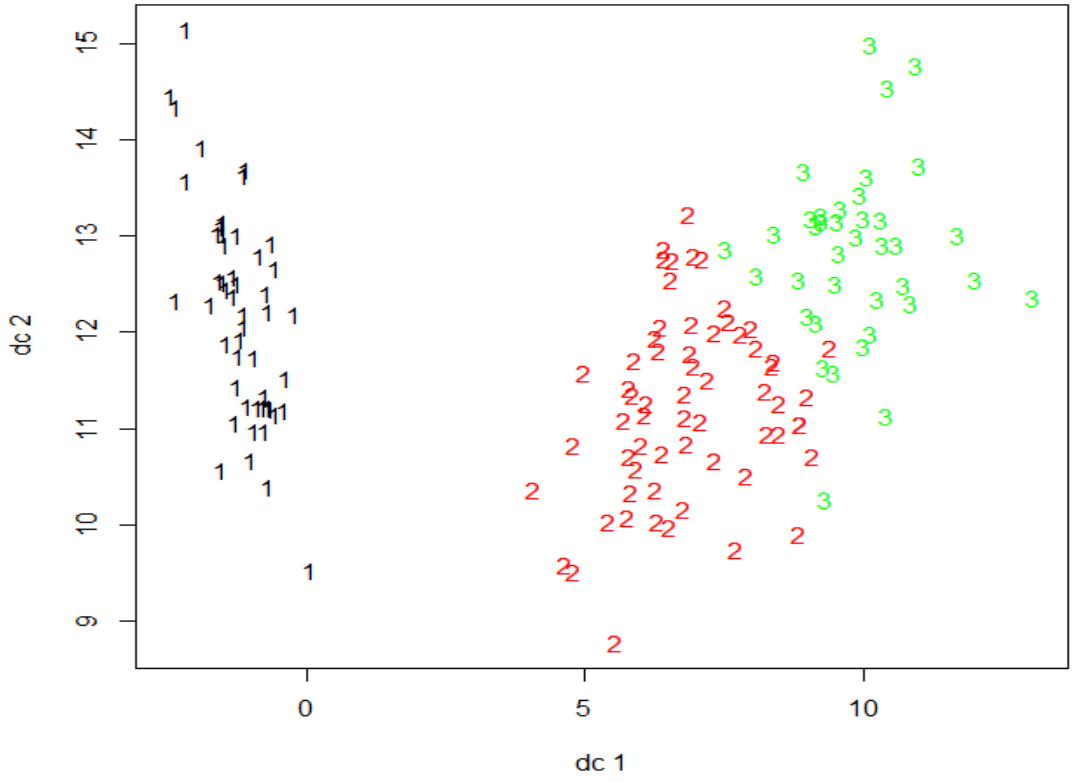
K-Means Clustering using R

```
#Loading data into memory - don't take the last column,as it has labels
sample.data<-iris[,-ncol(iris)]
summary(sample.data)
colnames(sample.data)
#K-Means Clustering
fit2 <- kmeans(sample.data, 3)
# get cluster means
Cluster.means<-aggregate(sample.data,by=list(fit2$cluster),FUN=mean)
# Total sum squares
Global.mean<-apply(sample.data,2, mean)
# Within sum squares
# Between Sum Squares
```

K-Means Clustering using R contd...

```
$cluster
  [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 [42] 3 2 2 2 2 2 2 2 2 4 4 4 3 1 1 4 3 1 3 3 1 3 1 1 4 1 3 3 3 4 1 1 1 1 1 4 1 3 3 3
 [83] 3 1 1 4 4 3 1 3 3 1 3 3 3 1 1 1 3 1 4 1 5 1 4 5 3 5 1 5 4 1 4 1 1 4 4 5 5 3 4 1 5
[124] 1 4 5 1 1 4 5 5 5 4 1 1 5 4 4 1 4 4 4 1 4 4 4 1 4 4 1

$centers
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1    0.2527555  -0.5360569    0.54703743  0.49112094
2   -0.9987207   0.9032290   -1.29875725 -1.25214931
3   -0.4201099  -1.4246794    0.03924137 -0.05279511
4    0.8596404   0.1928251    0.85201975  1.05041603
5    1.9704545   0.1552464    1.43998331  1.11601237

$totss
[1] 596

$withinss
[1] 19.10964 40.12172 17.04641 14.59610 12.01367

$tot.withinss
[1] 102.8875

$betweenss
[1] 493.1125

$size
[1] 38 49 22 29 12

$iter
[1] 3

$ifault
[1] 0
```

Visualization of K-Means Cluster - Results

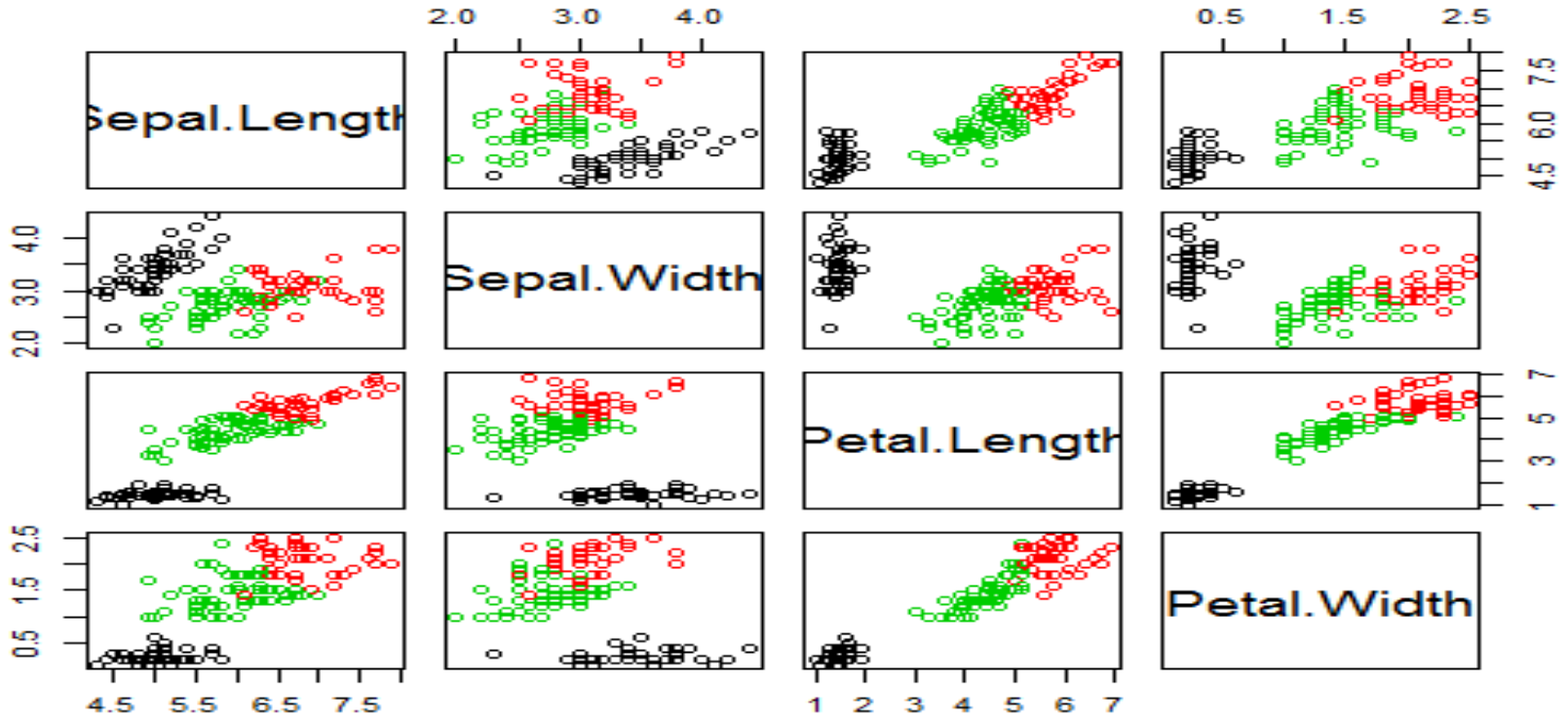


- Library(cluster)
- Library(fpc)
- Clus<-kmean(dat, centers=3)
- Plotcluster(data, clus$center)

- Uses Multidimensional scaling approach
- Multiple features of data observations are mapped to two dimensions for easy visualization

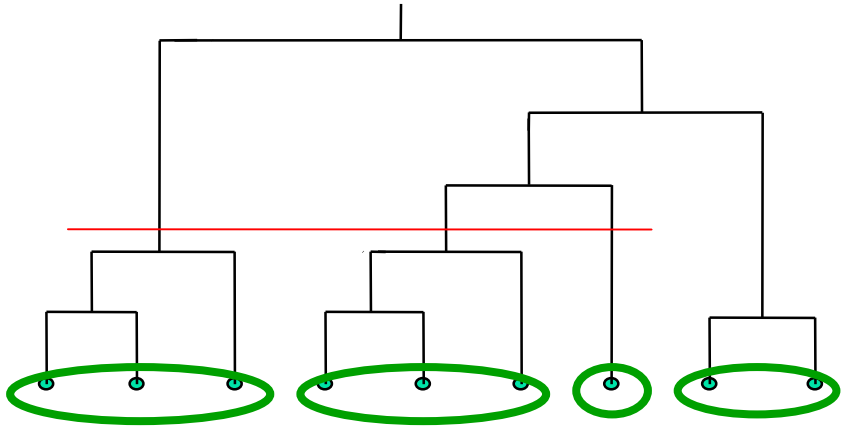
Visualization of K-Means Cluster – Results contd...

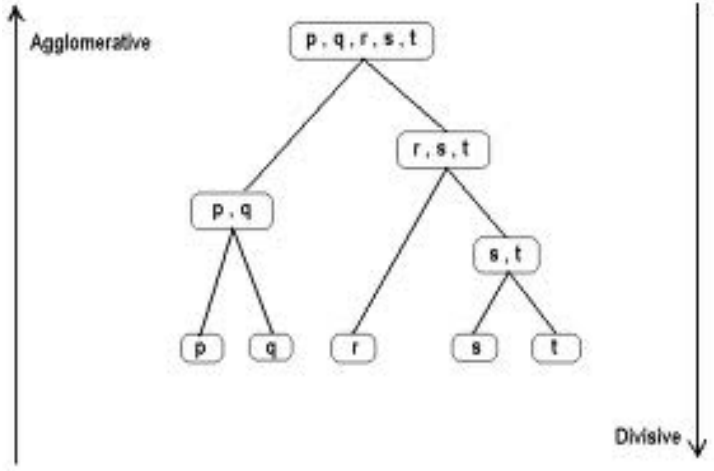
Relationships among various variables based on clustered observations



Hierarchical Clustering

- Build a tree-based hierarchical taxonomy (dendrogram) from a set of observations.
- Clustering obtained by cutting the dendrogram at a desired level: each connected component forms a cluster





Agglomerative (bottom-up)

- Start with each obs. being a single cluster
- Eventually all observations belong to the same cluster

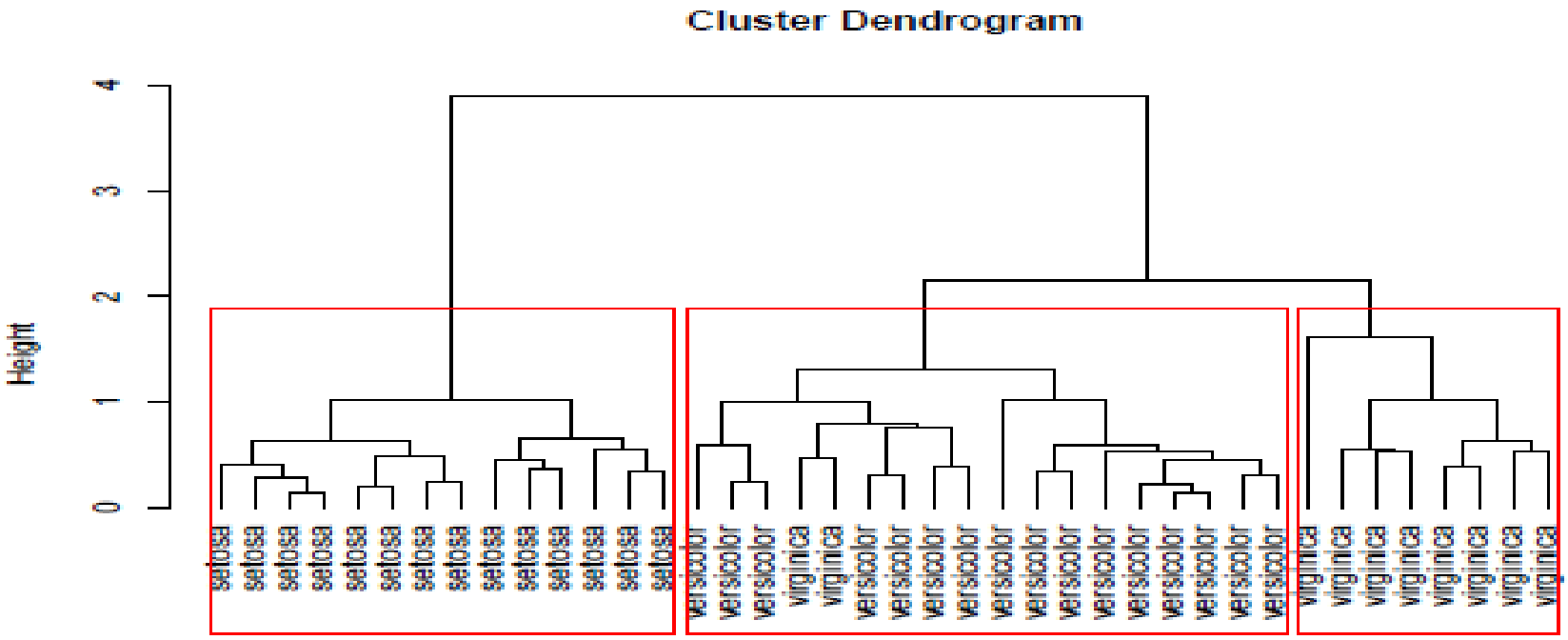
Divisive (top-down)

- Start with all obs. belonging to the same cluster
- Eventually each node forms a cluster on its own.

Hierarchical Clustering in R

```
> idx <- sample(1:dim(iris)[1], 40)
> irisSample <- iris[idx,]
> irisSample$Species <- NULL
> hc <- hclust(dist(irisSample), method="ave")
```

```
> plot(hc, hang = -1, labels=iris$Species[idx])
> # cut tree into 3 clusters
> rect.hclust(hc, k=3)
> groups <- cutree(hc, k=3)
```



Classification Models

An emergency room measures a patient's vital signs such as blood pressure, respiratory rate, pulse, etc.)

–Problem: to predict high-risk patients and discriminate them from low-risk patients which can help to put a patient to ICU.

A financial credit can be approved to only financial worthy customers.

–Problem: to decide whether an application should approved or not based on data such as age, job profession, earning capacity, previous barrowing history etc.

Advertisers get $$ for display ads only if users click them.

–Problem: Predict whether a user clicks a mobile display if by considering various ad parameters such as number of works, pictures, message etc.

Introduction to Classification Models

DEFINITIONS & CONCEPTS

- Classification models aims to identify the class to which new observations belong, on the basis of a training set of data containing observations whose class is known.

- It includes many techniques for modeling and analyzing several variables, when the focus is on the relationship between a Categorical dependent variable and one or more independent variables (can be numerical or categorical)

- Classification problems are widely applicable in real world scenarios

EXAMPLE 1

Whether or not a patient is cured of a disease, given the time of treatment, the dosage of medicine

EXAMPLE 2

Whether someone will default the loan or repay depends on his Credit history, repayment capacity etc.

Objective

- ✓ Predictive modeling is the analysis of data objects and identification of relationships among these data objects in a business context. Modeling involves finding good subsets of predictors or explanatory variables for a given ‘event’.

Few Modeling Techniques

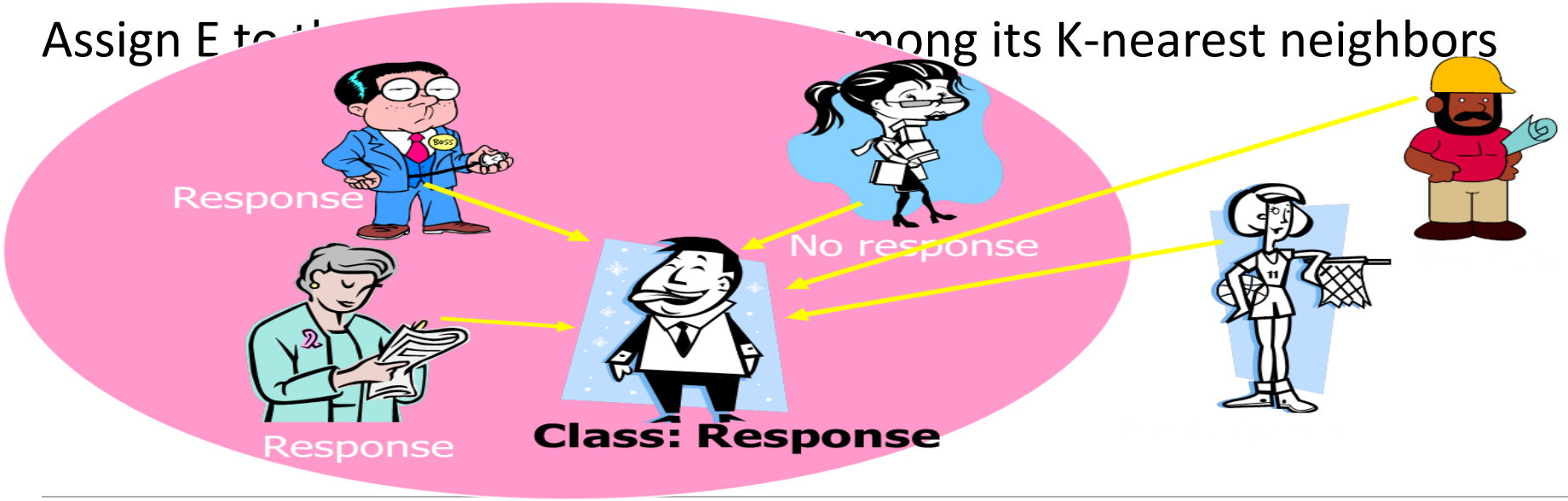
Linear Regression	Logistics Regression	Discriminant analysis	Artificial Neural Network
Support Vector Machines	Decision Trees	Random Forests	Boosting Trees
K-Nearest Neighbor	Bayesian networks	Bagging Methods	Hidden Markov Models

K Nearest Neighbors

- Learning by analogy -Tell me who your friends are and I'll tell you who you are
- A **new example** is assigned to the most common class among the (K) examples that are most similar to it
- No model is built: Store all training examples

To determine the class of a new example E:

- Calculate the distance between E and all examples in the training set
- Select K-nearest examples to E in the training set - “Closeness” is defined in terms of the *Euclidean* distance between two examples
- Assign E to t[illegible]among its K-nearest neighbors



Strengths

- Simple to implement & Use
- Comprehensive – easy to explain and predict
- Robust to noisy data by averaging k-nearest neighbors

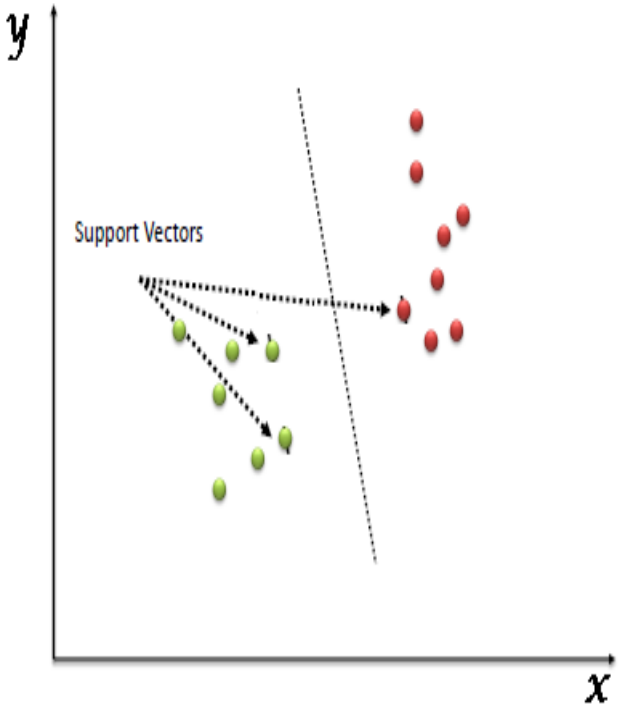
Weakness

- Need a lot of space to store all examples
- Takes more time to classify a new example than with a model
- Normalization of variables necessary

```
## prepare training data and label vector
train <- iris[-nrow(iris),-ncol(iris)]
cl <- iris[-nrow(iris),ncol(iris)]
# the object to be classified
test <-iris[nrow(iris),-ncol(iris)]

#installing and loading relevant package
install.packages("class")
library(class)

# call knn() and get its summary
summary(knn(train, test, cl, k = 20))
```

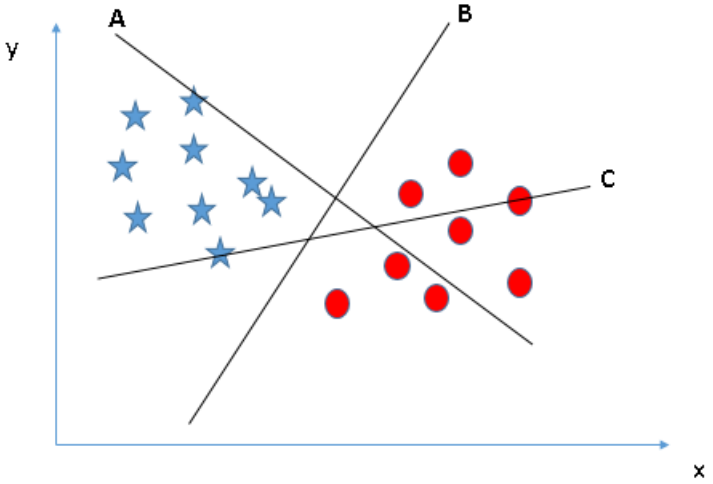
What is Support Vector Machine?

- “Support Vector Machine” (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges.
- However, it is mostly used in classification problems. In this algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate.
- Then, we perform classification by finding the hyper-plane that differentiate the two classes very well (look at the below snapshot).
- Support Vectors are simply the co-ordinates of individual observation. Support Vector Machine is a frontier which best segregates the two classes (hyper-plane/ line).



How does it work?

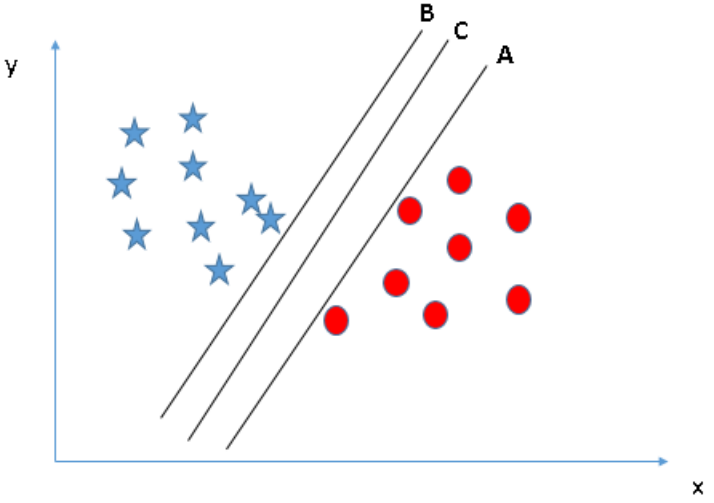
- Identify the right hyper-plane (Scenario-1): Here, we have three hyper-planes (A, B and C). Now, identify the right hyper-plane to classify star and circle.



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

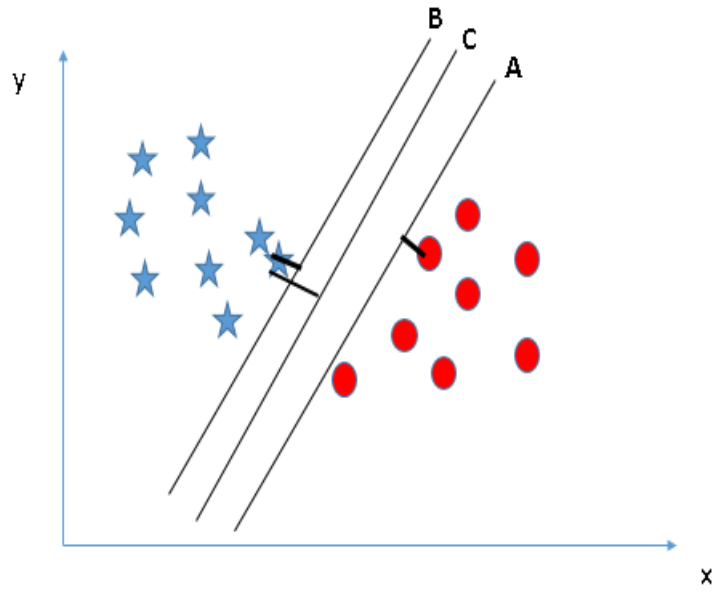
How does it work?

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



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

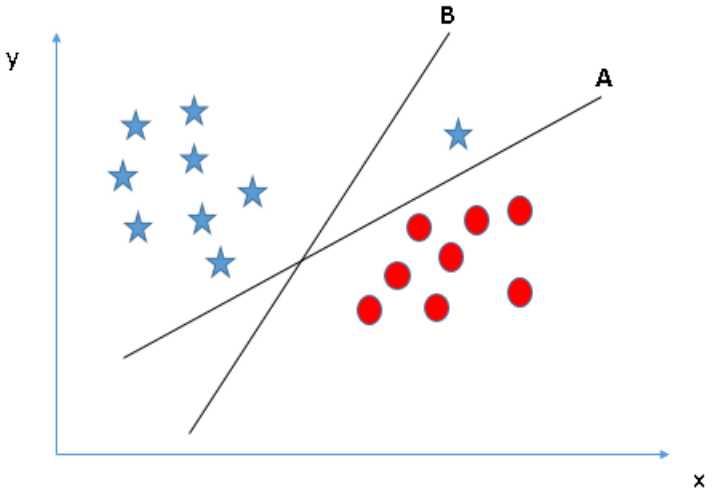
How does it work?



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

How does it work?

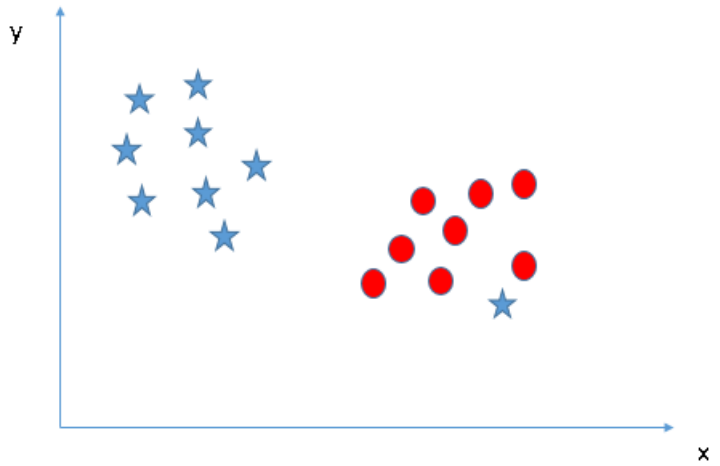
- Identify the right hyper-plane (Scenario-3):



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

How does it work?

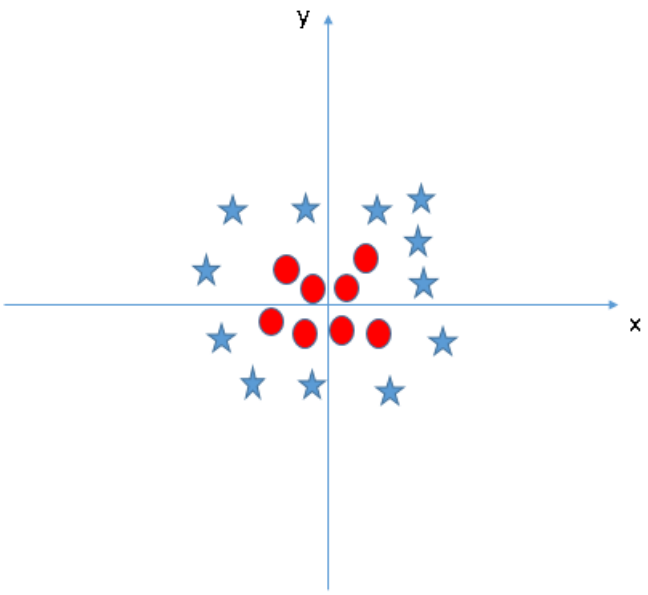
- Can we classify two classes (Scenario-4)?: Here, I am unable to segregate the two classes using a straight line, as one of star lies in the territory of other(circle) class as an outlier.



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

How does it work?

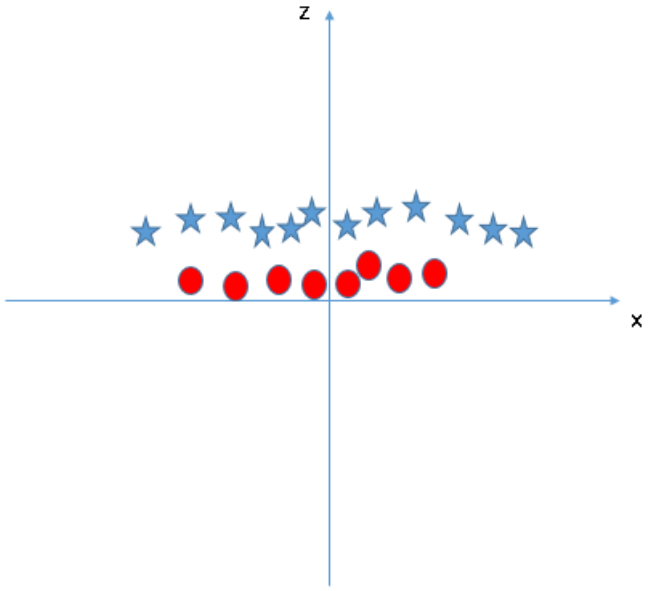
- Find the hyper-plane to segregate to classes (Scenario-5): In the scenario below, we can’t have linear hyper-plane between the two classes, so how does SVM classify these two classes? Till now, we have only looked at the linear hyper-plane.



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

How does it work?

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



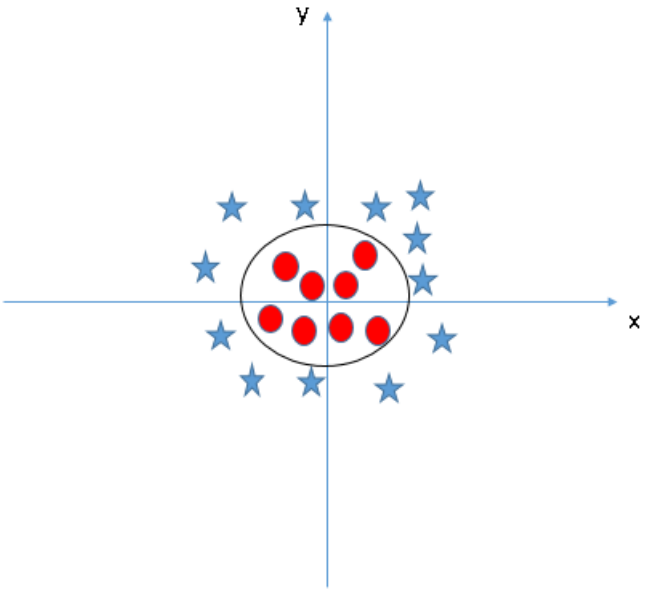
- All values for z would be positive always because z is the squared sum of both x and y
- In the original plot, red circles appear close to the origin of x and y axes, leading to lower value of z and star relatively away from the origin result to higher value of z.

How does it work?

- In SVM, it is easy to have a linear hyper-plane between these two classes.
- SVM has a technique called the kernel trick. These are functions which takes low dimensional input space and transform it to a higher dimensional space i.e. it converts not separable problem to separable problem, these functions are called kernels.
- It is mostly useful in non-linear separation problem. Simply put, it does some extremely complex data transformations, then find out the process to separate the data based on the labels or outputs you've defined.

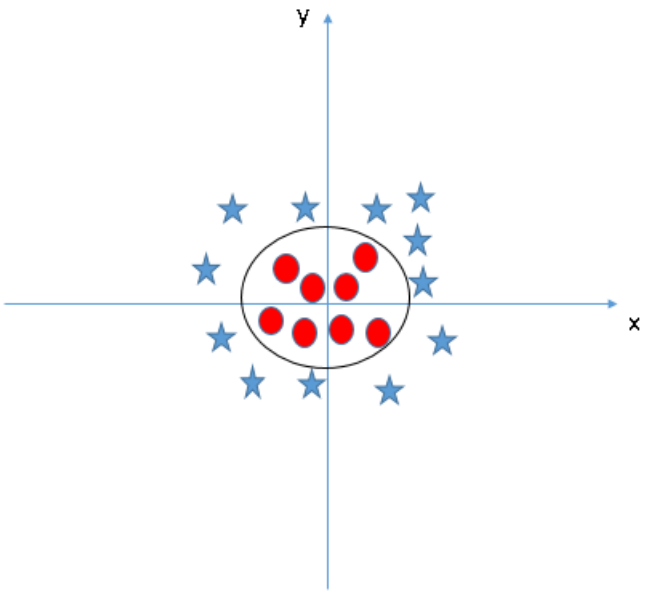
How does it work?

- When we look at the hyper-plane in original input space it looks like a circle:



How does it work?

- When we look at the hyper-plane in original input space it looks like a circle:



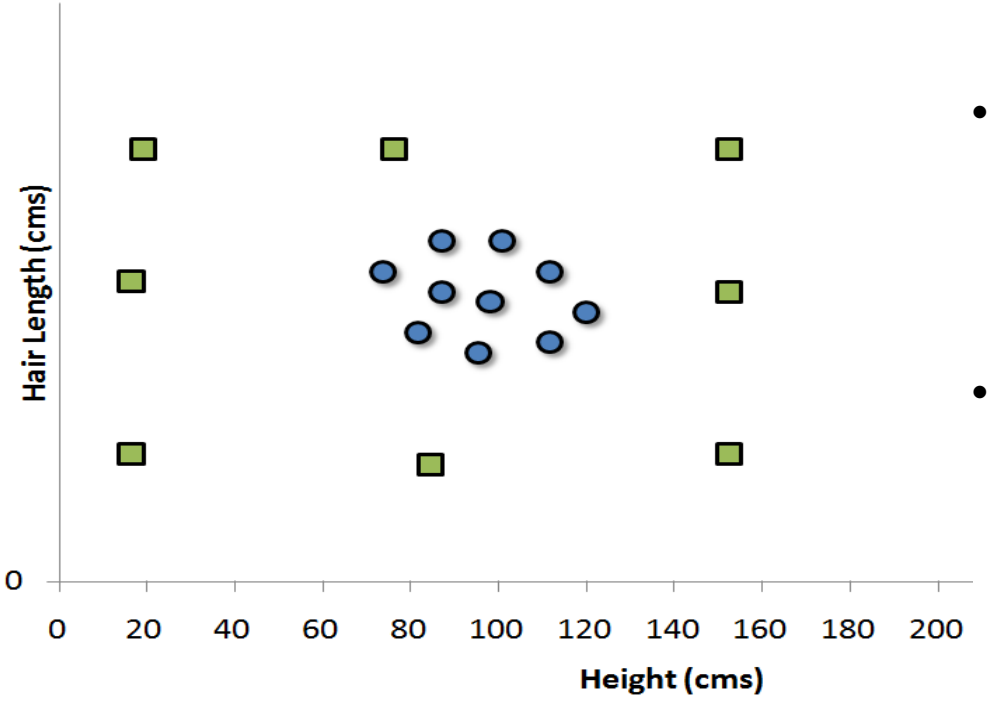
kernel: we have various options available with kernel like, “linear”, “rbf”,”poly” and others (default value is “rbf”). Here “rbf” and “poly” are useful for non-linear hyper-plane.

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

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

How does it work?

- When we look at the hyper-plane in original input space:

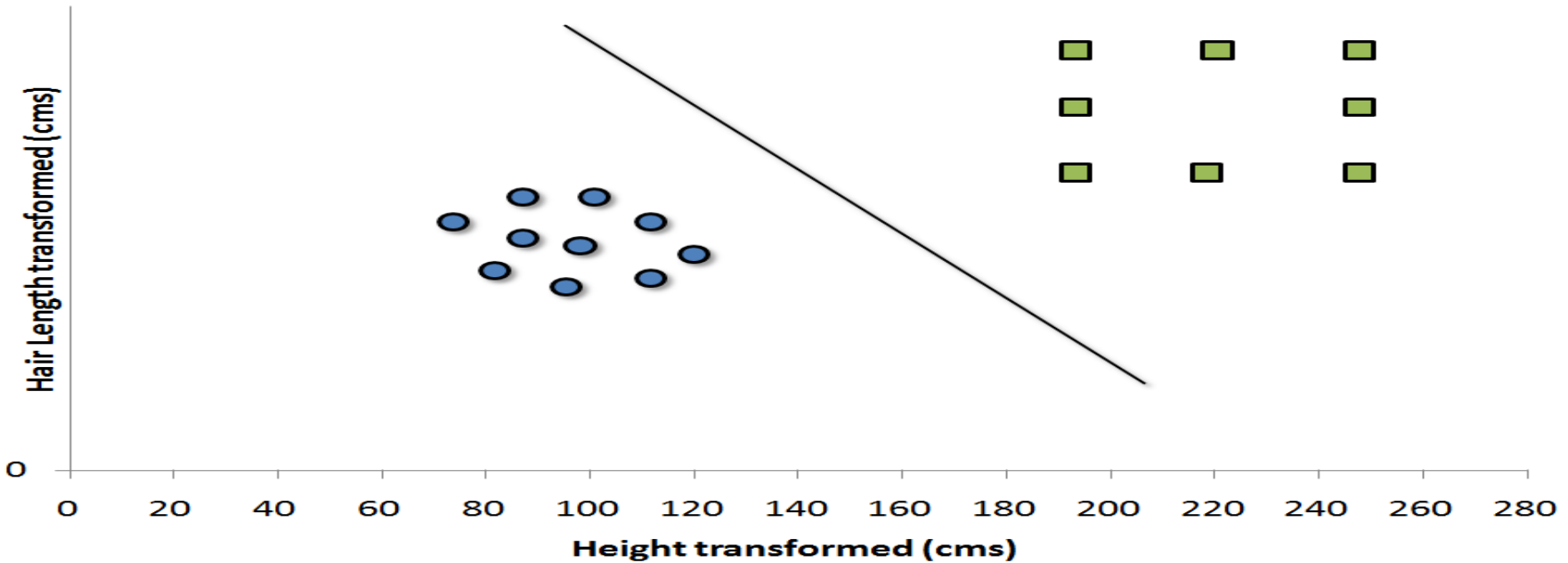


- In such cases, we do not see a straight line frontier directly in current plane which can serve as the SVM.

- In such cases, we need to map these vector to a higher dimension plane so that they get segregated from each other.

How does it work?

Each of the green square in original distribution is mapped on a transformed scale. And transformed scale has clearly segregated classes.



- **Advantages of SVM**
  1) SVM performs well in case of non-linear separable data using kernel trick.
  2) It works well in high dimensional space (i.e. large number of predictors)
  3) It works well in case of text or image classification.
  4) It does not suffer multicollinearity problem

- **Disadvantages of SVM**
  1) It takes a lot of time on large sized data sets
  2) It does not directly return probability estimates.
  3) The linear kernel is almost similar to logistic regression in case of linear separable data.

DECISION TREES

Building a single model considering all data observations may not give better predictive accuracy

Splitting data into multiple subsets and building separate models for each subset can help improve overall accuracy

Decision tree models help to identify the criteria to split the data in such a way to get the information gain (i.e. improvement in model accuracy)

The split criteria can be used as business rules and choose the right model for prediction

Suitable Conditions for Playing Golf

Predictors | Target

Outlook	Temp.	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No

- Would you make a Random choice of conditions or be more Analytical ??

		Play Golf	
		Yes	No
Outlook	Sunny	3	2
	Overcast	4	0
	Rainy	2	3

		Play Golf	
		Yes	No
Temp.	Hot	2	2
	Mild	4	2
	Cool	3	1

Play Golf	
Yes	No
9	5

		Play Golf	
		Yes	No
Humidity	High	3	4
	Normal	6	1

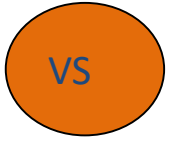
		Play Golf	
		Yes	No
Windy	False	6	2
	True	3	3

Information Gain - Example

Play Golf		Playing Golf	Decision	# of Correct Decisions
YES	NO			
9	5	9/14	YES	9

No Criteria - 9 Correct

Outlook criteria - 10 Correct



Improvement in number of correct decisions = 1

		Play Golf		Playing Golf	Decision	# of Correct Decisions
		YES	NO			
OUTLOOK	Sunny	3	2	3/5	YES	3
	Overcast	4	0	4/4	YES	4
	Rainy	2	3	2/5	NO	3

Some Hints for Analytical Minds

We should select an attribute to split the data in a way that we get maximum improvement in model accuracy

Error or disorder need to be reduced so that better decision making can be achieved

Information Gain is the expected reduction in disorder (measured through Entropy) caused by partitioning the data according to the given attribute

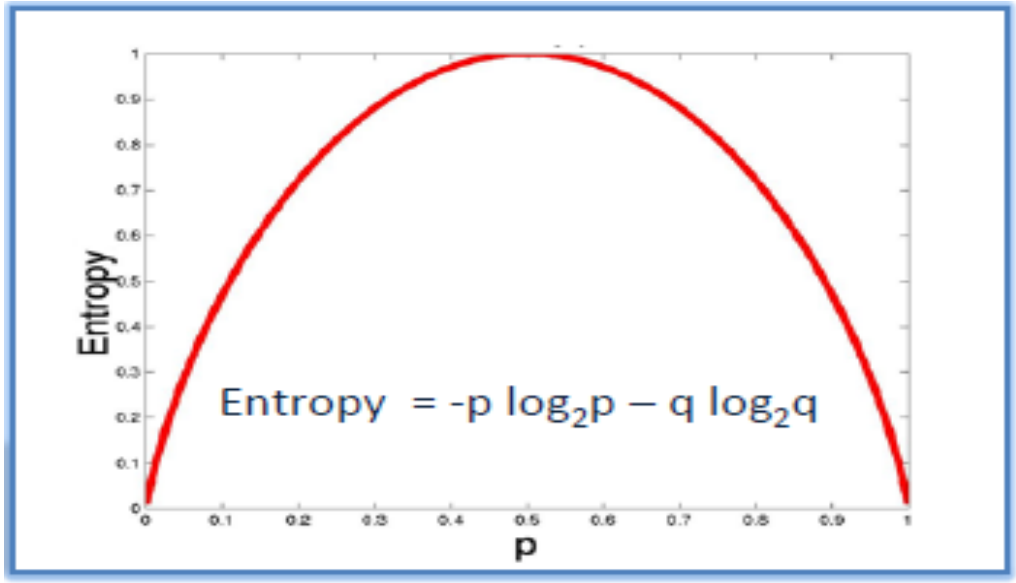
$IG(T, a) = H(T) - H(T|a)$, where

$H(T)$ & $H(T|a)$ are model accuracies without and with attributes

$IG(T, a)$ Information gain achieved with attribute a

Entropy - Measuring Information Gain

- Constructing a Decision tree is all about Information Gain - finding attribute that returns the highest information gain (i.e., the most homogeneous branches)



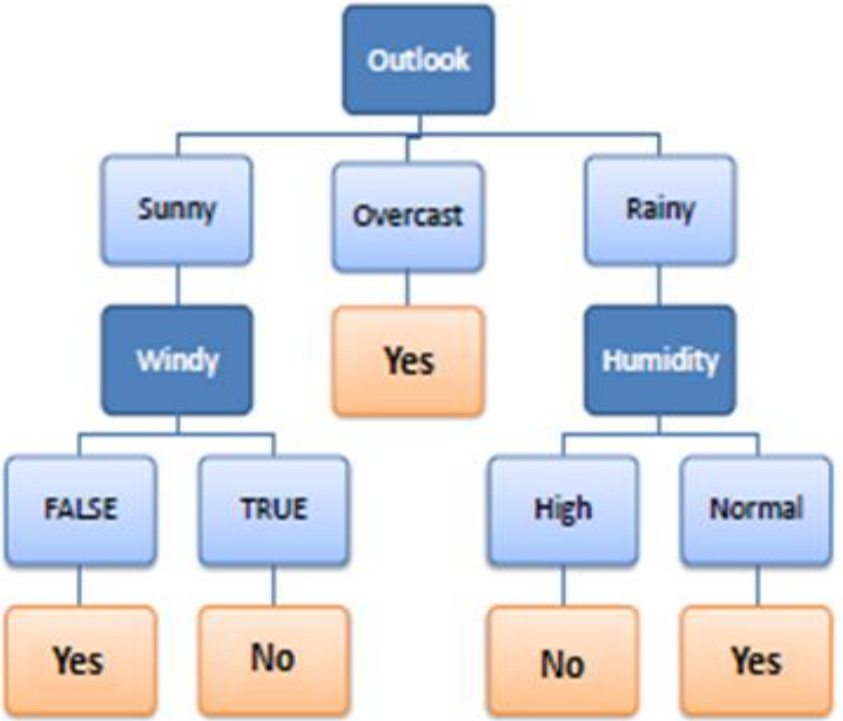
Entropy = $-0.5 \log_2 0.5 - 0.5 \log_2 0.5 = 1$

- Change in Entropy is a Measure of Information Gain
- If the sample is completely homogeneous the entropy is zero and if the sample is equally divided it has entropy of one.

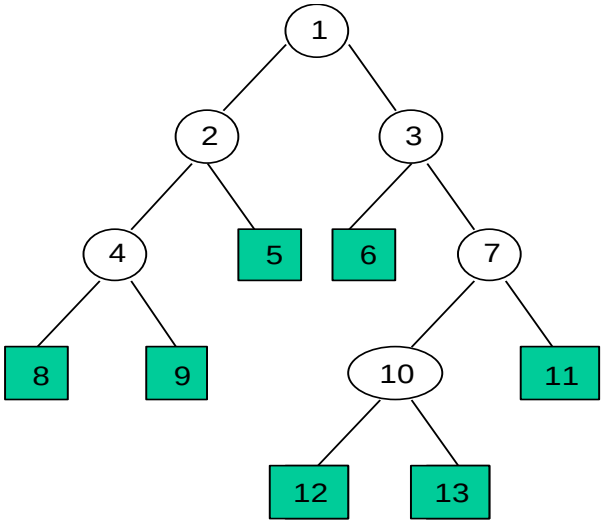
$$\sum_{i=1}^{m}$$

Information Gain & Decision Trees

A Decision Tree is a graphical representation of a sequence of Attributes that narrow down to a right model (e.g. decision to play or not-Play Golf)
The narrow down is achieved using the criterion of Information Gain



Decision Tree Terminologies



1 = root node

= terminal node

= Intermediate

e

- ***Root*** of Tree: Decision model based on entire data
- ***Node****:* Decision model on subset of data based on split criteria
- Typically a split criteria splits a node into two child nodes
- Terminal Node: No further splitting of a node.
- Models from terminal nodes are used for prediction
- Path from root to a terminal node indicates a business rule
- For prediction which ever business rule is satistified that particular model is used

Building a Decision Tree

Building a decision tree recursively splitting tree

–Stop criteria – no further splitting of data (less number of observations or information gain is less than threshold

Information gain for each attribute at a node is derived using entropy criteria.

For a splitting data at a node, identify an attribute that gives maximum information gain.

Entropy Calculations During Each Split

- To build a decision tree, we need to calculate two types of entropy using frequency tables

1. Entropy using the frequency table of one attribute:

$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

Play Golf	
Yes	No
9	5

Entropy(PlayGolf) = Entropy (5,9)
= Entropy (0.36, 0.64)
= - (0.36 $\log_2$ 0.36) - (0.64 $\log_2$ 0.64)
= 0.94

$$E(T, X) = \sum_{c \in X} P(c)E(c)$$

2. Entropy using the frequency table of two attributes:

		Play Golf		
		Yes	No	
Outlook	Sunny	3	2	5
	Overcast	4	0	4
	Rainy	2	3	5
				14

E(PlayGolf, Outlook) = **P**(Sunny)***E**(3,2) + **P**(Overcast)***E**(4,0) + **P**(Rainy)***E**(2,3)
= (5/14)*0.971 + (4/14)*0.0 + (5/14)*0.971
= 0.693

Information Gain Calculations for Each Attribute

- Derive information gain for each of the attributes
- An attribute information gain = Entropy of node – Entropy given attribute
- The result is the Information Gain, or decrease in entropy

		Play Golf	
		Yes	No
Outlook	Sunny	3	2
	Overcast	4	0
	Rainy	2	3
Gain = 0.247			

		Play Golf	
		Yes	No
Temp.	Hot	2	2
	Mild	4	2
	Cool	3	1
Gain = 0.029			

$$Gain(T, X) = Entropy(T) - Entropy(T, X)$$

G(PlayGolf, Outlook) = **E**(PlayGolf) – **E**(PlayGolf, Outlook)

= 0.940 – 0.693 = 0.247

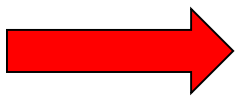
		Play Golf	
		Yes	No
Humidity	High	3	4
	Normal	6	1
Gain = 0.152			

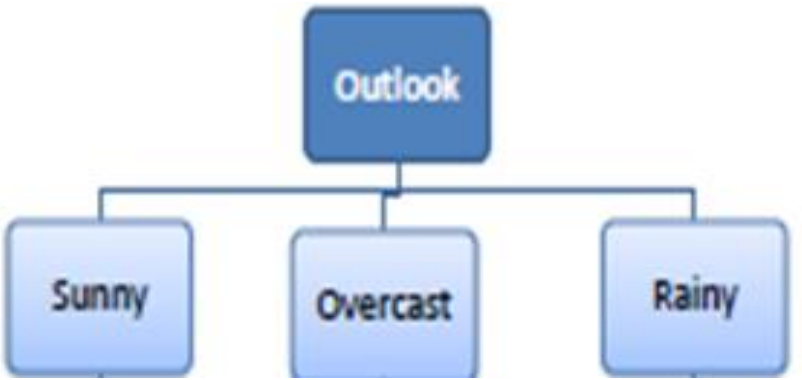
		Play Golf	
		Yes	No
Windy	False	6	2
	True	3	3
Gain = 0.048			

Identify an Attribute to Split a Node

- Choose attribute with the largest information gain as the criteria for splitting a node

★		Play Golf	
		Yes	No
Outlook	Sunny	3	2
	Overcast	4	0
	Rainy	2	3
Gain = 0.247			

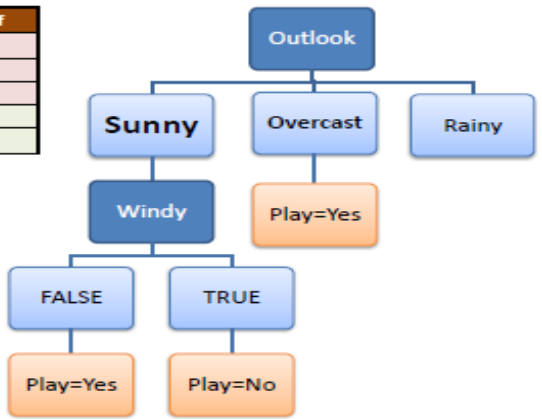




Build Decision Tree

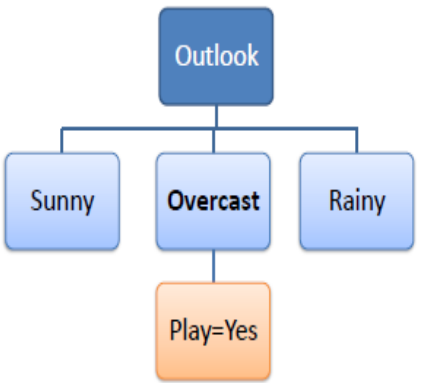
- Recursively partition the nodes to achieve improvements to models accuracy
- To decide if it's a node requires further splitting, use the following criteria
  - Number of observations in a node is more than threshold
  - Information gain achieved from a split is more than the threshold

Temp	Humidity	Windy	Play Golf
Mild	High	FALSE	Yes
Cool	Normal	FALSE	Yes
Mild	Normal	FALSE	Yes
Cool	Normal	TRUE	No
Mild	High	TRUE	No



Further split required

Temp	Humidity	Windy	Play Golf
Hot	High	FALSE	Yes
Cool	Normal	TRUE	Yes
Mild	High	TRUE	Yes
Hot	Normal	FALSE	Yes
Hot	High	FALSE	Yes



No further split needed

Output: Decision Tree to Decision Rules

- A decision tree can easily be transformed to a set of rules by mapping from the root node to the leaf nodes one by one

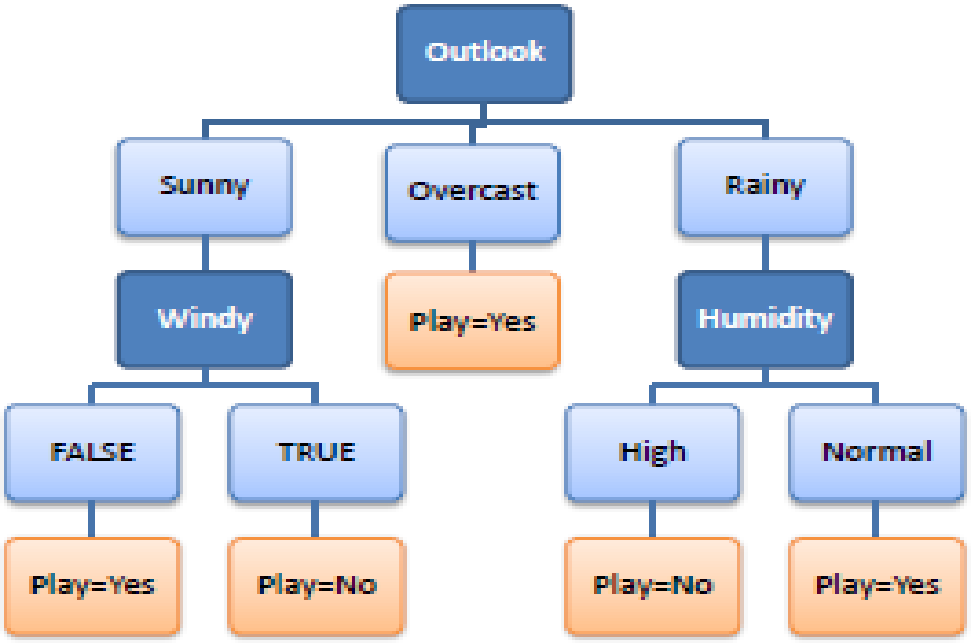
R_1: IF (Outlook=Sunny) AND (Windy=FALSE) THEN Play=Yes

R_2: IF (Outlook=Sunny) AND (Windy=TRUE) THEN Play=No

R_3: IF (Outlook=Overcast) THEN Play=Yes

R_4: IF (Outlook=Rainy) AND (Humidity=High) THEN Play=No

R_5: IF (Outlook=Rain) AND (Humidity=Normal) THEN Play=Yes



Different Tree building Algorithms

ID3 (Iterative Dichotomiser 3)
C4.5 (successor of ID3)
CART (Classification And Regression Tree)
CHAID (CHi-squared Automatic Interaction Detector). Performs multi-level splits when computing classification trees
MARS: extends decision trees to handle numerical data better
Conditional Inference Trees. Statistics-based approach that uses non-parametric tests as splitting criteria, corrected for multiple testing to avoid overfitting. This approach results in unbiased predictor selection and does not require pruning

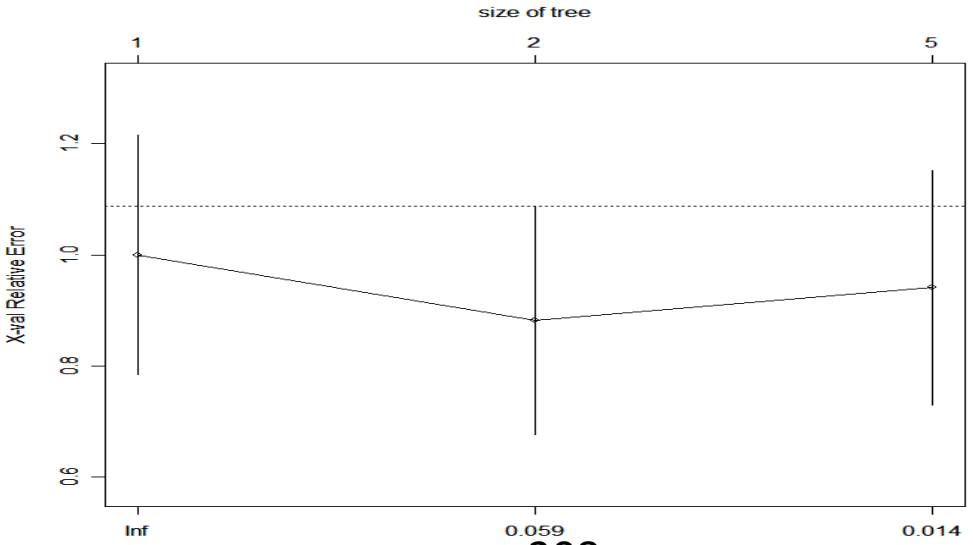
3 Steps for Choosing Right Decision Tree in CART

- Tree growing
  - Splitting rules to generate tree
  - Stopping criteria: how far to grow?
  - Missing values: using surrogates
- Tree pruning
  - Trimming off parts of the tree that don’t work
  - Ordering the nodes of a large tree by contribution to tree accuracy … which nodes come off first?
- Optimal tree selection
  - Deciding on the best tree after growing and pruning

Decision Trees: R example

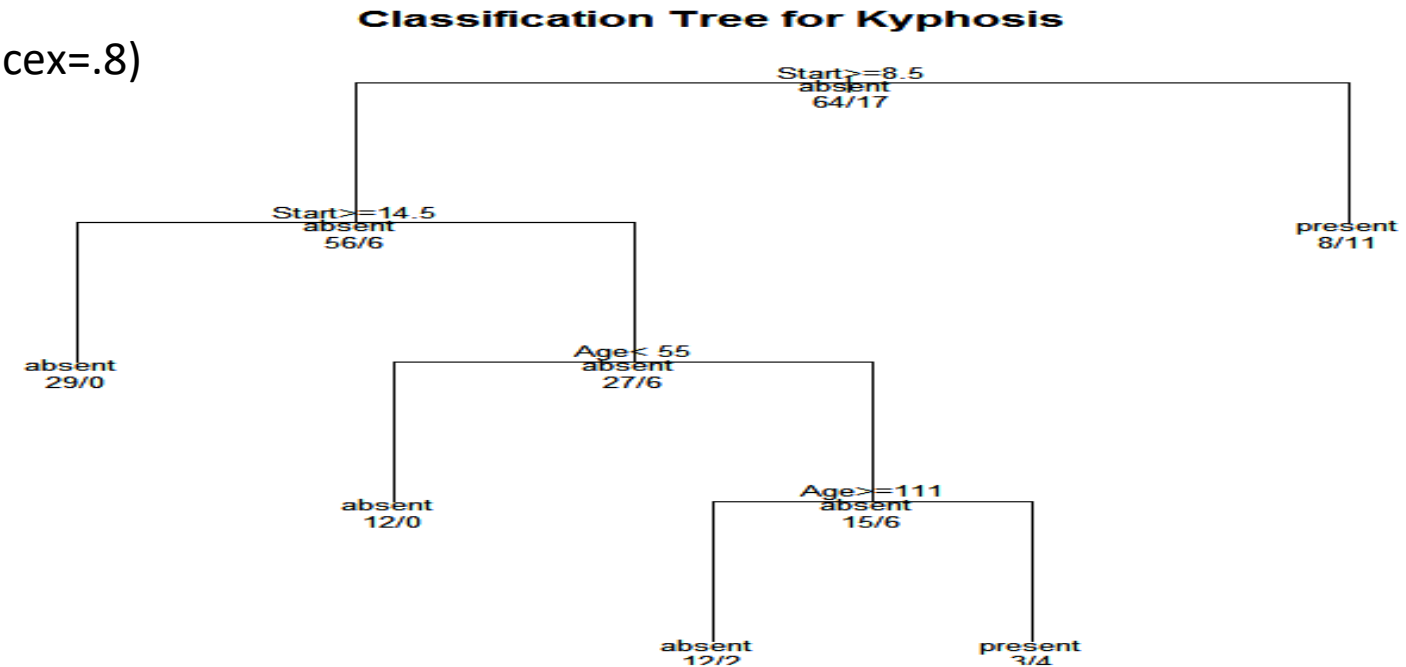
```
#DECISION TREES
install.packages("rpart")
library(rpart)


# grow tree
fit <- rpart(Kyphosis ~ Age + Number + Start, method="cl
```

```
printcp(fit) # display the results
plotcp(fit) # visualize cross-validation results
summary(fit) # detailed summary of splits
```



Decision Trees: R example

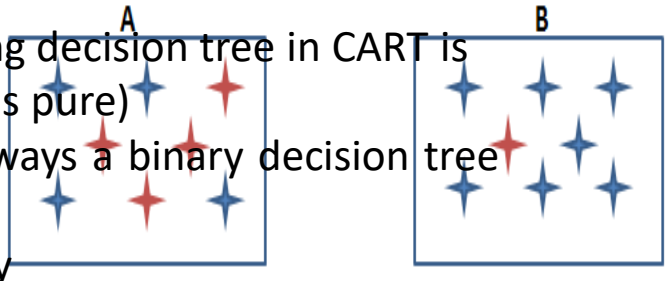
```
# plot tree
plot(fit, uniform=TRUE,  main="Classification Tree for Kyphosis")
text(fit, use.n=TRUE, all=TRUE, cex=.8)
```



GINI Index

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

- It works with categorical target variable "Success" or "Failure".
- The impurity (or purity) measure used in building decision tree in CART is Gini Index. (Here, Node A is impure and Node B is pure)
- The decision tree built by CART algorithm is always a binary decision tree (each node will have only two child nodes)
- Higher the value of Gini higher the homogeneity



Steps to Calculate Gini for a split:

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

Actual working on GINI

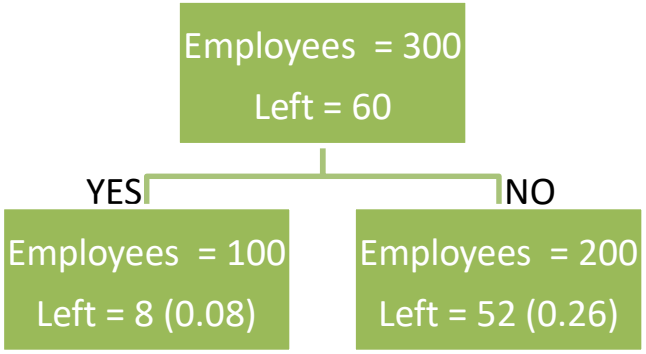
➢Split on **Promotion in last 5 years:**

Here, p = 8/100 = 0.08

Gini for sub-node “Yes” = (0.08)^2+(1-0.08)^2 = 0.8528

Here, p = 52/200 = 0.26

Gini for sub-node “No” = (0.26)^2+(1-0.26)^2 = 0.6152



Weighted Gini for Promotion in last 5 years

= (100/300)* 0.8528 +(200/300)* 0.6152

= **0.6944**

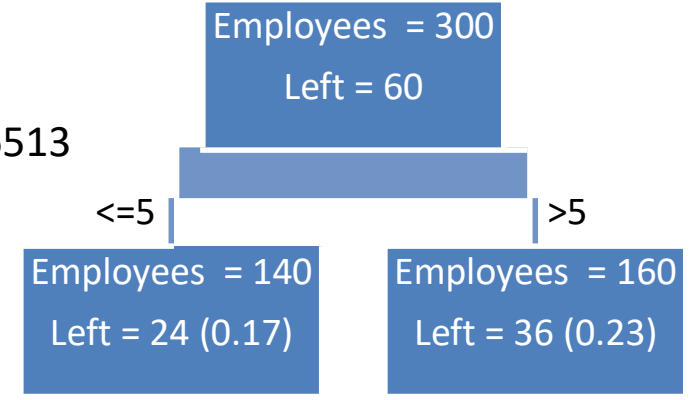
➢Split on **Projects completed:**

Here, p = 24/140 = 0.17

Gini for sub-node “<=5” = (0.17)^2+(1-0.17)^2 =

0.7159 Here, p = 36/160 = 0.23

Gini for sub-node “>5” = (0.23)^2+(1-0.23)^2 = 0.6513



Weighted Gini for Promotion in last 5 years

= (140/300)* 0.7159 +(160/300)* 0.6513

= **0.6814**

Actual working on GINI

Split on **Satisfaction Level:**

Here, p = 20/120 = 0.17

Gini for sub-node “<=0.5” = (0.17)^2+(1-0.17)^2 = 0.7222

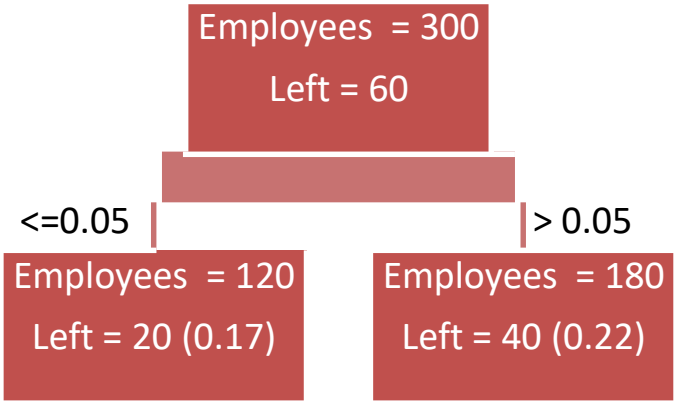
Here, p = 40/180 = 0.22

Gini for sub-node “>0.5” = (0.22)^2+(1-0.22)^2 = 0.6543

Weighted Gini for Promotion in last 5 years

= (120/300)* 0.7222 +(180/300)* 0.6543

= **0.6815**



We can see that Gini score for ***Promotion in last 5 years*** is higher than remaining variables, so the node split will take place on **Promotion in last 5 years**.

Chi Square

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

- It works with categorical target variable “Success” or “Failure”.
- It can perform two or more splits.
- Chi-Square of each node is calculated using formula:

  Chi-square = ((Actual – Expected)^2 / Expected)^1/2
- It generates tree called CHAID (Chi-square Automatic Interaction Detector)
- Higher the value of Chi-Square higher the statistical significance of differences between sub-node and Parent node.

Steps to Calculate Chi-square for a split:

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

Actual working on Chi Square

- Split on '**Promotion in last 5 years**'

Node	Left	Stay	Total	Exp Left	Exp Stay	Deviation Left	Deviation Stay	Chi sqr	
								Left	Stay
Yes	8	92	100	50	50	-42	42	5.939697	5.939697
No	52	148	200	100	100	-48	48	4.8	4.8
							Total chi sqr	**21.47939392**	

- Split on '**Projects completed**'

Node	Left	Stay	Total	Exp Left	Exp Stay	Deviation Left	Deviation Stay	Chi sqr	
								Left	Stay
<=5	24	116	140	70	70	-46	46	5.498052	5.498052
>5	36	124	160	80	80	-44	44	4.91935	4.91935
							Total chi sqr	**20.83480231**	

- Split on '**Satisfaction Level**'

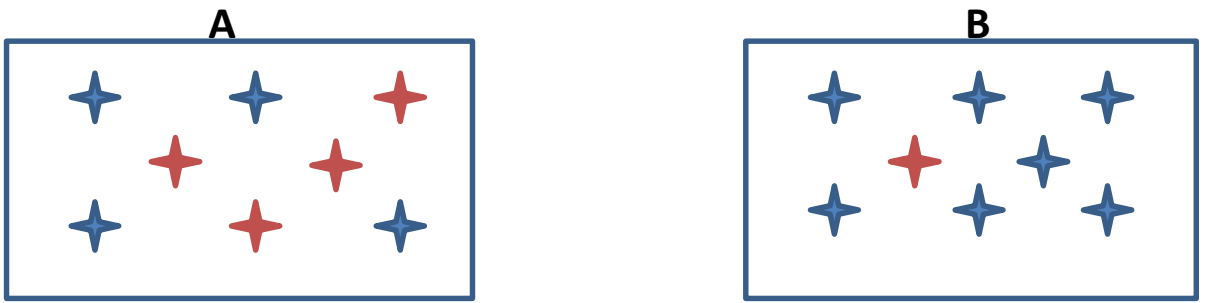
Node	Left	Stay	Total	Exp Left	Exp Stay	Deviation Left	Deviation Stay	Chi sqr	
								Left	Stay
<=0.5	20	100	120	60	60	-40	40	5.163978	5.163978
>0.5	40	140	180	90	90	-50	50	5.270463	5.270463
							Total chi sqr	**20.86888112**	

As observed, that Chi-square also identify the **Promotion in last 5 years** split is more significant compare to other variables as its chi square value is higher.

Information and Entropy

Information Gain

By using information gain as a criterion, we try to estimate the information contained by each attribute. Look at the image below,



These are 2 nodes A and B. We can easily explain node B as it requires less information because it has more of blue stars and can be classified as 'Blue Star' category, whereas in Node A it is equal mixture of both blue and red. Node B is called as a pure node/ homogenous node.

Constructing a decision tree is all about finding attribute that returns the highest information gain (i.e., the most homogeneous branches).

Entropy

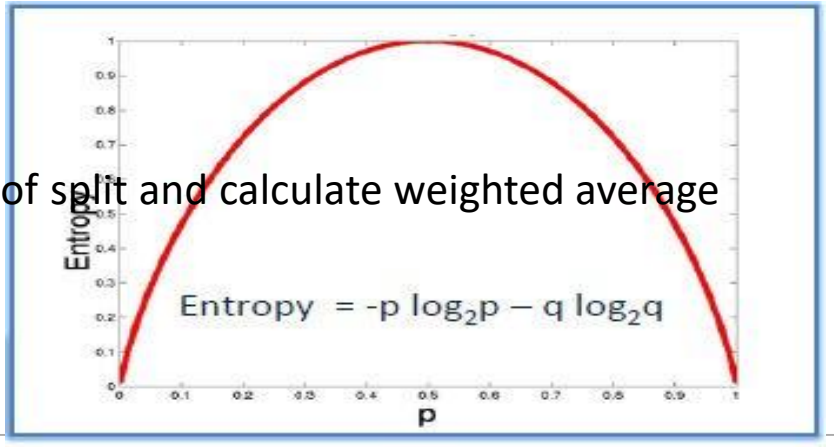
To measure the randomness or uncertainty of a random variable X is defined by **Entropy**. If the sample is completely homogeneous the entropy is zero and if the
sample is an equally divided it has entropy of one. Entropy is calculated as,
Entropy = -p log2 p – q log2 q

Here p and q is probability of success and failure respectively in that node. The lesser the entropy, the better it is.

Steps to calculate entropy for a split:

•Calculate entropy of parent node

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



Entropy = $-0.5 \log_2 0.5 - 0.5 \log_2 0.5 = 1$

Actual working on Entropy

- Split on **Promotion in last 5 years:**

Entropy for 'YES' node = -(8/100) log2 (8/100) – (92/100) log2 (92/100) = 0.4022
and
For 'NO' node = -(52/200) log2 (52/200) – (148/200) log2 (148/200) = 0.8267

Weighted entropy of sub-nodes = (100/300)* 0.4022 + (200/300)* 0.8267 = **0.6852**

- Split on '**Projects completed**'

Entropy for '<= 5' node = -(24/140) log2 (24/140) – (116/140) log2 (116/140) = 0.6610 and
For '> 5' node = -(36/160) log2 (36/160) – (124/160) log2 (124/160) = 0.7692

Weighted entropy of sub-nodes = (140/300)* 0.6610 + (160/300)* 0.7692 = **0.7187**

Actual working on Entropy

- Split on '**Satisfaction Leve**l'

Entropy for '<= 0.5' node = -(20/120) log2 (20/120) – (100/120) log2 (100/120) = 0.6500 and
For '> 0.5' node = -(40/180) log2 (40/180) – (140/180) log2 (140/180) = 0.7642

Weighted entropy of sub-nodes = (120/300)* 0.6500 + (180/300)* 0.7642
= **0.7185**

Above, we see that entropy for *Split on* ***Promotion in last 5 years*** is the lowest among all, so the tree will split on ***Promotion in last 5 years***.

We can derive information gain from entropy as **1- Entropy.**
Information Gain for Promotion in last 5 years = 1 – 0.6852 = 0.3148

Decision Trees Strengths and Weekness

- No assumptions are required about the distribution of data.
- The explanatory variables can be a mixture of categorical, interval and continuous.
- Especially good for missing variables, high-dimensional or large data sets.
- Produce useful results by using a few important variables.
- Unaffected by outliers, collinearities or heteroscedascity.
- Results are easy to interpret.
- An important weakness: Not based on a probabilistic model, no confidence interval.

Ensemble Methods

Einstein or the crowd sourcing

Ask an Expert

Audience Poll

Let consider a group of M voters.

If each voter has an independent probability $p > \frac{1}{2}$ of voting for the correct decision, then adding more voters increases the probability of the majority decision to be correct.

When $M \to \infty$, the probability that the decision taken by the group is correct approaches 1.

Marquis de Condorcet
French philosopher, mathematician, and political scientist

N GOOD MODELS

SINGLE PREDICTION

Multiple Models used together provide better predictive performance than could be obtained from any of the constituent single models

Statistics?

Branch of mathematics dealing with

- **Data collection**
- **Organization**
- **Analysis**
- **Interpretation** and **presentation**.

Statistically you can analyze the data using two methods,

Descriptive statistics: Summarizing data from a sample using indexes such as the mean or standard deviation

Inferential statistics : Drawing conclusions from data that are subjected to random variation (e.g., observational errors, sampling variation).

Descriptive Statistics

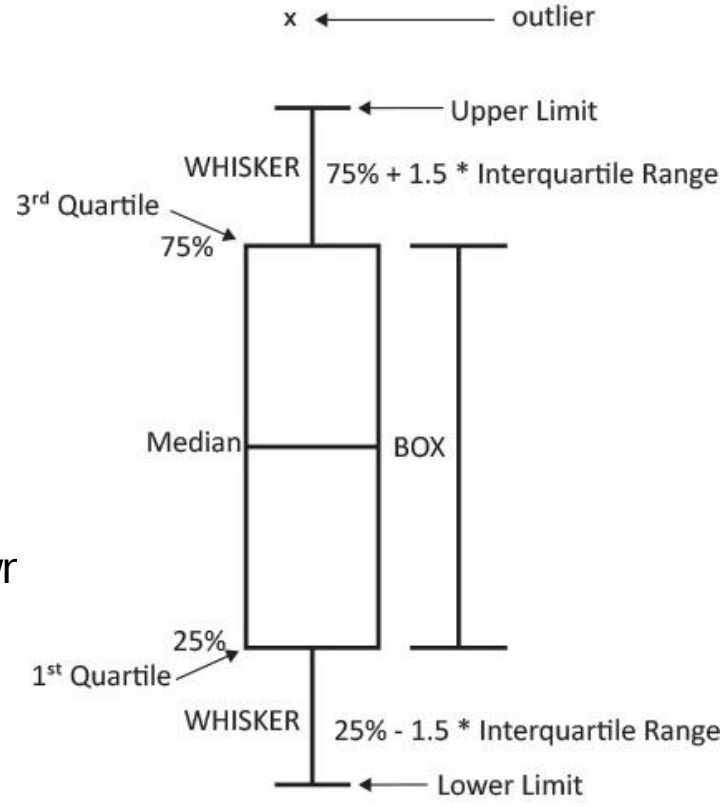
- **Measures of central tendency** – mean, median, mode
- **Measures of dispersion** – range, variance, standard deviation
- **Measures of shape** – skewness, kurtosis

Inferential Statistics

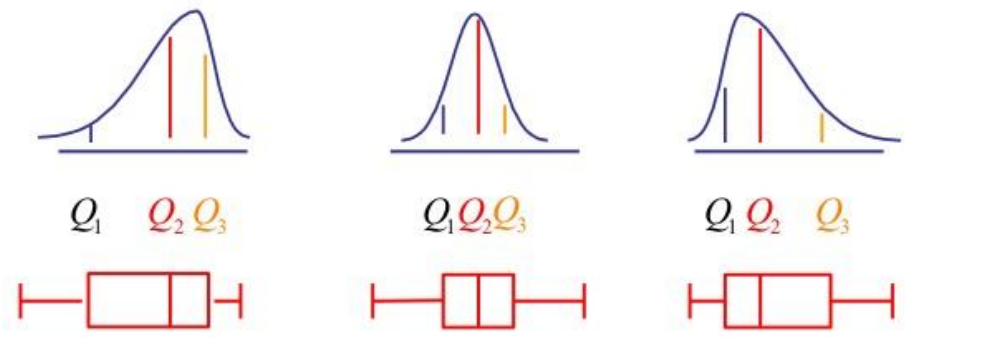
Inferential statistical analysis infers properties of a population, for example by testing hypotheses and deriving estimates.

Box and Whisker Plot

- **Box-and-whisker plots** are a handy way to display data broken into summary statistics by dividing data into four quartiles, each with an equal number of data values. It shows where the middle of the data, maximum and minimum lies.
- The first quartile represents the point where 25% of the data is below it.
- The median is the middle value of the data where half of the points are above and half are below this value.
- The third quartile represents the point where 75% of the data is below it.
- The whisker extends up to the highest value of upper limit and dowr to the lowest value of the lower limit.
- The lowest point of the lower whisker is called the lower limit. It equals Q1 – 1.5 * (Q3-Q1 or interquartile range).
- The highest point of the upper whisker is the called the upper limit. It equals Q3 + 1.5 * (Q3-Q1).
- Outliers are points that fall outside the limits of the whiskers.
- The interquartile is represented by the distance between Q1 and Q3.



Distribution Shape & Box-and-Whisker



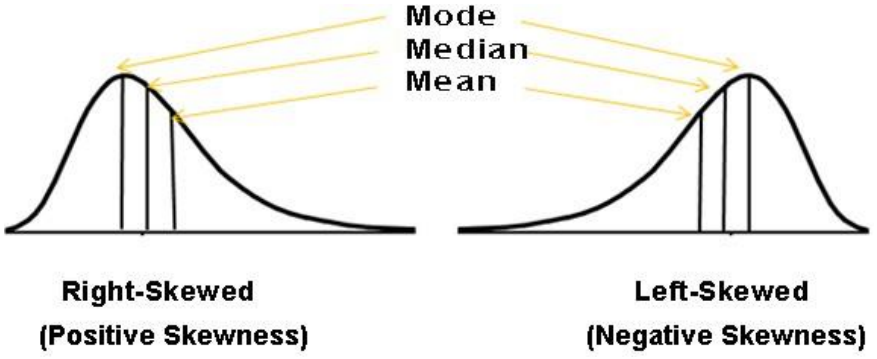
Measures of Shape

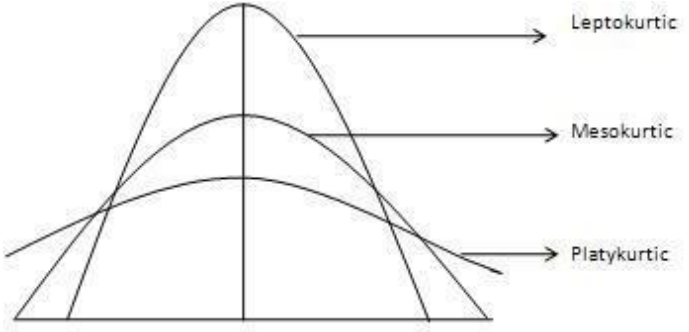
- Measures of shape describes the distribution or pattern of the data in a set
- The distribution shape of the quantitative data can be described as there is a logical order to the values and the low and high end values on the horizontal axis of the histogram
- The distribution shape of the qualitative data cannot be described.

Measures of shape are as follows:

✓ Degree of Skewness

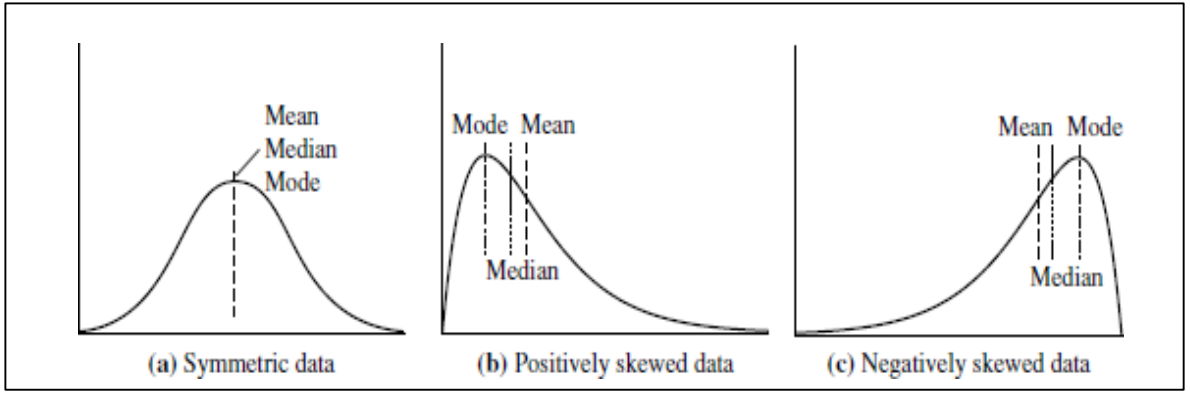
✓ Kurtosis





Degree of Skewness

- Skewness is the tendency for the values to be more frequent around the high or low ends of the x axis
- Skewness is a measure of symmetry
- **Symmetric data -** The data is symmetrically distributed on both side of medium
  - ✓ *mean = median =mode*
- **Positively skewed** -
  - ✓ Tail on the right side is longer than the left side.
  - ✓ *mode < median <mean*
- **Negatively skewed** -
  - ✓ Tail on the left side is longer than the right side.
  - ✓ *mode > median >mean*



Skewness Interpretation

- If skewness is less than -1 or greater than 1, the distribution is highly skewed.
- If skewness is between -1 and -0.5 or between 0.5 and 1, the distribution is moderately skewed.
- If skewness is between -0.5 and 0.5, the distribution is approximately symmetric, close to Normal Distribution.

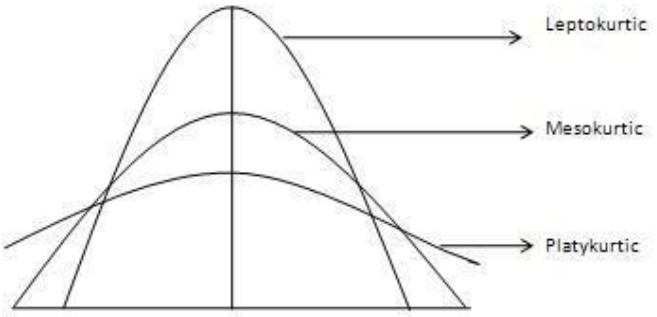
Kurtosis

- Kurtosis is the sharpness of the peak of a frequency-distribution curve.
- It describes the shape of the distribution of the tail's in relation to its shape

Types of Kurtosis

✓ Mesokurtic – It has flatter tail than standard normal distribution and slightly lower peak
✓ Leptokurtic – It has extremely thick tail and a very thin and tall peak
✓ Platykurtic – It has slender tail and a peak that's smaller than Mesokurtic distribution

Kurtosis - Measure of the relative peak of a distribution.
K= 3 indicates a normal "bell-shaped" distribution (mesokurtic).
K < 3 indicates a platykurtic distribution (flatter than a normal distribution with shorter tails).
K> 3 indicates a leptokurtic distribution (more peaked than a normal distribution with longer tails).



Population and Sample

Population:

•A **population** is any large collection of objects or individuals, such as Working professionals, students, or house makers about which information is desired.

Example:

- ✓ Collection of items
- ✓ A group of people suffering from a particular disease,
- ✓ Collection of books,

Sample:

•Sample is the representative unit of the target population, which is worked upon by the researchers

Image credits : Towards DataScience

Conditional Probability

- In probability theory, **conditional probability** is a measure of the probability of an event (some particular situation occurring) given that (by assumption, presumption, assertion or evidence) another event has occurred

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

where:

- $P(A \mid B)$ = Conditional probability that event A will occur given that event B has occurred already
- $P(A \cap B)$ = Unconditional probability that event A and event B both occur
- $P(B)$ = Probability that event B occurs

Binomial Distribution

- The number of successes *x* in *n* repeated trials of a binomial experiment is called binomial random variable
  - ✓ Toss a coin it has only two outcome i.e. Head or Tail
  - ✓ Result of an exam has two outcomes, pass or fail

- The **binomial distribution** is a common discrete **distribution** used in statistics, as opposed to a continuous **distribution** such as the normal **distribution**. This is because the **binomial distribution** only counts two states, typically represented as 1 (for a success) or 0 (for a failure) given a number of trials in the data

  - ➢ The experiment consists of *n* repeated trials.
  - ➢ Each trial can result in just two possible outcomes – Success or Failure
  - ➢ The probability of success, denoted by *P*, is the same on every trial.
  - ➢ Independent trials i.e. the outcome on one trial does not affect the outcome on other trials

Example

- A company has 500 employees, salary of whom is normally distributed, with an average of Rs.40,000 and Standard deviation of Rs.6000. Suppose you pick a random employee from the 500 employees, what are chances he/she earns less than Rs.30,000

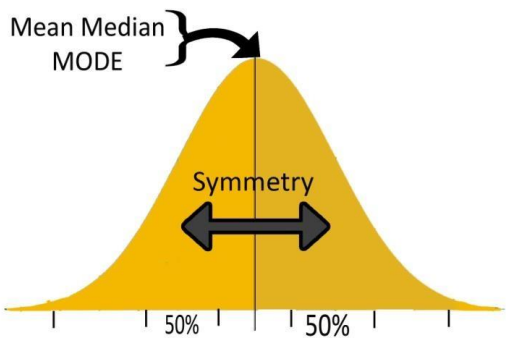
- The following information is available:

  Distribution: Normally distributed

  Mean: 40,000

  SD: Rs.6000

Make a bell curve showing mean and …..



Standard Scores or Zscore

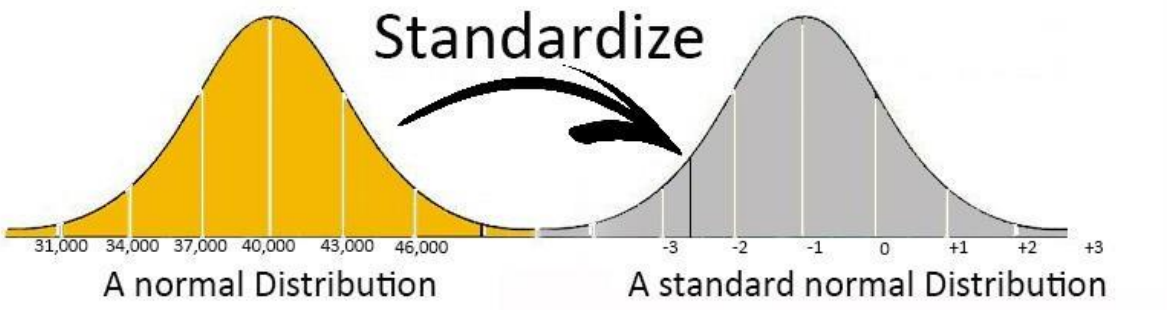
To find out the answer of previous questions, first of all we need to understand the standard scores or Z score

- The number of standard deviations from the mean is also called the "Standard Score", "sigma" or "z-score". Get used to those words!

$$z = \frac{X - \mu}{\sigma}$$

So to convert a value to a Standard Score ("z-score"):

- first subtract the observation from the mean: 30,000 – 40,000 = -10,000
- then divide by the Standard Deviation: -10,000/6000 = - 1.66
- And doing that is called "Standardizing":
- We can take any Normal Distribution and convert it to The Standard Normal Distribution.



Ztest

Since the z score was negative (-1.66), we need to use the negative Z- Scores

From the table, we can find out the probabilities of z score at -1.66

= 0.0485 0r 4.85%

It means that when we pick a random employee from the 500 employees, the chances he/she earns less than Rs.30,000 is 4.85%

z	.00	.01	.02	.03	.04	.05	.06	.07	.08	.09
−3.4	.0003	.0003	.0003	.0003	.0003	.0003	.0003	.0003	.0003	.0002
−3.3	.0005	.0005	.0005	.0004	.0004	.0004	.0004	.0004	.0004	.0003
−3.2	.0007	.0007	.0006	.0006	.0006	.0006	.0006	.0005	.0005	.0005
−3.1	.0010	.0009	.0009	.0009	.0008	.0008	.0008	.0008	.0007	.0007
−3.0	.0013	.0013	.0013	.0012	.0012	.0011	.0011	.0011	.0010	.0010
−2.9	.0019	.0018	.0018	.0017	.0016	.0016	.0015	.0015	.0014	.0014
−2.8	.0026	.0025	.0024	.0023	.0023	.0022	.0021	.0021	.0020	.0019
−2.7	.0035	.0034	.0033	.0032	.0031	.0030	.0029	.0028	.0027	.0026
−2.6	.0047	.0045	.0044	.0043	.0041	.0040	.0039	.0038	.0037	.0036
−2.5	.0062	.0060	.0059	.0057	.0055	.0054	.0052	.0051	.0049	.0048
−2.4	.0082	.0080	.0078	.0075	.0073	.0071	.0069	.0068	.0066	.0064
−2.3	.0107	.0104	.0102	.0099	.0096	.0094	.0091	.0089	.0087	.0084
−2.2	.0139	.0136	.0132	.0129	.0125	.0122	.0119	.0116	.0113	.0110
−2.1	.0179	.0174	.0170	.0166	.0162	.0158	.0154	.0150	.0146	.0143
−2.0	.0228	.0222	.0217	.0212	.0207	.0202	.0197	.0192	.0188	.0183
−1.9	.0287	.0281	.0274	.0268	.0262	.0256	.0250	.0244	.0239	.0233
−1.8	.0359	.0351	.0344	.0336	.0329	.0322	.0314	.0307	.0301	.0294
−1.7	.0446	.0436	.0427	.0418	.0409	.0401	.0392	.0384	.0375	.0367
−1.6	.0548	.0537	.0526	.0516	.0505	.0495	.0485	.0475	.0465	.0455
−1.5	.0668	.0655	.0643	.0630	.0618	.0606	.0594	.0582	.0571	.0559
−1.4	.0808	.0793	.0778	.0764	.0749	.0735	.0721	.0708	.0694	.0681
−1.3	.0968	.0951	.0934	.0918	.0901	.0885	.0869	.0853	.0838	.0823
−1.2	.1151	.1131	.1112	.1093	.1075	.1056	.1038	.1020	.1003	.0985
−1.1	.1357	.1335	.1314	.1292	.1271	.1251	.1230	.1210	.1190	.1170
−1.0	.1587	.1562	.1539	.1515	.1492	.1469	.1446	.1423	.1401	.1379
−0.9	.1841	.1814	.1788	.1762	.1736	.1711	.1685	.1660	.1635	.1611
−0.8	.2119	.2090	.2061	.2033	.2005	.1977	.1949	.1922	.1894	.1867