Setting up your machine

- Download a copy of R on your local computer from the Comprehensive R Archive Network (CRAN). You can choose between binaries for Linux, Mac and Windows.
- Install one of R's integrated development environment (IDE), RStudio, which makes R coding much easier and faster as it allows you to type multiple lines of code, handle plots, install and maintain packages and navigate your programming environment much more productively.
- Install all suggested packages or dependencies including GUI.

https://cran.r-project.org/doc/manuals/R-intro.pdf

Introduction to Data Science in R
Participant's Guide

February 17th, 2021

Team: Vincent Lott
Brian Ashford
Yvonne Phillips

BeVera

About BeVera

- Founded in 2013
- HQ in Metro Atlanta
- Veteran-Owned Small Business
- Data Science Training and Consulting
- Public Health and Technical Staffing
- CDC Customers: CSELS, CGH, NCEH, NCEZID

BK Ashford, Director of Data Science

- Over 22 years of Data Science solution delivery experience
- Analytical Consultant for SAS Institute for over 11 years
- Successful Data Science solution delivery across multiple industries
  - Insurance Carriers
  - Major Airlines
  - Credit Bureaus
  - Retail
- Education:
  - M.Sc. Applied Mathematics, Lehigh University
  - B.S. Mathematics, Morehouse College

BeVera

Yvonne Phillips, INSTRUCTOR

- 18 years of Predictive Analytics work
- Currently Adjunct Professor, Morehouse College – Computer Science/Data Science
- Experienced analytic professional with demonstrated history in:
  - Insurance: Underwriting Auto & Property, Motor Vehicle Reporting Analytics
  - Government: Federal and State/Local government customer solutions needs
  - Financial/Credit Cards
  - Retail
- Education:
  - M.Sc., Decision Science, Georgia State University, Robinson College of Business
  - B.S., Mathematics, Spelman College

BeVera

Agenda

1. Big Data
   - What is it? How to define it? Challenges and advantages?
2. Overview of the Data Science Lifecycle
   - Phases of a Data Science Project
3. R Programming Basics
   - Data Structures
   - Exploratory Data Analysis (Basic Plots and Graphs)
   - Intro to Dplyr and Tidyr
4. Quick Review of Statistics
5. Intro into Machine Learning Algorithms

Learning OBJECTIVES: during these sessions, Our Data Science trainers will:

- Describe the data science process and how its components interact
- Define big data and know the challenges
- Explain the significance of exploratory data analysis and data science
- Demonstrate basic data science techniques using the R programming language
  - Apply basic tools plots, graphs, some re-statistics to carry out EDA
  - Use are the carry out basic statistical modeling and analysis
- Relate data science techniques to common scenarios
- Introduce attendees to machine learning techniques commonly used

BeVera

Module 1: Big Data

BeVera

“Big data is a collection of data from traditional and digital sources inside and outside your company that represents a source for ongoing discovery and analysis.” (Lisa Arthur, CMO Network, 8/15/2013). “Big Data” is data whose scale, diversity, and complexity require new architecture, new tools, techniques, algorithms, and analytics to manage it and extract value and hidden knowledge from it…

The FOUR V's of Big Data

From traffic patterns and music downloads to web history and medical records, data is recorded, stored, and analyzed to enable the technology and services that the world relies on every day. But what exactly is big data, and how can these massive amounts of data be used?

As a leader in the sector, IBM data scientists break big data into four dimensions: **Volume, Velocity, Variety and Veracity**

Depending on the industry and organization, big data encompasses information from multiple internal and external sources such as transactions, social media, enterprise content, sensors and mobile devices. Companies can leverage data to adapt their products and services to better meet customer needs, optimize operations and infrastructure, and find new sources of revenue.

By 2015
4.4 MILLION IT JOBS
will be created globally to support big data, with 1.9 million in the United States

Volume
SCALE OF DATA

40 ZETTABYTES
[43 TRILLION GIGABYTES]
of data will be created by 2020, an increase of 300 times from 2005

It's estimated that
2.5 QUINTILLION BYTES
[2.3 TRILLION GIGABYTES]
of data are created each day

6 BILLION PEOPLE
have cell phones

Most companies in the U.S. have at least
100 TERABYTES
[100,000 GIGABYTES]
of data stored

Variety
DIFFERENT FORMS OF DATA

As of 2011, the global size of data in healthcare was estimated to be
150 EXABYTES
[161 BILLION GIGABYTES]

By 2014, it's anticipated there will be
420 MILLION WEARABLE, WIRELESS HEALTH MONITORS

4 BILLION+ HOURS OF VIDEO
are watched on YouTube each month

30 BILLION PIECES OF CONTENT
are shared on Facebook every month

400 MILLION TWEETS
are sent per day by about 200 million monthly active users

Velocity
ANALYSIS OF STREAMING DATA

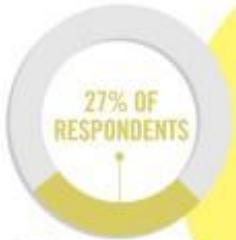
The New York Stock Exchange captures
1 TB OF TRADE INFORMATION
during each trading session

Modern cars have close to
100 SENSORS
that monitor items such as fuel level and tire pressure

By 2016, it is projected there will be
18.9 BILLION NETWORK CONNECTIONS
– almost 2.5 connections per person on earth

Veracity
UNCERTAINTY OF DATA

1 IN 3 BUSINESS LEADERS
don't trust the information they use to make decisions

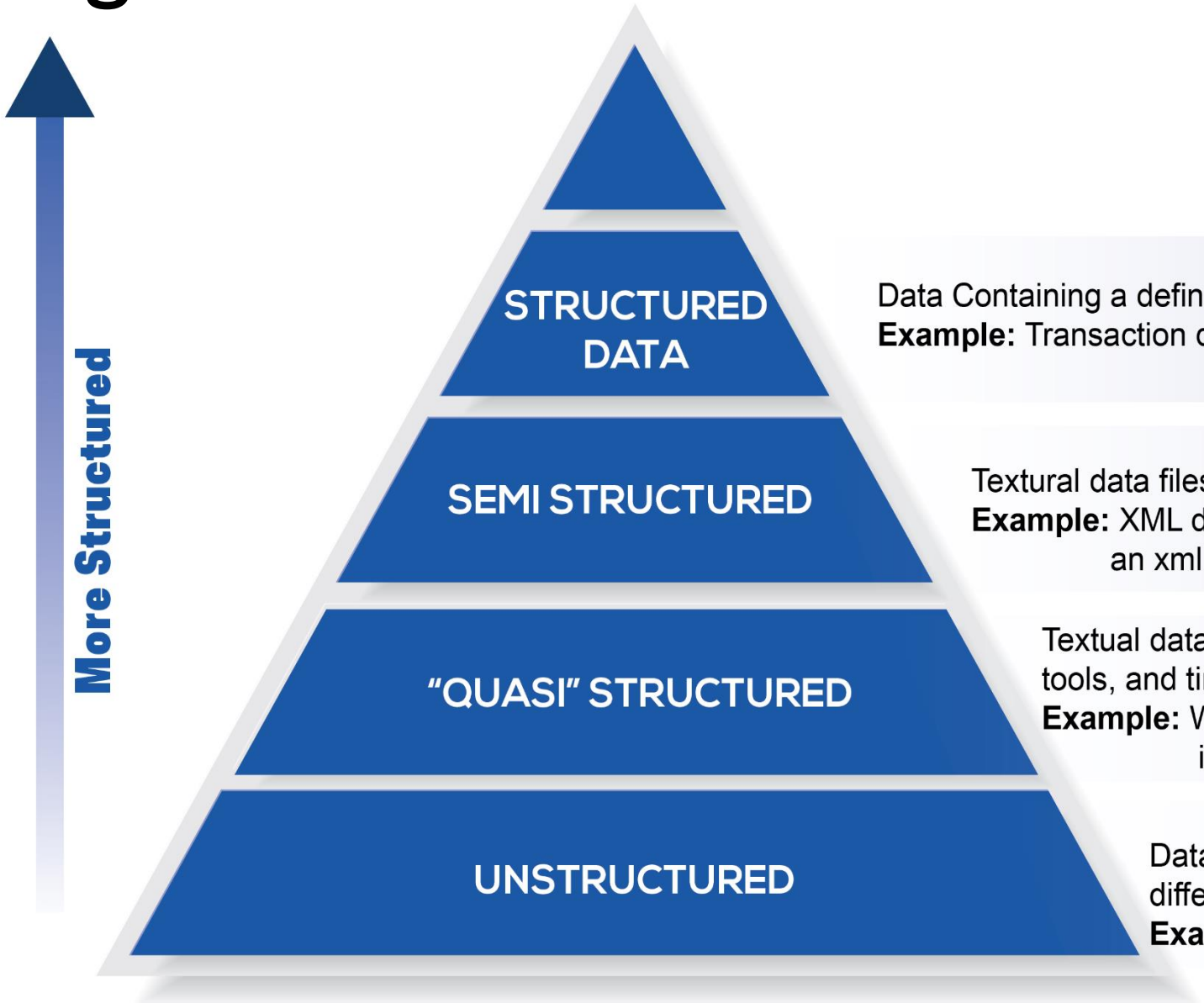


in one survey were unsure of how much of their data was inaccurate

Poor data quality costs the US economy around
$3.1 TRILLION A YEAR

Sources: McKinsey Global Institute, Twitter, Cisco, Gartner, EMC, SAS, IBM, MEPTEC, QAS

Big Data Characteristics: Data Structure



Data Containing a defined data type, format, structure
Example: Transaction data and OLAP

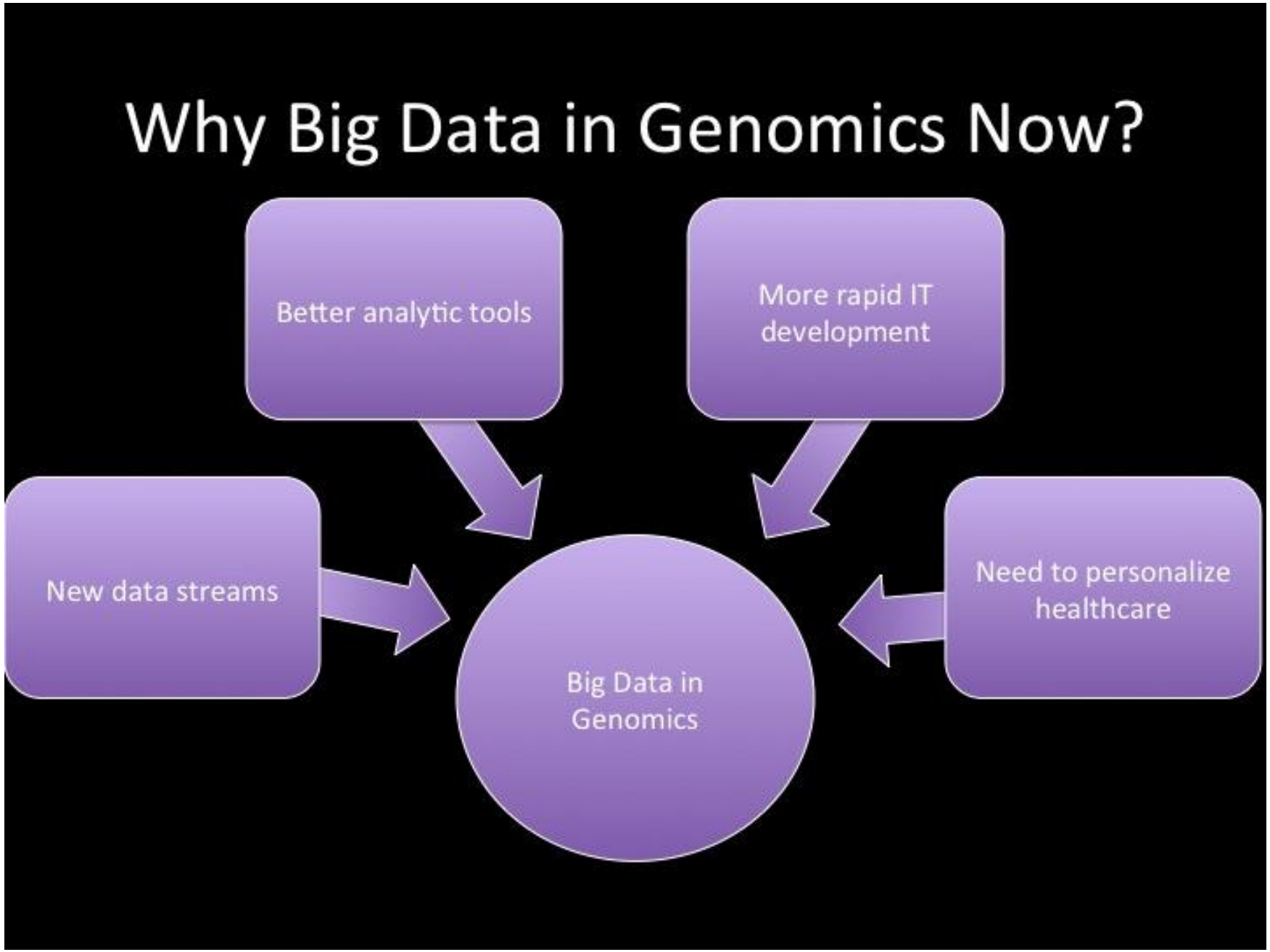
Textural data files with a discernable pattern, enabling parsing
Example: XML data files that are self describing and defined by an xml schema.

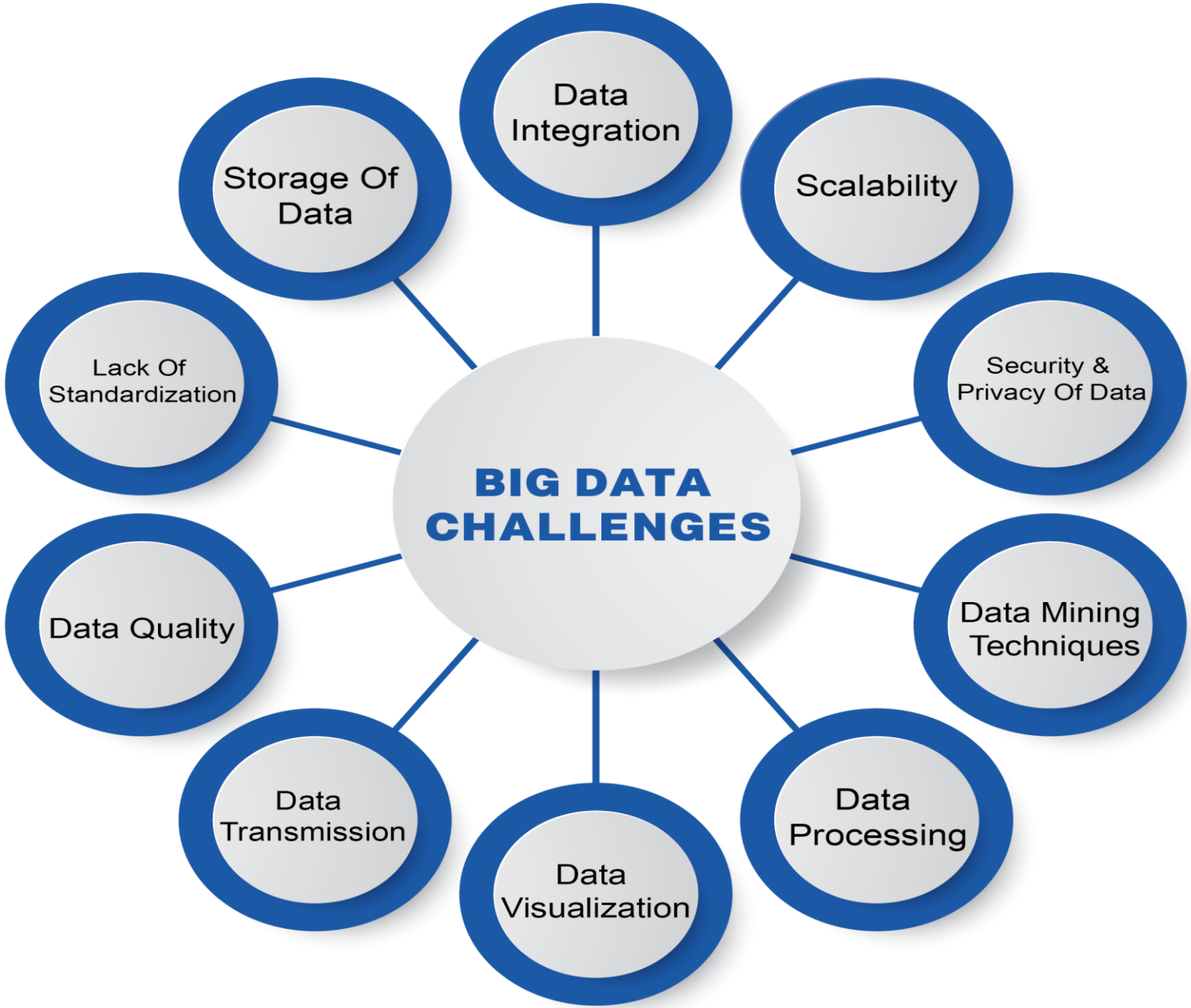
Textual data with erratic data formats, can be formatted with effort, tools, and time
Example: Web clickstream data that may contain some inconsistenscies in datavalues and formats..

Data that has no inherent structure and is usually stored as different types of files.
Example: Text document, PDFs, images and video.

Traditional Data vs. Big Data

- Challenges
- Advantages

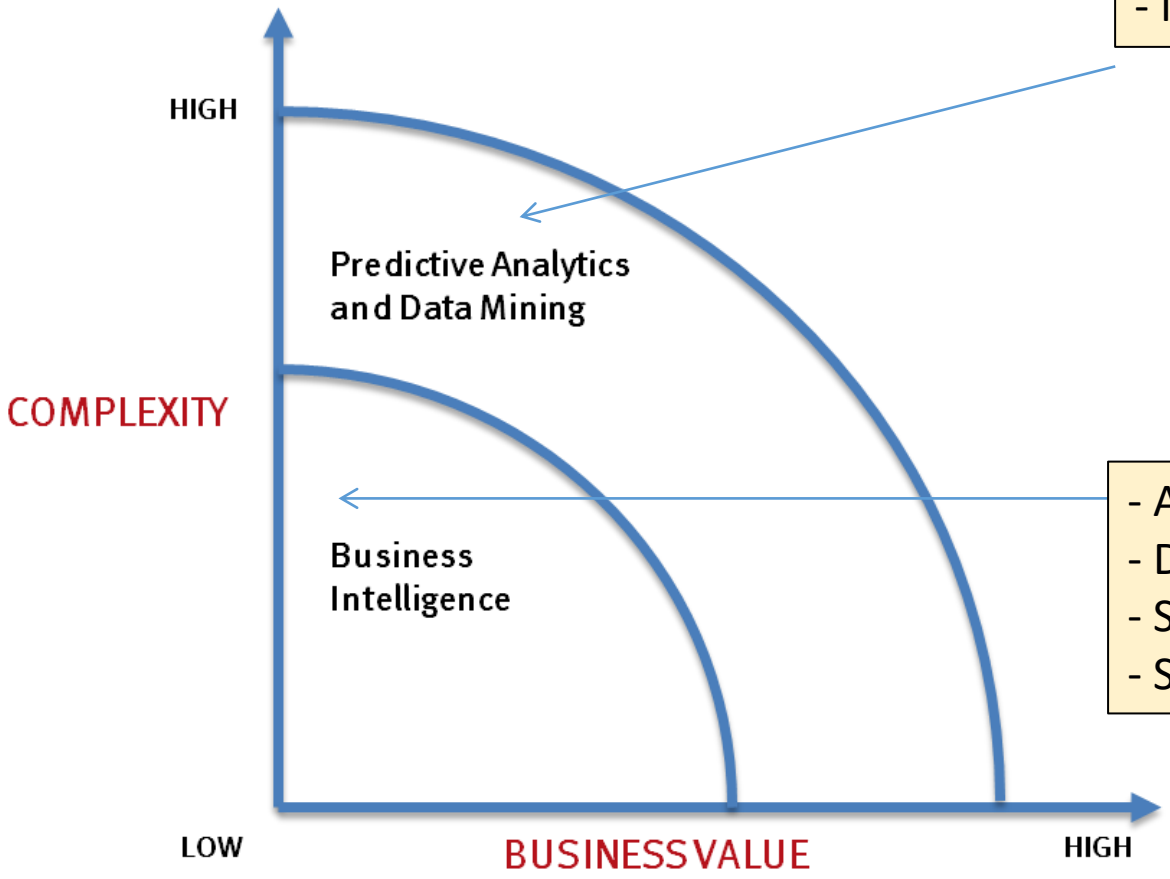

Why Big Data in Genomics Now?
Better analytic tools
More rapid IT development
New data streams
Need to personalize healthcare
Big Data in Genomics


BIG DATA CHALLENGES
Data Integration
Scalability
Security & Privacy Of Data
Data Mining Techniques
Data Processing
Data Visualization
Data Transmission
Data Quality
Lack Of Standardization
Storage Of Data

What's Driving Big Data

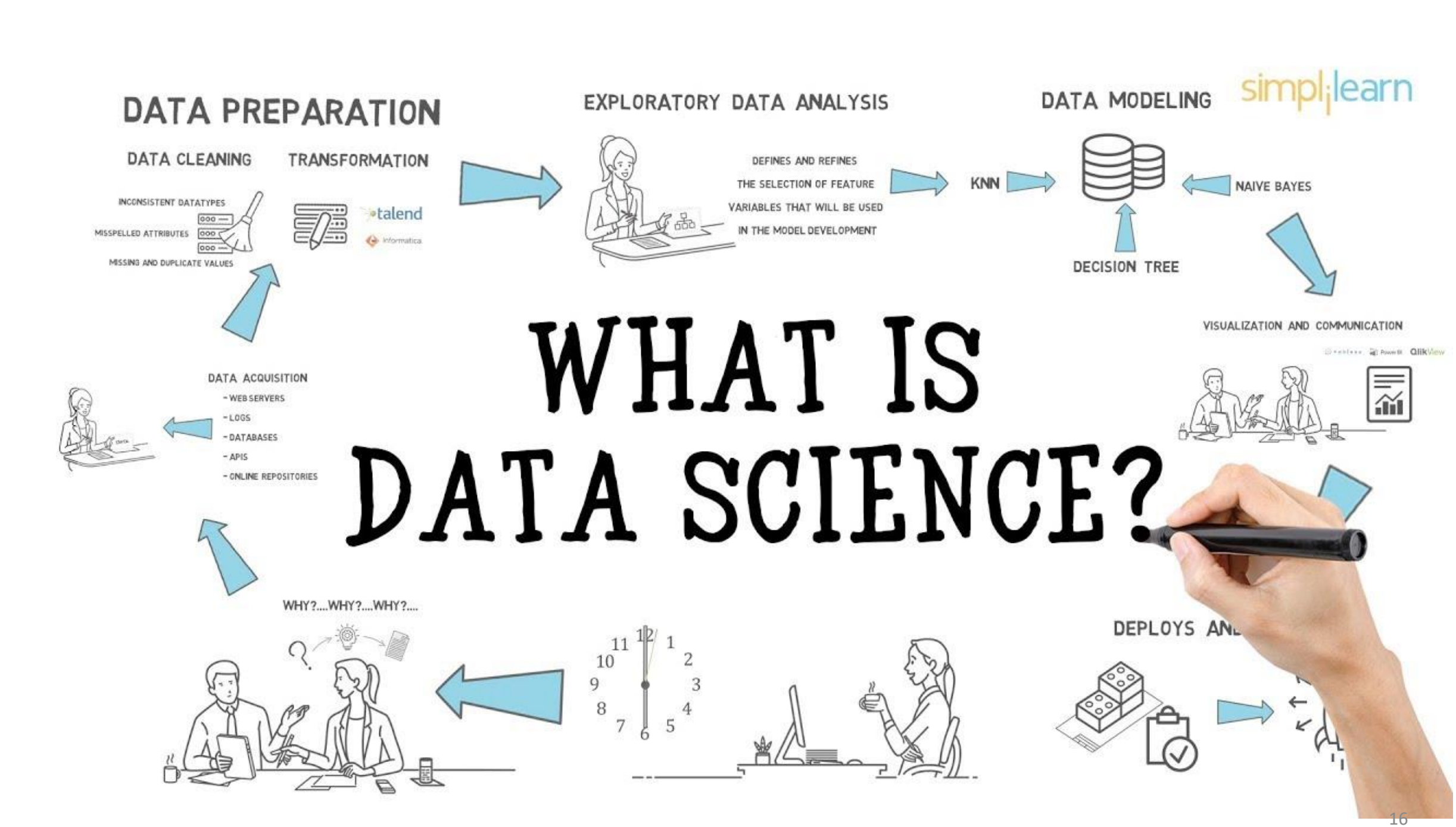
- Optimizations and predictive analytics
- Complex statistical analysis
- All types of data, and many sources
- Very large datasets
- More of a real-time

Common questions
- What if…
- What's the optimal scenario for our business?
- What will happen next? What if these trends continue? Why is this happening?



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

Common questions
- What happened last quarter?
- How many did we sell?
- Where is the problem? In which situation?


DATA PREPARATION
DATA CLEANING
TRANSFORMATION
INCONSISTENT DATATYPES
MISSPELLED ATTRIBUTES
MISSING AND DUPLICATE VALUES
talend
Informatica
EXPLORATORY DATA ANALYSIS
DEFINES AND REFINES
THE SELECTION OF FEATURE
VARIABLES THAT WILL BE USED
IN THE MODEL DEVELOPMENT
DATA MODELING
simplilearn
KNN
NAIVE BAYES
DECISION TREE
VISUALIZATION AND COMMUNICATION
DEPLOYS AN
DATA ACQUISITION
- WEB SERVERS
- LOGS
- DATABASES
- APIS
- ONLINE REPOSITORIES
WHY?....WHY?....WHY?....
WHAT IS
DATA SCIENCE?

Module 2: R Programming Basics

https://cran.r-project.org/doc/manuals/R-intro.pdf

BeVera

Setting up your machine

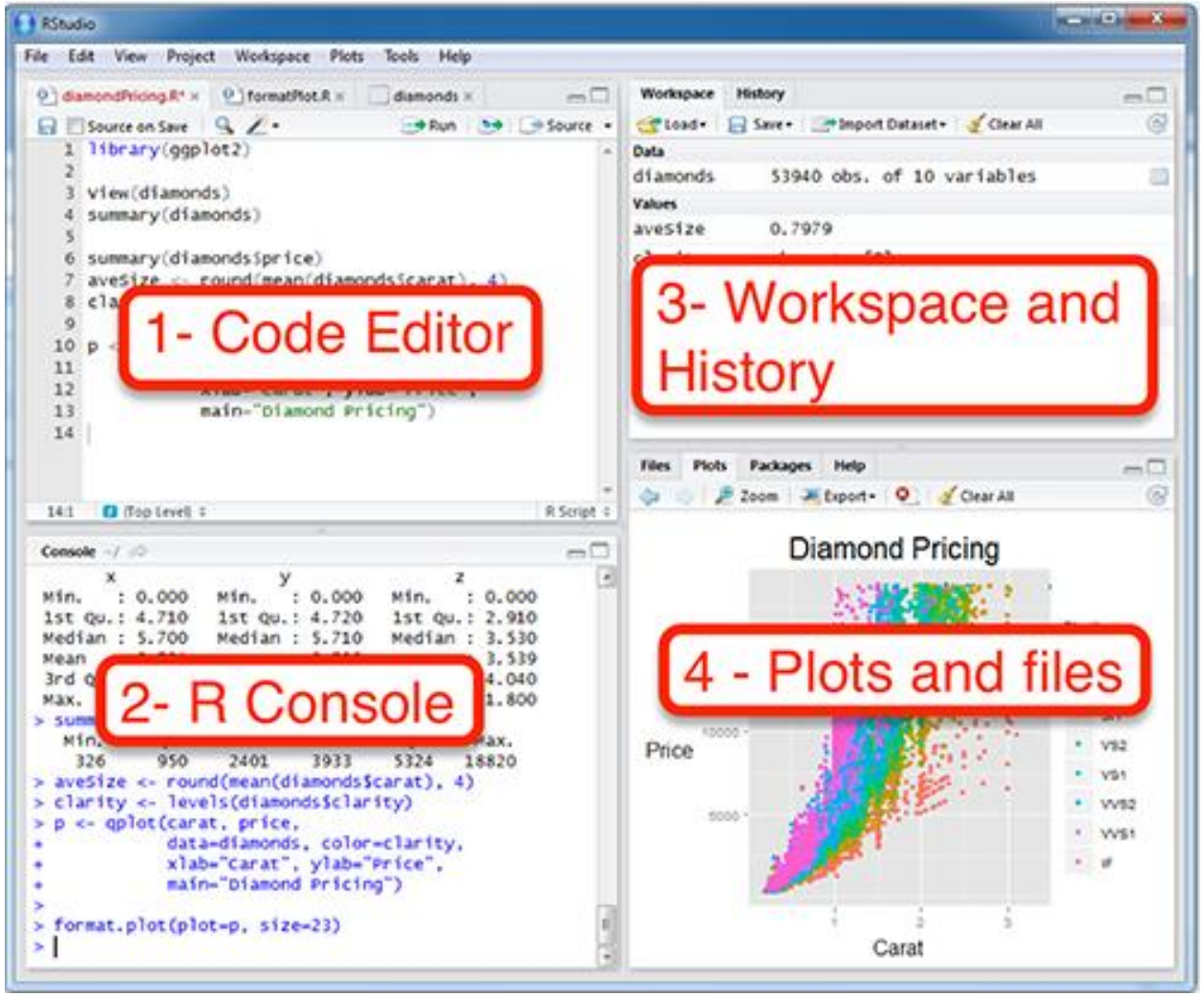
- Download a copy of R on your local computer from the Comprehensive R Archive Network (CRAN). You can choose between binaries for Linux, Mac and Windows.
- Install one of R's integrated development environment (IDE), RStudio, which makes R coding much easier and faster as it allows you to type multiple lines of code, handle plots, install and maintain packages and navigate your programming environment much more productively.
- Install all suggested packages or dependencies including GUI.

https://cran.r-project.org/doc/manuals/R-intro.pdf

RStudio is a four pane work-space for 1) creating file containing R script, 2) typing R commands, 3) viewing command histories, 4) viewing plots and more.

1.Top-left panel:
◦Code editor allowing you to create and open a file containing R script.
◦The R script is where you keep a record of your work. R script can be created as follow: File –> New –> R Script

2.Bottom-left panel:
◦R console for typing R commands



3.Top-right panel:
◦Workspace tab: shows the list of R objects you created during your R session
◦History tab: shows the history of all previous commands

4.Bottom-right panel:
◦Files tab: show files in your working directory
◦Plots tab: show the history of plots you created. From this tab, you can export a plot to a PDF or an image files
◦Packages tab: show external R packages available on your system. If checked, the package is loaded in R.

Labeled key	Ctrl-key combination	Effect
Up arrow	Ctrl-P	Recall previous command by moving backward through the history of commands
Down arrow	Ctrl-N	Move forward through the history of commands
Backspace	Ctrl-H	Delete the character to the left of cursor
Delete	Ctrl-D	Delete the character to the right of cursor
Home	Ctrl-A	Move cursor to the start of the line
End	Ctrl-E	Move cursor to the end the line
Rsish arrow	Ctrl-F	Move cursor right (forward) one character
Left arrow	Ctrl-B	Move cursor left (back) one character
	Ctrl-K	Delete everything from the cursor position to the end of the line.
	Ctrl-U	Clear the whole line and start over
Tab		Name completion (used on some platforms)

Getting help on a function

> help(*functionname*)

> args(*functionname*)

> example(*functionname*)

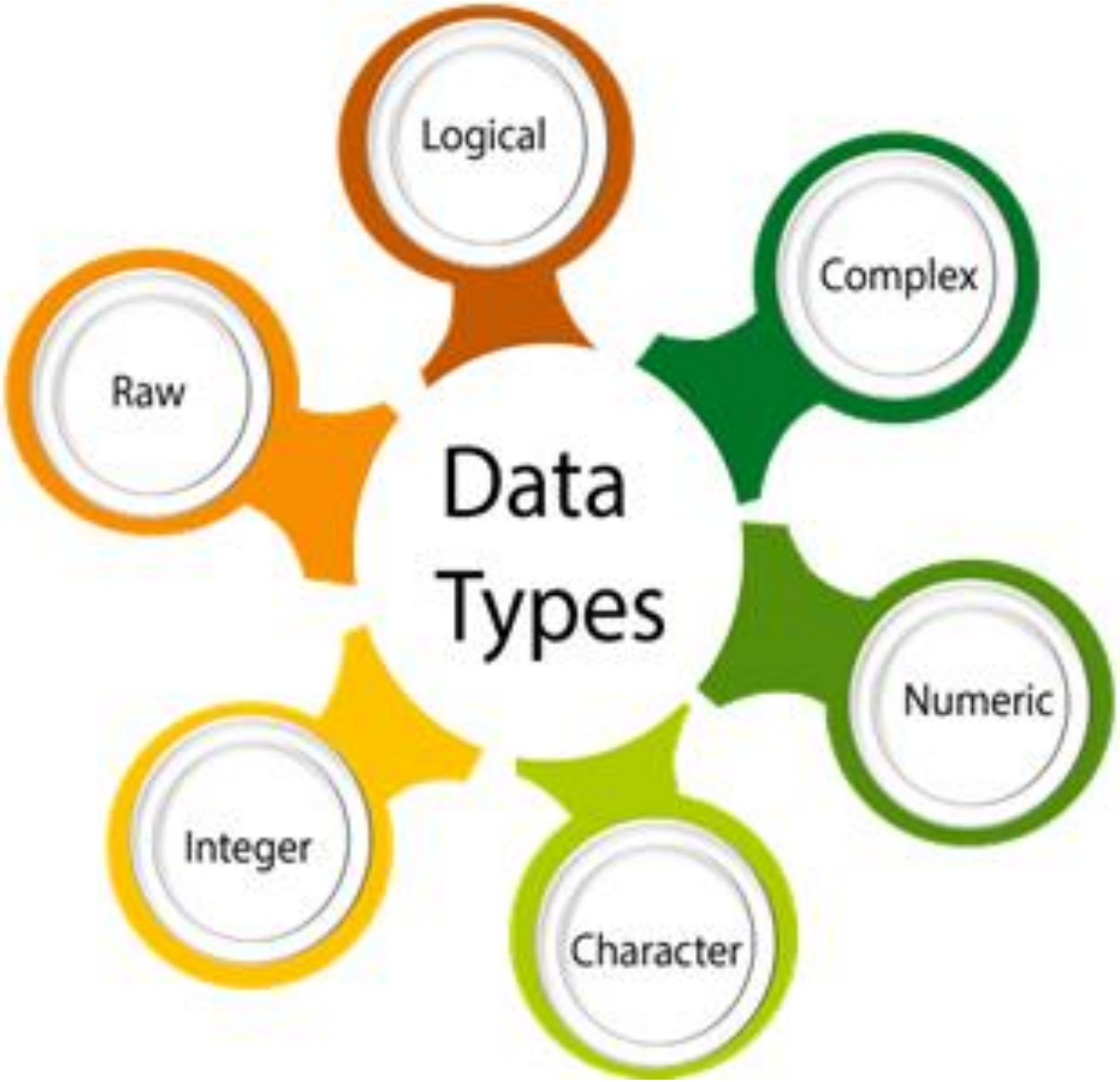
The Working Directory

- The working directory is the default location for all input and output
  - Reading and writing data files
  - Opening and saving script files
  - Saving workspace image

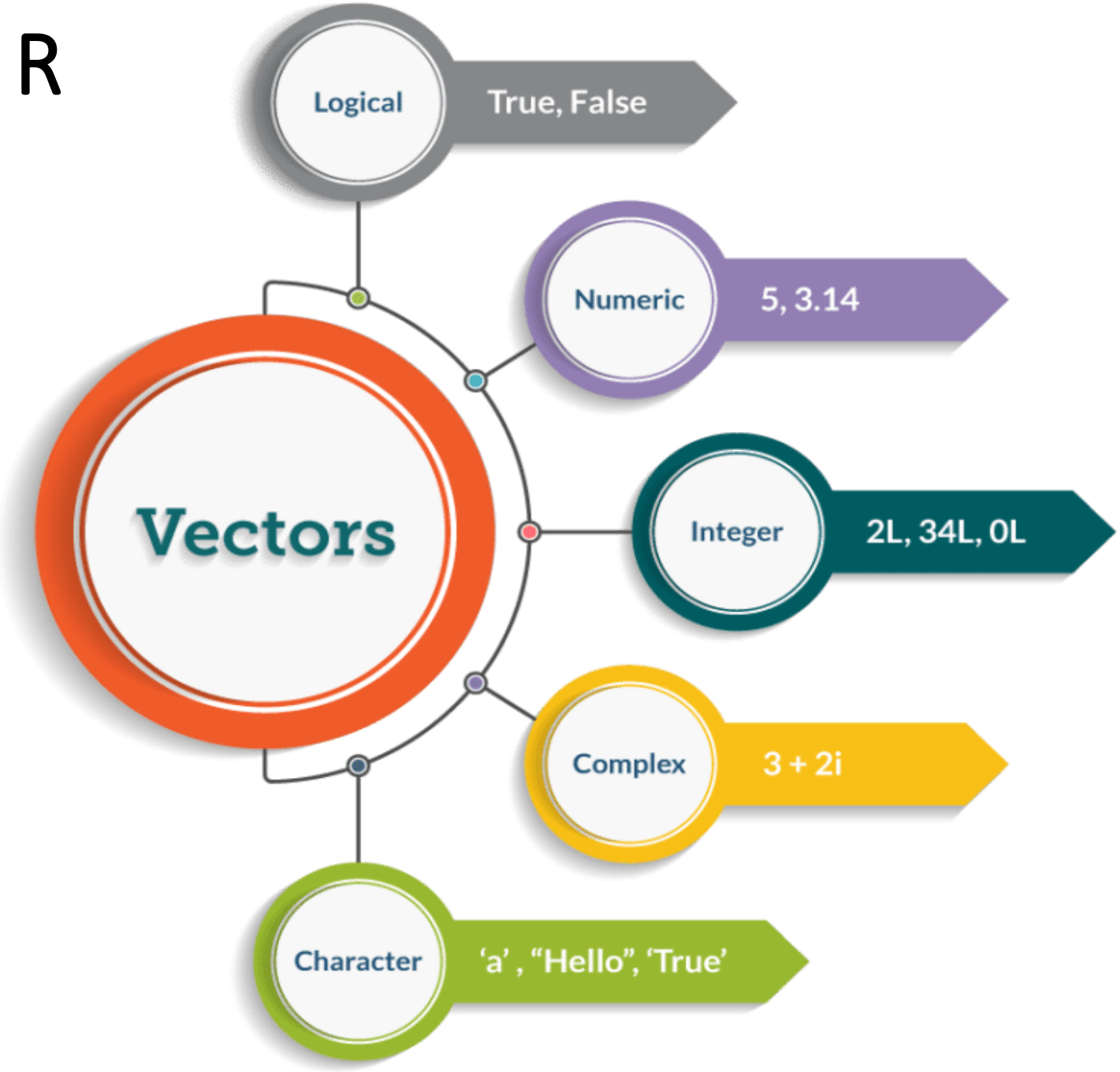
- From the command line,
  - getwd() – reports the working directory
  - setwd() – changes the working directory

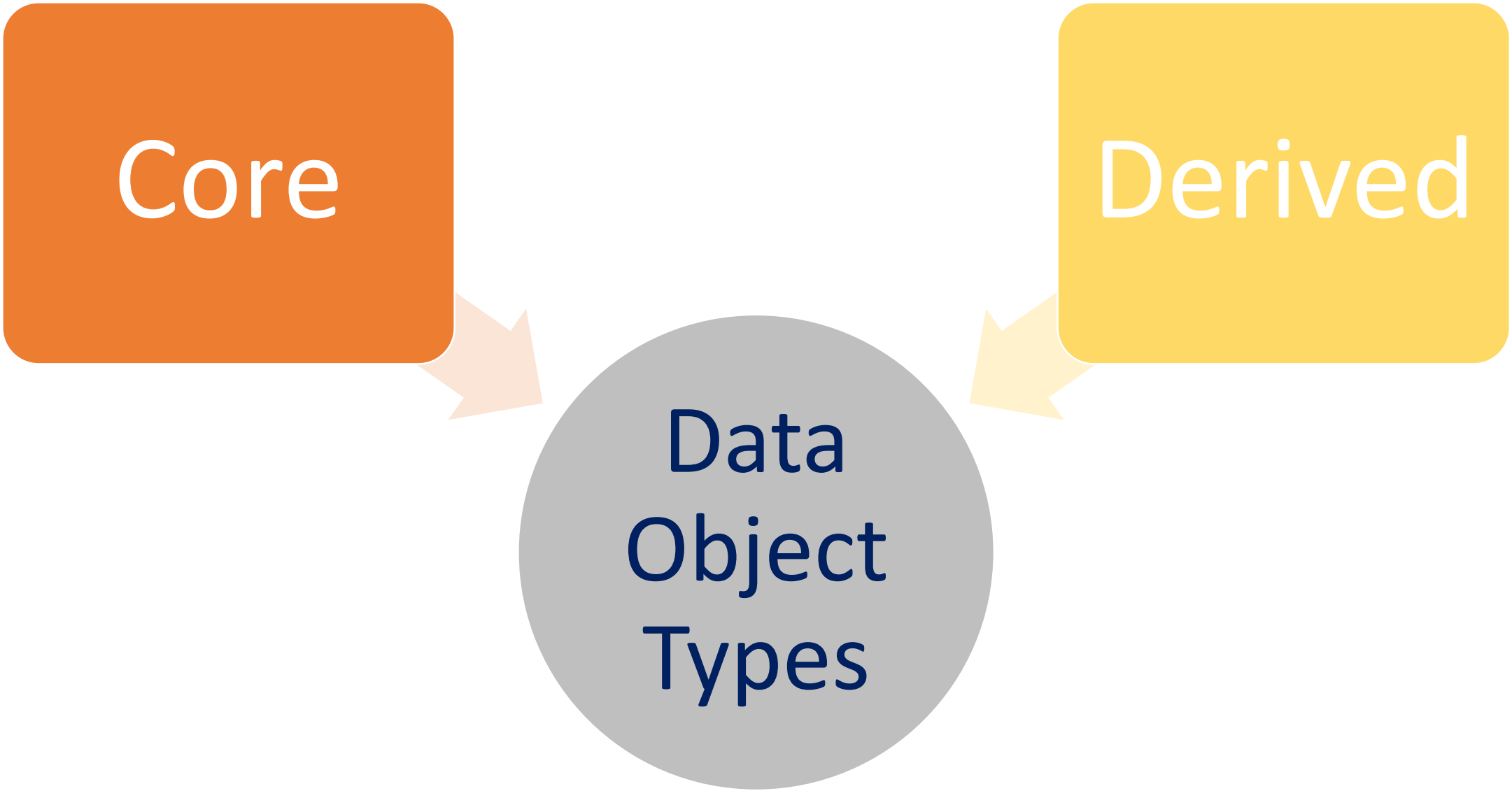
Search Path – list of packages currently loaded into the memory

```
> search( )

 [1] ".GlobalEnv"         "tools:rstudio"         "package:stats"
 [4] "package:graphics"   "package:grDevices"   "package:utils"
 [7] "package:datasets"   "package:methods"     "Autoloads"
[10] "package:base"
```


Logical
Complex
Raw
Data Types
Numeric
Integer
Character

Data Types in R




Core
Derived
Data Object Types

Types

Converting between common data types in R. Can always go from a higher value in the table to a lower value.

`as.logical`	`TRUE, FALSE, TRUE`	Boolean values (TRUE or FALSE).
`as.numeric`	`1, 0, 1`	Integers or floating point numbers.
`as.character`	`'1', '0', '1'`	Character strings. Generally preferred to factors.
`as.factor`	`'1', '0', '1', levels: '1', '0'`	Character strings with preset levels. Needed for some statistical models.

Maths Functions

`log(x)`	Natural log.	`sum(x)`	Sum.
`exp(x)`	Exponential.	`mean(x)`	Mean.
`max(x)`	Largest element.	`median(x)`	Median.
`min(x)`	Smallest element.	`quantile(x)`	Percentage quantiles.
`round(x, n)`	Round to n decimal places.	`rank(x)`	Rank of elements.
`signif(x, n)`	Round to n significant figures.	`var(x)`	The variance.
`cor(x, y)`	Correlation.	`sd(x)`	The standard deviation.

Variable Assignment

```
> a <- 'apple'
> a
[1] 'apple'
```

The Environment

`ls()`	List all variables in the environment.
`rm(x)`	Remove x from the environment.
`rm(list = ls())`	Remove all variables from the environment.

You can use the environment panel in RStudio to browse variables in your environment.

Matrices

`m <- matrix(x, nrow = 3, ncol = 3)`
Create a matrix from x.


`m[2, ]` - Select a row

`m[ , 1]` - Select a column

`m[2, 3]` - Select an element

`t(m)`
Transpose

`m %*% n`
Matrix Multiplication

`solve(m, n)`
Find x in: m * x = n

Lists

`l <- list(x = 1:5, y = c('a', 'b'))`
A list is a collection of elements which can be of different types.

`l[[2]]`	`l[1]`	`l$x`	`l['y']`
Second element of l.	New list with only the first element.	Element named x.	New list with only element named y.

Also see the **dplyr** package.

Data Frames

`df <- data.frame(x = 1:3, y = c('a', 'b', 'c'))`
A special case of a list where all elements are the same length.

x	y
1	a
2	b
3	c

List subsetting


`df$x`


`df[[2]]`

Understanding a data frame

`View(df)` See the full data frame.

`head(df)` See the first 6 rows.

Matrix subsetting


`df[ , 2]`


`df[2, ]`

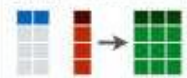

`df[2, 2]`

`nrow(df)`
Number of rows.

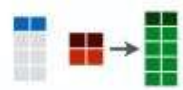
`ncol(df)`
Number of columns.

`dim(df)`
Number of columns and rows.

`cbind` - Bind columns.



`rbind` - Bind rows.



Strings

Also see the **stringr** package.

`paste(x, y, sep = ' ')`	Join multiple vectors together.
`paste(x, collapse = ' ')`	Join elements of a vector together.
`grep(pattern, x)`	Find regular expression matches in x.
`gsub(pattern, replace, x)`	Replace matches in x with a string.
`toupper(x)`	Convert to uppercase.
`tolower(x)`	Convert to lowercase.
`nchar(x)`	Number of characters in a string.

Factors

`factor(x)`
Turn a vector into a factor. Can set the levels of the factor and the order.

`cut(x, breaks = 4)`
Turn a numeric vector into a factor by 'cutting' into sections.

Statistics

`lm(y ~ x, data=df)`
Linear model.

`glm(y ~ x, data=df)`
Generalised linear model.

`summary`
Get more detailed information out a model.

`t.test(x, y)`
Perform a t-test for difference between means.

`pairwise.t.test`
Perform a t-test for paired data.

`prop.test`
Test for a difference between proportions.

`aov`
Analysis of variance.

Distributions

	Random Variates	Density Function	Cumulative Distribution	Quantile
Normal	`rnorm`	`dnorm`	`pnorm`	`qnorm`
Poisson	`rpois`	`dpois`	`ppois`	`qpois`
Binomial	`rbinom`	`dbinom`	`pbinom`	`qbinom`
Uniform	`runif`	`dunif`	`punif`	`qunif`

Plotting

Also see the **ggplot2** package.


`plot(x)`
Values of x in order.


`plot(x, y)`
Values of x against y.


`hist(x)`
Histogram of x.

Dates

See the **lubridate** package.

Declare variables of different types

Example 1:
# Numeric
x <- 28
class(x)
Output: ## [1] "numeric"

Example 2:
# String
y <- "R is Fantastic"
class(y)
Output: ## [1] "character“

Example 3:
# Boolean
z <- TRUE
class(z)
Output: ## [1] "logical"

Variables

- Variables store values and are an important component in programming. A variable can store a number, an object, a statistical result, vector, dataset, a model prediction basically anything R outputs. We can use that variable later simply by calling the name of the variable.

- To declare a variable, we need to assign a variable name. The name should not have space. We can use _ to connect to words.

- To add a value to the variable, use <- or =.

Here is the syntax:

```
# First way to declare a variable:  use the `<-`
name_of_variable <- value

# Second way to declare a variable:  use the `=`
name_of_variable = value
```

In the command line, we can write the following codes to see what happens:

Example 1:

```
# Print variable x
x <- 42
x
```

Output: ## [1] 42

Example 2:

```
y  <- 10
y
```

Output: ## [1] 10

Example 3:

```
# We call x and y and apply a subtraction
x-y
```

Output: ## [1] 32

:

Basic Arithmetic Operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
^ or **	Exponentiation

```
# Examples:
# A multiplication
3*9
Output: ## [1] 27

# A division
(5+13)/2
Output: ## [1] 9

# Exponentiation
-3^2
Output: ## [1] 9
```

Logical Operators return values inside the vector based on logical conditions.

Operator	Description
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Exactly equal to
!=	Not equal to
!x	Not x
x	y
x & y	x AND y
isTRUE(x)	Test if X is TRUE

You can add many conditional statements, but we need to include them in a parenthesis. Follow this structure to create a conditional statement:

```
variable_name[(conditional_statement)]
```

```
# Example:


# Create a vector from 1 to 8
logical_vector <- c(1:8)
logical_vector > 6


Output: ## [1]FALSE FALSE FALSE FALSE FALSE FALSE TRUE TRUE
```

```
# Example:


logical_vector <- c(1:8)
logical_vector[(logical_vector>3) & (logical_vector<5)]


Output: ## [1] 4
```

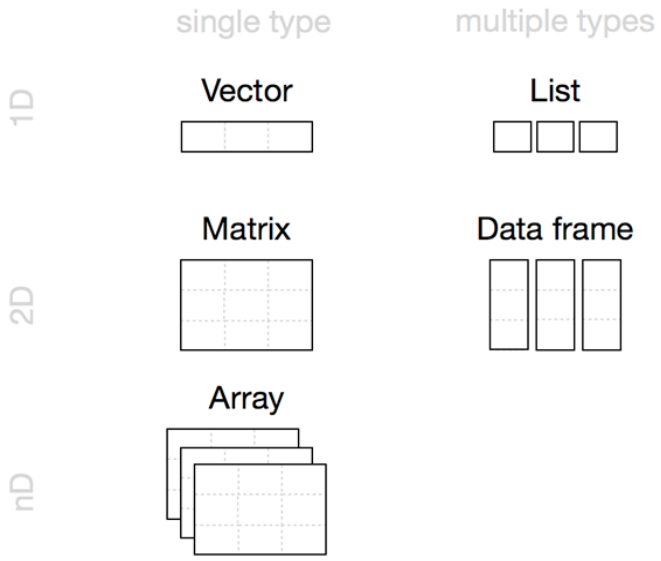
Data Structures in

1. VECTORS
2. MATRIX
3. ARRAY
4. LIST
5. DATA FRAME

Learning objectives of this module:

- What are the three properties of a vector, other than its contents?
- What are the four common types of atomic vectors? What are the two rare types?
- What are attributes? How do you get them and set them?
- How is a list different from an atomic vector? How is a matrix different from a data frame?
- Can you have a list that is a matrix?

Fundamental Data Structures



Vector

- A sequence of numbers or characters, or higher-dimensional arrays like matrices

Matrix

- The basic two dimensional data structure in R is the vector

Array

- If a higher dimension vector is desired, then use the function to generate the n-dimensional object.

List

- An ordered set of components stored in a 1D vector

Data.Frame

- A table-like structure (experimental results often collected in this form)

Factor

- A sequence assigning a category to each index.

Atomic Data Elements: Vectors

- In R the “base” type is a vector, not a scalar.
- A vector is an indexed set of values that are all of the same type. The type of the entries determines the class of the vector.
- The possible vectors data types are:
  - integer
  - numeric
  - character
  - complex
  - logical
- An integer is a subclass of numeric.
- Cannot combine vectors of different modes

Creating Vectors

Assignment of a value to a variable is done with <- (two symbols, no space).

> v <- 1

➢ v

Vectors can only contain entries of the same type: numeric or character; you can't mix them.

- Note that characters should be surrounded by " ". The most basic way to create a vector is with **c**(x1, . . . , xn), and it works for characters and numbers alike.

> x <- c("a", "b", "c")

> length(x)

[1] 3

Can also generate regular sequences:

seq(from = #, to = #, by = #): allows you to create a sequence from a starting number to an ending number.

rep(x = c(1, 2), each = 3): function allows you to repeat a scalar (or vector) a specified number of times, or to a desired length.

Selecting Vector Elements

You can select the indexing technique appropriate:

- Use square brackets [ ] to select vector elements by their position
  v[2] – second element of v
- Use negative indexes to exclude elements
- Use a vector of indexes to select multiple values.
- Use a logical vector to select elements based on a condition
- Use names to access named elements.

Vector Arithmetic

Numeric vectors can be used in arithmetic expressions, in which case the operations are performed element by element to produce another vector.

```
> x <- rnorm(3)
> y <- rnorm(3)
> x
> x + 1
> c(1, 2, 3, 4, 5) + c(5, 4, 3, 2, 1)
```

More Vector Arithmetic Statistical operations on numeric vectors

- In studying data, you will make frequent use of sum, which gives the sum of the entries, max, min, mean.

Function	Example	Result
sum(x), product(x)	sum(1:20)	210
min(x), max(x)	min(1:20)	1
mean(x), median(x)	mean(1:20)	10.5
sd(x), var(x), range(x)	sd(1:20)	5.91608
quantile(x, probs)	quantile(1:20, probs = .2)	20%, 4.8
summary(x)	summary(1:20)	Min = 1.00. 1st Qu. = 5.75, Median = 10.50, Mean = 10.50, 3rd Qu. = 15.25, Max = 20.0

> (i.e., Sum((x - mean(x))^2)/(length(x) - 1))

- A useful function for quickly getting properties of a vector: summary(y)

More Vector Arithmetic Statistical operations on continuous vectors

Function	Description	Example	Result
round(x, digits)	Round elements in x to digits digits	round(c(3.712, 3.1415), digits = 1)	3.7, 3.1
ceiling(x), floor(x)	Round elements x to the next highest (or lowest) integer	ceiling(c(2.6, 8.1))	3, 9
x %% y	Modular arithmetic (ie. x mod y)	8 %% 4	0

Vector Arithmetic Statistical operations on discrete vectors

Function	Description	Example	Result
unique(x)	Returns a vector of all unique values.	unique(c(3, 3, 4,5,12))	3, 4, 5, 12
table(x, exclude)	Returns a table showing all the unique values as well as a count of each occurrence. To include a count of NA values, include the argument exclude = NULL	table(c("x", "x", "y", "z"))	x y z 2 1 1

See the “indexing vectors.R” in the folder

baby.names	baby.city	baby.ages	baby.weight	baby.eyecolor
amy	macon	13	21	brown
brittany	athens	21	22	brown
carol	pink	32	41	green
donna	savannah	6	16	blue
erin	savannah	12	18	blue
fran	atlanta	11	19.4	grey
gigi	atlanta	18	26	brown
helen	athens	16	23	green
irene	macon	17	22	brown
jackie	macon	34	36	brown

Factors in R: Categorical & Continuous Variables

Factors are variables in R which take on a limited number of different values; such variables are often referred to as categorical variables.

In a dataset, we can distinguish two types of variables:

categorical and **continuous**

- In a categorical variable, the value is limited and usually based on a particular finite group. For example, a categorical variable can be countries, year, gender, occupation.

- A continuous variable, however, can take any values, from integer to decimal. For example, we can have the revenue, price of a share, etc..

Categorical Variables

R stores categorical variables into a factor. Let's check the code below to convert a character variable into a factor variable. Characters are not supported in a machine learning algorithm, and the only way is to convert a string to an integer.

Syntax

```
factor(x = character(), levels, labels = levels, ordered = is.ordered(x))
```

Arguments:

- **x**: A vector of data. Need to be a string or integer, not decimal.
- **Levels**: A vector of possible values taken by x. This argument is optional. The default value is the unique list of items of the vector x.
- **Labels**: Add a label to the x data. For example, 1 can take the label `male` while 0, the label `female`.
- **ordered**: Determine if the levels should be ordered.

Nominal Categorical Variable

A categorical variable has several values but the order does not matter. For instance, male or female categorical variable do not have ordering.

```
# Create a color vector
color_vector <- c(‘turquoise', 'red', 'green', ivory', 'black', 'yellow')

# Convert the vector to factor
factor_color <- factor(color_vector)
factor_color
```

Output:

```
## [1] turquoise red green ivory black yellow
## Levels: black turquoise green red ivory yellow
```

Ordinal Categorical Variable

Ordinal categorical variables do have a natural ordering. We can specify the order, from the lowest to the highest with order = TRUE and highest to lowest with order = FALSE.

Example: We can use summary to count the values for each factor.

```
# Create Ordinal categorical vector
day_vector <- c('evening', 'morning', 'afternoon', 'midday', 'midnight', 'evening')
# Convert `day_vector` to a factor with ordered level
factor_day <- factor(day_vector, order = TRUE, levels =c('morning', 'midday', 'afternoon', 'evening', 'midnight'))
# Print the new variable
factor_day
```

Output:

```
## [1] evening morning afternoon midday midnight evening
```

Example:

```
## Levels: morning < midday < afternoon < evening < midnight
# Append the line to above code
# Count the number of occurrence of each level summary(factor_day)
```

Output:

```
## morning midday afternoon evening midnight
## 1 1 1 2 1
```

Continuous Variables

- Continuous class variables are the default value in R.
- They are stored as numeric or integer.
- We can see it from the dataset below. mtcars is a built-in dataset. It gathers information on different types of car. We can import it by using mtcars and check the class of the variable mpg, mile per gallon. It returns a numeric value, indicating a continuous variable.

```
dataset <- mtcars
class(dataset$mpg)
## [1]   “numeric”
```

Lists in R:

- A list is a generic object consisting of an ordered collection of objects.
- Lists are heterogeneous data structures.
- A list is an one-dimensional data structures.
- A list can be a list of vectors, list of matrices, a list of characters and a list of functions and so on.
- Lists are different from atomic vectors because their elements can be of any type, including lists.

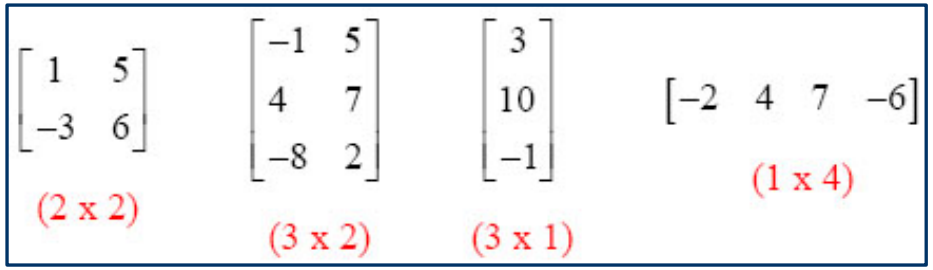
- You construct lists by using list() instead of c():

```
x <- list(1:5, “d", c(FALSE, FALSE, TRUE), c(3.5 7.2))
str(x)
```

```
# Example - Illustrate a List

# The first attributes is a numeric vector
# containing the employee IDs which is
# created using the 'c' command here
empId = c(1, 2, 3, 4)

# The second attribute is the employee's name
# which is created using this line of code here
# which is the character vector
empName = c(“Ann", “Sanjan", “Ellison", “Evette")

# The third attribute is the number of employees
# which is a single numeric variable.
numberOfEmp = 4

# We can combine all these three different
# data types into a list
# containing the details of employees
# which can be done using a list command
empList = list(empId, empName, numberOfEmp)

print(empList)
```

Matrix – What is a Matrix?

- A matrix is a 2-dimensional array that has m number of rows and n number of columns. In other words, matrix is a combination of two or more vectors with the same data type.



- You can create a matrix with the function matrix(). This function takes three arguments:
  matrix(data, nrow, ncol, byrow = FALSE)

Arguments:

- data: The collection of elements that R will arrange into the rows and columns of the matrix.
- nrow: Number of rows
- ncol: Number of columns
- byrow: The rows are filled from the left to the right. We use `byrow = FALSE` (default values), if we want the matrix to be filled by the columns i.e. the values are filled top to bottom.

#Print dimension of the matrix with dim()

Defining names of columns and rows in a matrix

```
# Print dimension of the matrix with dim()
  dim(matrix_a)
```

In order to define rows and column names, you can create two vectors of different names, one for row and other for a column. Then, using the Dimnames attribute, you can name them appropriately:

```
rows = c("row1", "row2", "row3", "row4")     #Creating our character vector of row names

cols = c("coln1", "coln2", "coln3")          #Creating our character vector of column names

mat <- matrix(c(4:15), nrow = 4, byrow = TRUE, dimnames = list(rows, cols) )
#creating our matrix mat and assigning our vectors to dimnames
```

```
# Print matrix
  print(mat)
```

Add a Column to a Matrix with the cbind()

- You can add a column to a matrix with the cbind() command.
- cbind() means column binding. cbind() can concatenate as many matrix or columns as specified.

Example:

```
# concatenate c(1:5) to the matrix_a
matrix_a1 <- cbind(matrix_a, c(1:5))
# Check the dimension dim(matrix_a1)
```

Output:

```
## [1] 5 3
```

Example:

```
matrix_a1
```

Output

```
##      [,1] [,2] [,3]
## [1,]    1    2    1
## [2,]    3    4    2
## [3,]    5    6    3
## [4,]    7    8    4
## [5,]    9   10    5
```

Example:
We can also add more than one column. Add the next sequence of number to the matrix_a2 matrix. The dimension of the new matrix will be 4x6 with number from 1 to 24.

```
matrix_a2 <-matrix(13:24, byrow = FALSE, ncol = 3)

##      [,1] [,2] [,3]
## [1,]   13   17   21
## [2,]   14   18   22
## [3,]   15   19   23
## [4,]   16   20   24
```

Example:

```
matrix_c <-matrix(1:12, byrow = FALSE, ncol = 3)
matrix_d <- cbind(matrix_a2, matrix_c)
dim(matrix_d)
```

Output:

```
## [1] 4 6
```

Slice a Matrix: Accessing individual components

We can select elements one or many elements from a matrix by using the square brackets [ ]. This is where slicing comes into the picture.

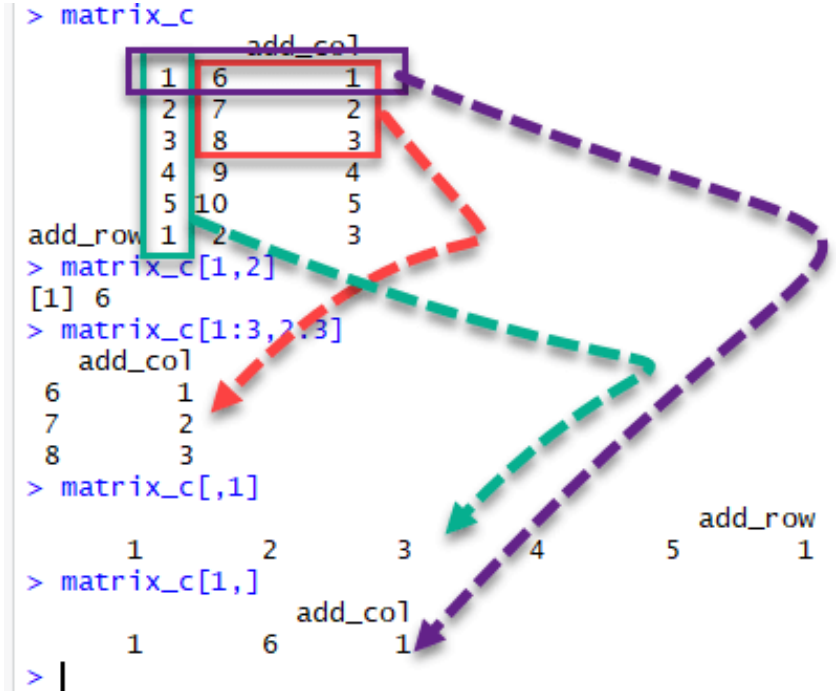
For example:

matrix_c[1,2] selects the element at the first row and second column.

matrix_c[1:3,2:3] results in a matrix with the data on the rows 1, 2, 3 and columns 2, 3,

matrix_c[,1] selects all elements of the first column.

matrix_c[1,] selects all elements of the first row.



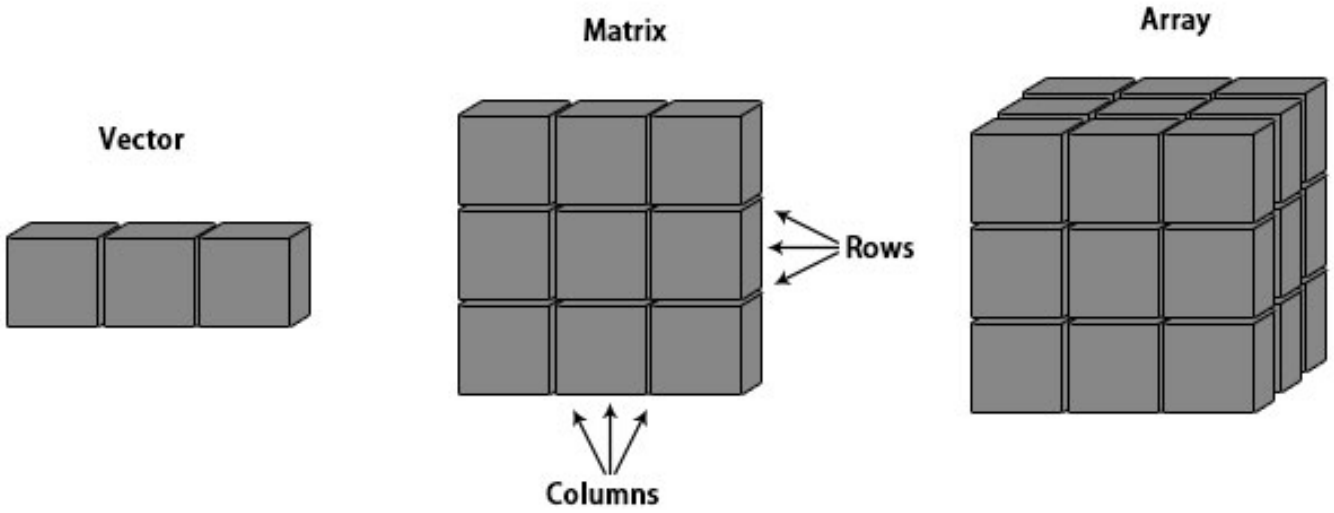
```
# Examples:
> A = matrix( +   c(2, 4, 3, 1, 5, 7),      # the data elements
                  +   nrow=2,               # number of rows
                  +   ncol=3,               # number of columns
                  +   byrow = TRUE)         # fill matrix by rows
> A


> A[2, 3]                                   # element at 2nd row, 3rd column
> A[2, ]                                    # the 2nd row
> A[ ,c(1,3)]                               # the 1st and 3rd columns
> t(A)                                      # transpose of B
> c(B)                                      # deconstruct a matrix
```

Functions for viewing matrices and dataframes and returning information about them.

Function	Description
head(x), tail(x)	Print the first few rows (or last few rows).
View(x)	Open the entire object in a new window
nrow(x), ncol(x), dim(x)	Count the number of rows and columns
rownames(), colnames(), names()	Show the row (or column) names
str(x), summary(x)	Show the structure of the dataframe (ie., dimensions and classes) and summary statistics

Introduction to Arrays in R

- In arrays, data is stored in the form of matrices, rows, and columns.



R Array Syntax

Array_NAME <- array(data, dim = (row_Size, column_Size, matrices, dimnames)

- **data –** Data is an input vector that is given to the array.
- **matrices –** Array in R consists of multi-dimensional matrices.
- **row_Size –** row_Size describes the number of row elements that an array can store.
- **column_Size –** Number of column elements that can be stored in an array.
- **dimnames –** Used to change the default names of rows and columns to the user's preference.

Arguments in Array

The array function in R can be written as:

array(data = NA, dim = length(data), dimname = NULL)

- **data** is a vector that provides data to fill the array.
- **dim** attribute provides maximum indices in each dimension
- **dimname** can be either NULL or can have a name for the array.

How to Create an Array in R

```
# Example:

# Create two vectors of different lengths.
vector1 <- c(2,8,3)
vector2 <- c(10,14,17,12,11,15)



# Take these vectors as input to the array.
result <- array(c(vector1,vector2), dim = c(3,3,2))

print(result)
```

Different Operations on Rows and Columns

Naming Columns And Rows

```
# Example:
# Create two vectors of different lengths.
vector1 <- c(2,8,3)
vector2 <- c(10,14,17,12,11,15)
column.names <- c("COL1","COL2","COL3")
row.names <- c("ROW1","ROW2","ROW3")
matrix.names <- c("Matrix1","Matrix2")


# Take these vectors as input to the array.
result <- array(c(vector1,vector2),dim = c(3,3,2), dimnames = list(row.names,
column.names,  matrix.names))


print(result)
```

```
# Print the third row of the first matrix of the array.
print(result[3,,1])

# Print the element in the 2nd row and 3rd column of the 2nd matrix.
print(result[2,3,2])

# Print the 2nd Matrix.
print(result[,,2])
```

Manipulating R Array Elements

```
# Example:

# Create two vectors of different lengths.
vector1 <- c(1,4,6)
vector2 <- c(2,3,5,6,7,9)

# Take these vectors as input to the array.
array1 <- array(c(vector1,vector2), dim = c(3,3,2))

# Create two vectors of different lengths.
vector3 <- c(6,4,1)
vector4 <- c(9,7,6,5,3,2)
array2 <- array(c(vector1,vector2), dim = c(3,3,2))
```

```
# create matrices from these arrays.
matrix1 <- array1[,,2]
matrix2 <- array2[,,2]

# Add the matrices.
result <- matrix1+matrix2

print(result)
```

Calculations across R Array Elements

- apply()function for calculations in an array in R.
- Syntax apply(x, margin, fun)

Following is the description of the parameters used:

- x is an array.
- a margin is the name of the dataset used.
- fun is the function to be applied to the elements of the array.
- For Example:

- We use the apply() function below in different ways. To calculate the sum of the elements in the rows of an array across all the matrices.

```
# Example:
# We will create two vectors of different lengths.

vector1 <- c(1,4,6)
vector2 <- c(2,3,5,6,7,9)

# Now, we will take these vectors as input to the array.
new.array <- array(c(vector1,vector2),dim = c(3,3,2))
print(new.array)

# Use apply to calculate the sum of the rows across all the matrices.
result <- apply(new.array, c(1), sum)
print(result)
```

R Data Frames:

- Data frames combine the behavior of lists and matrices to make a structure ideally suited for the needs of statistical data. The data frame is a list of vectors which are of equal length.
- A data-frame must have column names and every row should have a unique name.
  - names(), colnames(), and rownames()
- Each column must have the identical number of items.
- Each item in a single column must be of the same data type.
- Different columns may have different data types.
- A matrix contains only one type of data, while a data frame accepts different data types (numeric, character, factor, etc.). This makes it a 2-dimensional structure, so it shares properties of both the matrix and the list.

How to Create a Data Frame

- We can create a data frame by passing the variable a,b,c,d into the data.frame() function. We can name the columns with name() and simply specify the name of the variables.

- data.frame(df, stringsAsFactors = TRUE)

Arguments:

- df: It can be a matrix to convert as a data frame or a collection of variables to join
- stringsAsFactors: Convert string to factor by default

```r
df <- data.frame(x = 1:3, y = c("a", "b", "c"))
str(df)

#> 'data.frame': 3 obs. of 2 variables:
#> $ x: int 1 2 3
#> $ y: Factor w/ 3 levels "a","b","c": 1 2 3
```

```r
df <- data.frame(
        x = 1:3,
        y = c("a", "b", "c"),
        stringsAsFactors = FALSE)
str(df)

#> 'data.frame': 3 obs. of 2 variables:
#> $ x: int 1 2 3
#> $ y: chr "a" "b" "c"
```

```
# Example R program to illustrate dataframe

# A vector which is a character vector
Name = c("Auriel", "Ray", "Asia")

# A vector which is a character vector
Type = c("O+", "B-", "A-")

# A vector which is a numeric vector
Age = c(36, 23, 62)

# To create dataframe use data.frame command
# and then pass each of the vectors
# we have created as arguments
# to the function data.frame()

df = data.frame(Name, Type, Age)

print(df)
```

Because a data.frame is an S3 class, its type reflects the underlying vector used to build it: the list. To check if an object is a data frame, use class() or test explicitly with is.data.frame():

```
typeof(df)
#> [1] "list"
class(df)
#> [1] "data.frame"
is.data.frame(df)
#> [1] TRUE
```

You can coerce an object to a data frame with as.data.frame():

- A vector will create a one-column data frame.
- A list will create one column for each element; it's an error if they're not all the same length.
- A matrix will create a data frame with the same number of columns and rows as the matrix.

Merging: Combine data frames using cbind() and rbind()

```
cbind(df, data.frame(z = 3:1))
#> x y z
#> 1 1 a 3
#> 2 2 b 2
#> 3 3 c 1

rbind(df, data.frame(x = 10, y = "z"))
#> x y
#> 1 1 a
#> 2 2 b
#> 3 3 c
#> 4 10 z
```

- When combining column-wise, the number of rows must match, but row names are ignored.
- When combining row-wise, both the number and names of columns must match.
- Use plyr::rbind.fill() to combine data frames that don't have the same columns.

It’s a common mistake to try and create a data frame by cbind()ing vectors together. This doesn’t work because cbind() will create a matrix unless one of the arguments is already a data frame. Instead use data.frame() directly:

```
bad <- data.frame(cbind(a = 1:2, b = c("a", "b")))
str(bad)
#> 'data.frame': 2 obs. of 2 variables:
#> $ a: Factor w/ 2 levels "1","2": 1 2
#> $ b: Factor w/ 2 levels "a","b": 1 2

good <- data.frame(a = 1:2, b = c("a", "b"), stringsAsFactors = FALSE)
str(good)
#> 'data.frame': 2 obs. of 2 variables:
#> $ a: int 1 2
#> $ b: chr "a" "b"
```

Example:

Let's create a factor data frame.

```
# Create gender vector
gender_vector <- c("Male", "Female", "Female", "Male", "Male")
class(gender_vector)

# Convert gender_vector to a factor
factor_gender_vector <-factor(gender_vector)
class(factor_gender_vector)
```

Output:

```
## [1] "character"
## [1] "factor"
```

Examine a Data Frame in R with 7 Basic Functions

- dim(): shows the dimensions of the data frame by row and column
- str(): shows the structure of the data frame
- summary(): provides summary statistics on the columns of the data frame
- colnames(): shows the name of each column in the data frame
- head(): shows the first 6 rows of the data frame
- tail(): shows the last 6 rows of the data frame
- View(): shows a spreadsheet-like display of the entire data frame

Fixing a broken data frame

Some useful functions:

- ?read.csv
- head()
- str()
- class()
- unique()
- levels()
- which()
- droplevels()

Note: For these functions you have to put the name of the data object in the parentheses (i.e., head(CO2)).

Also remember that you can use “?” to look up help for a function (i.e. ?str).

Knowledge Check:

1. What are the three properties of a vector, other than its contents?
   The three properties of a vector are type, length, and attributes.

2a. What are the four common types of atomic vectors?
   1. Logical
   2. Integer
   3. Double (sometimes called numeric)
   4. Character

2b. What are the two rare types?
   1. Complex
   2. Raw

Knowledge Check (cont.)

3. What are attributes? How do you get them and set them?

Attributes allow you to associate arbitrary additional metadata to any object. You can get and set individual attributes with attr(x, "y") and attr(x, "y") <- value; or get and set all attributes at once with attributes().

4a. How is a list different from an atomic vector?

The elements of a list can be any type (even a list); the elements of an atomic vector are all of the same type.

4b. How is a matrix different from a data frame?

Similarly, every element of a matrix must be the same type; in a data frame, the different columns can have different types

5a. Can you have a list that is a matrix?

You can make “list-array” by assigning dimensions to a list.

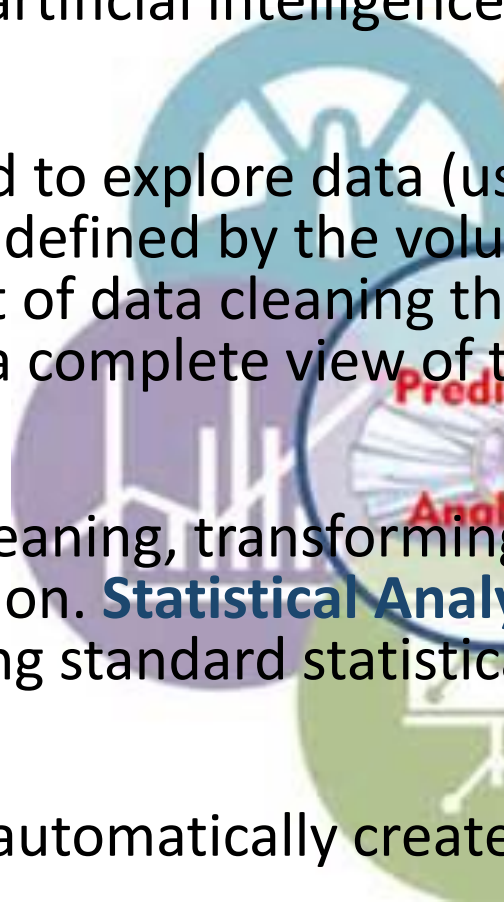
Module 3: Data Science

BeVera

Data Mining? Statistical Modeling? Predictive Analytics?

- **Predictive analytics** is the branch of advanced analytics which is used to make predictions about unknown future events. Predictive analytics uses techniques from data mining, statistics, modeling, machine learning, and artificial intelligence.

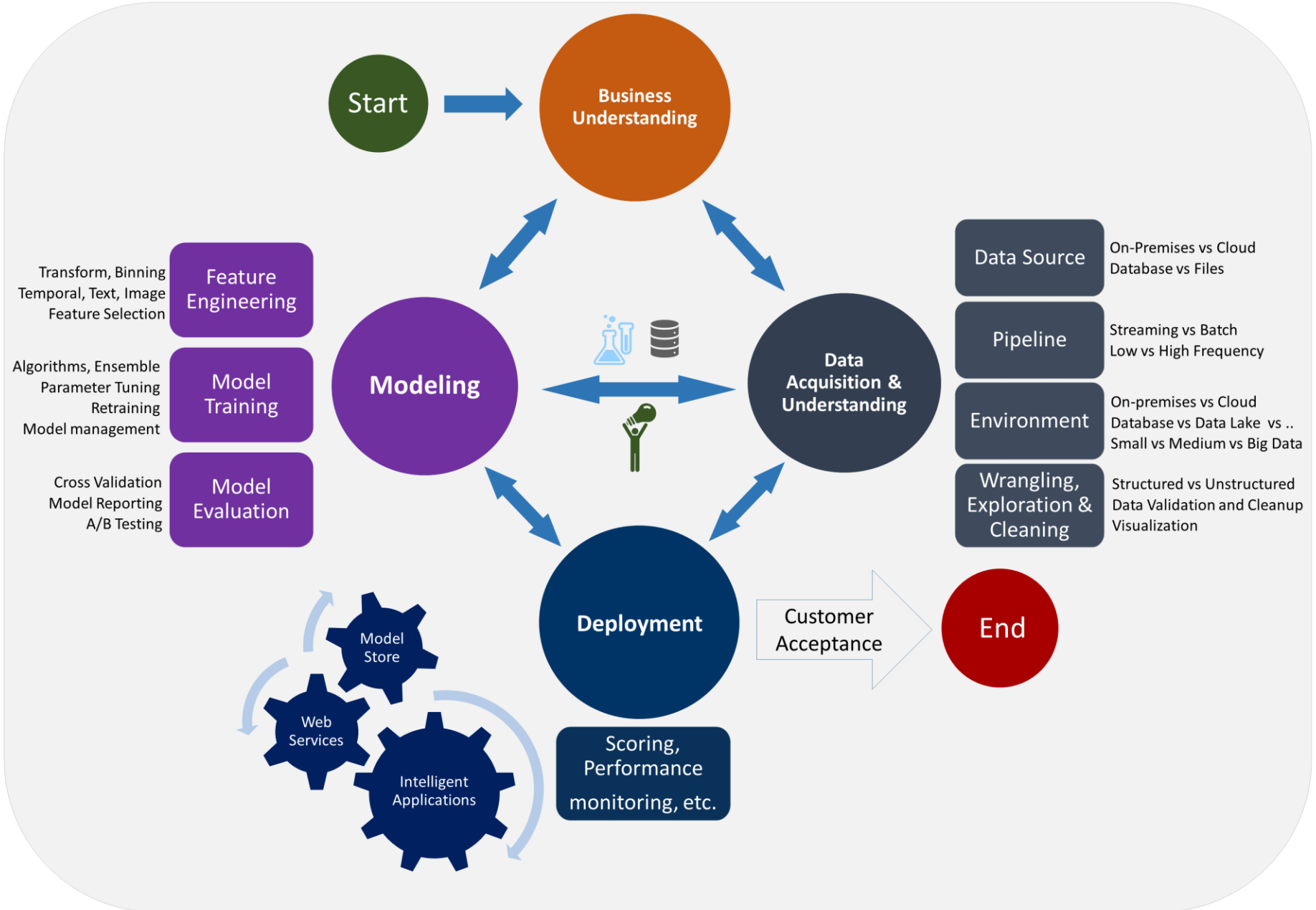
- **Data mining** is an analytics process designed to explore data (usually large amounts of data – also known as "big data"). Big data is defined by the volume, velocity, variety, variability and veracity. Data mining is a part of data cleaning that prepares data from multiple sources for analysis. This provides a complete view of the customer interactions.



- Data analysis is the process of inspecting, cleaning, transforming and modeling data with the objective of discovering useful information. **Statistical Analysis** enables to validate assumptions, hypothesis, and test them using standard statistical models.

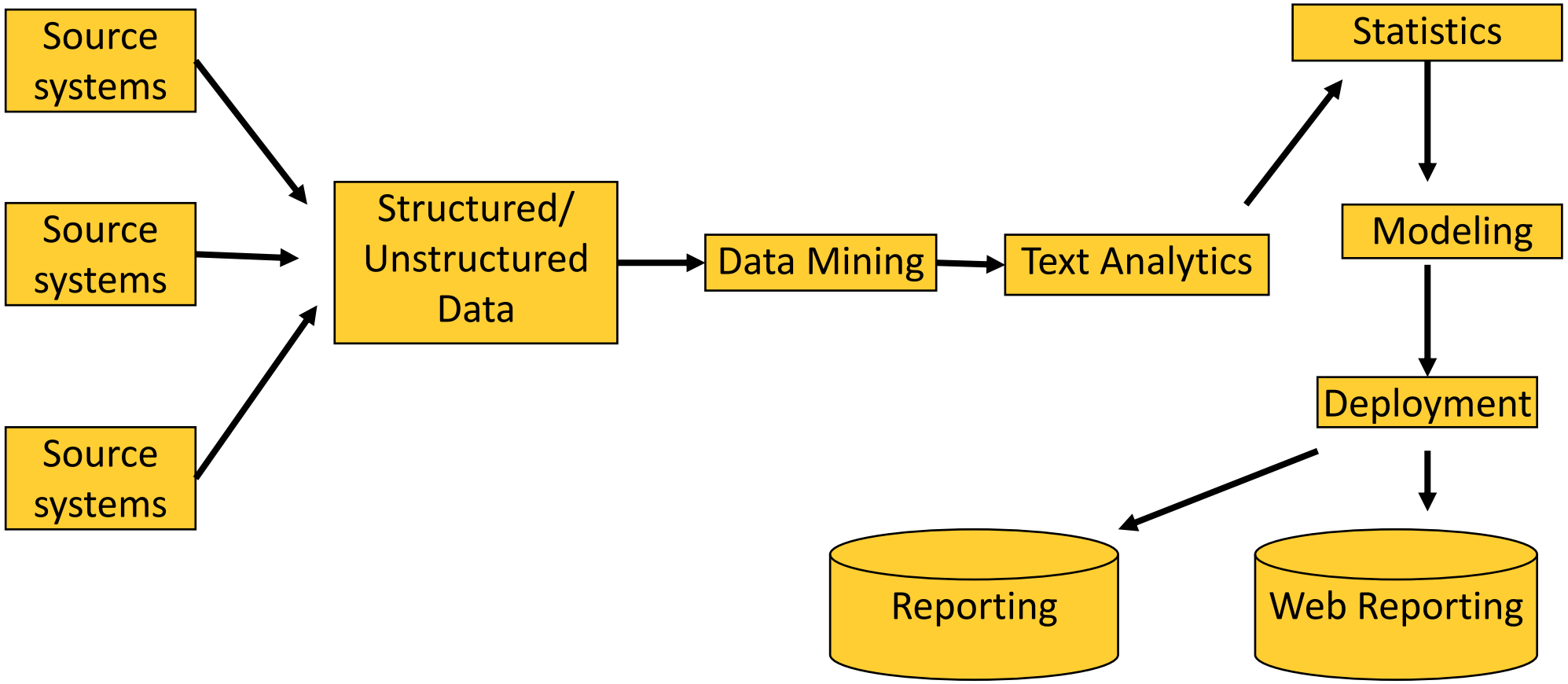
- **Predictive modeling** provides the ability to automatically create accurate predictive models about future.

Data Science Lifecycle



Data Mining? Statistical Modeling? Predictive Analytics?

Predictive Analytics



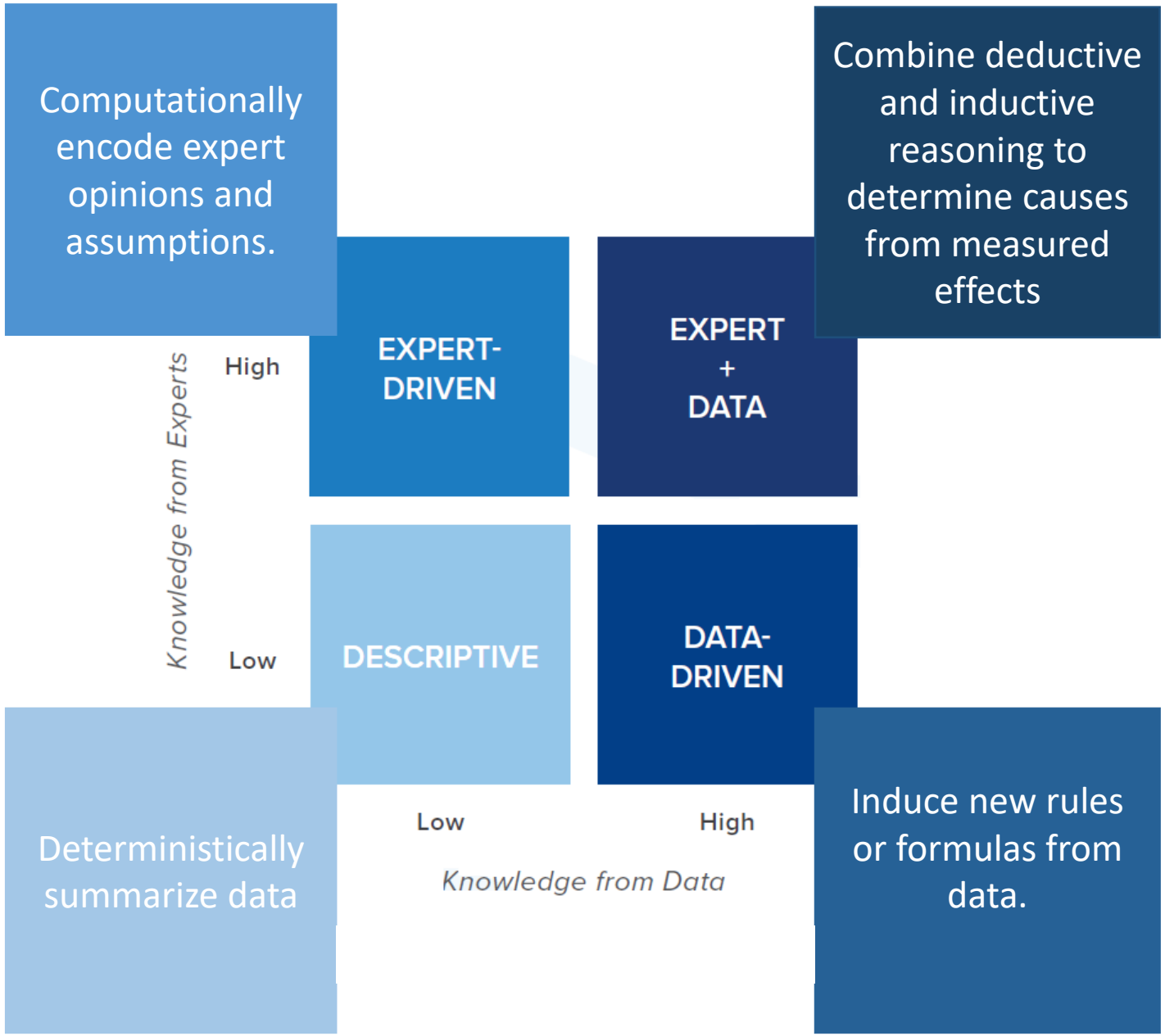
4 Levels of Knowledge Model

https://www.elderresearch.com/consulting-services/ten-levels-of-analytics

Sources of knowledge whether data or expert are independent. Combine data- and expert-driven approaches to maximize insight and value.

Expert-driven approaches is preferred if:

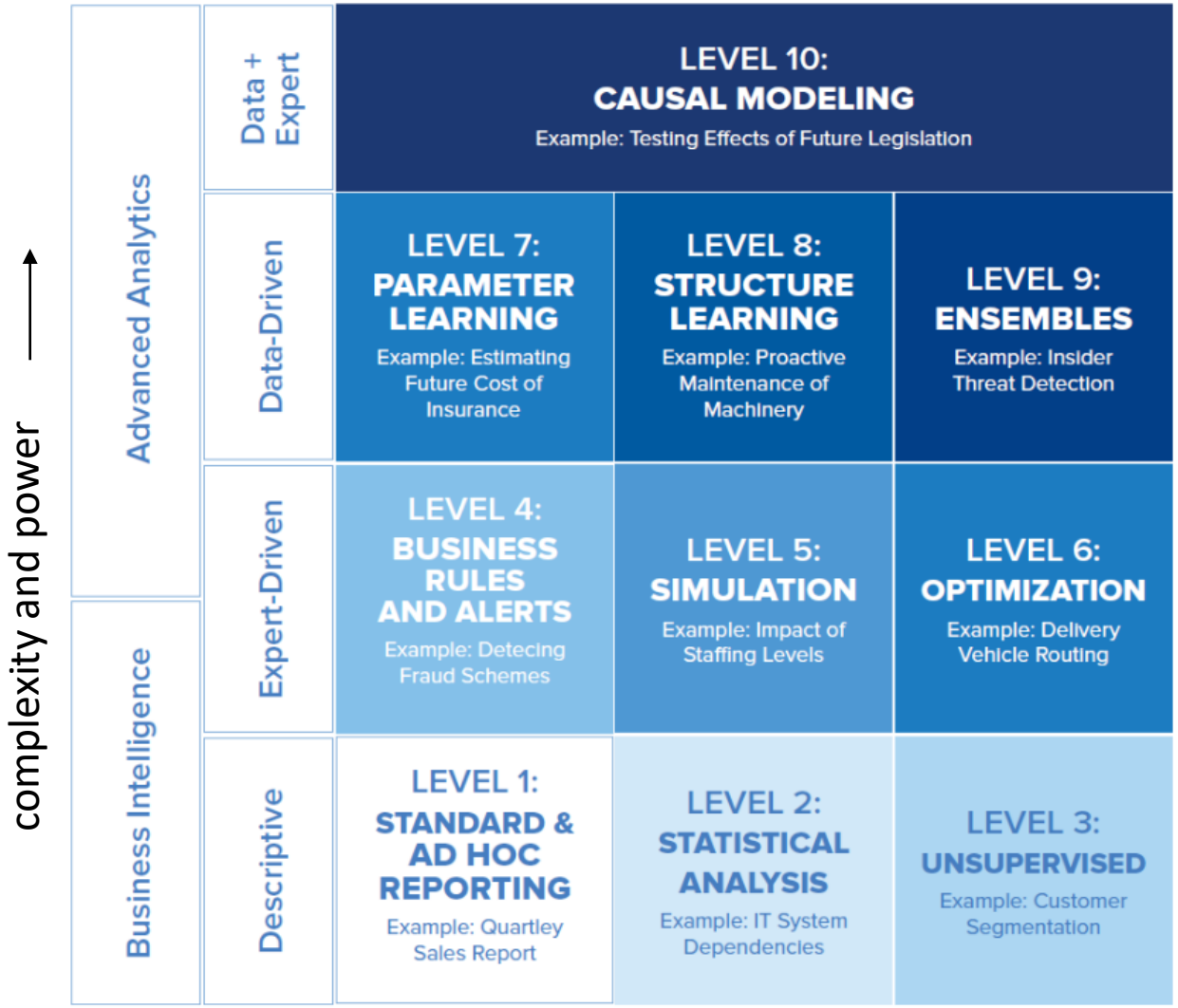
- an accepted premise
- the data is filtered
- poorly represents the full situation



Data –driven modeling:

- is inductive
- graded superior to expert-driven ones.
- allow unknown rules or relationships to be discovered from the data
- are less susceptible to the biases and misconceptions common to human reasoning.

10 Levels of Analytics



https://www.elderresearch.com/consulting-services/ten-levels-of-analytics

Analytic Techniques and their Disciplines

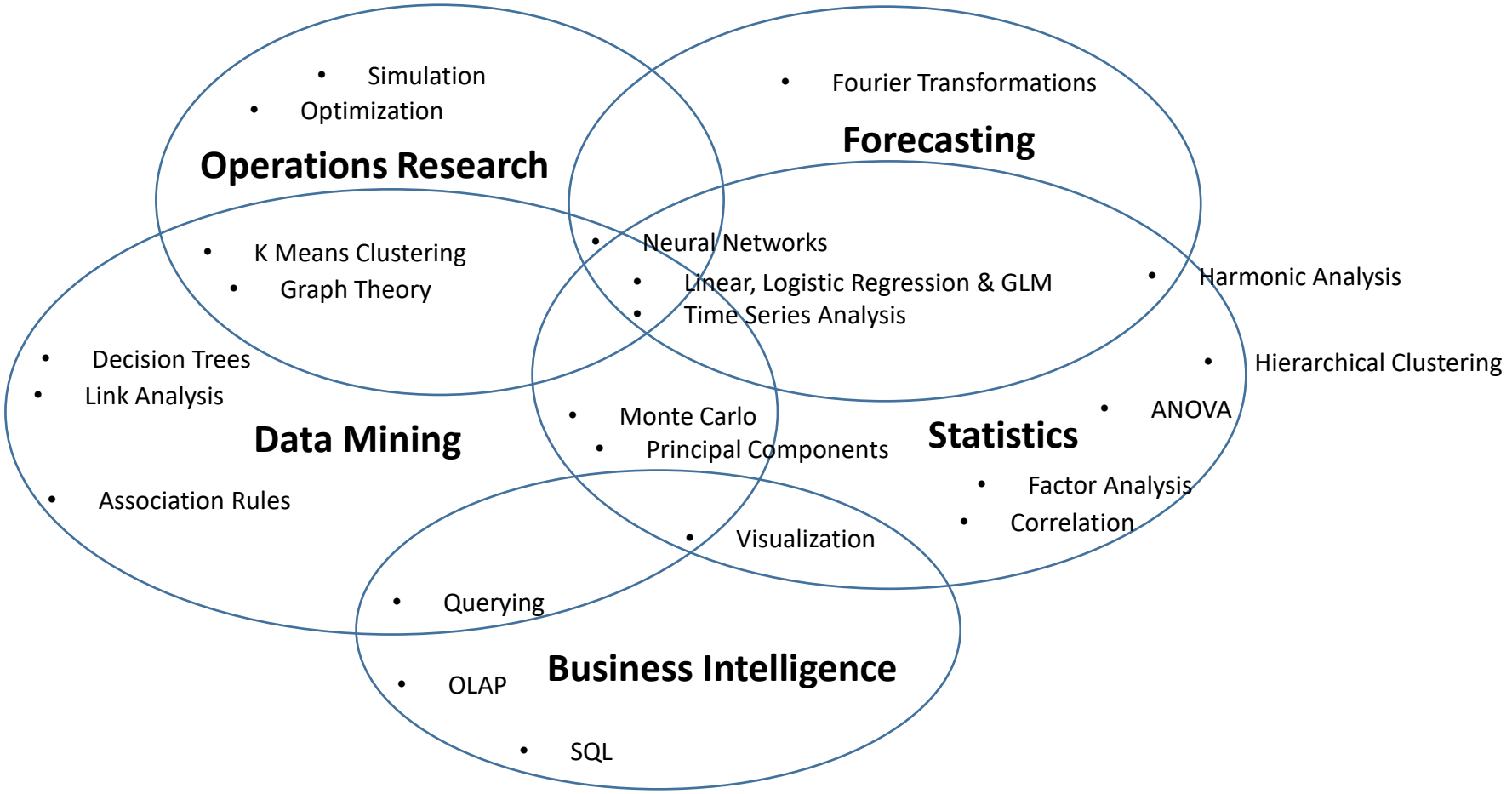
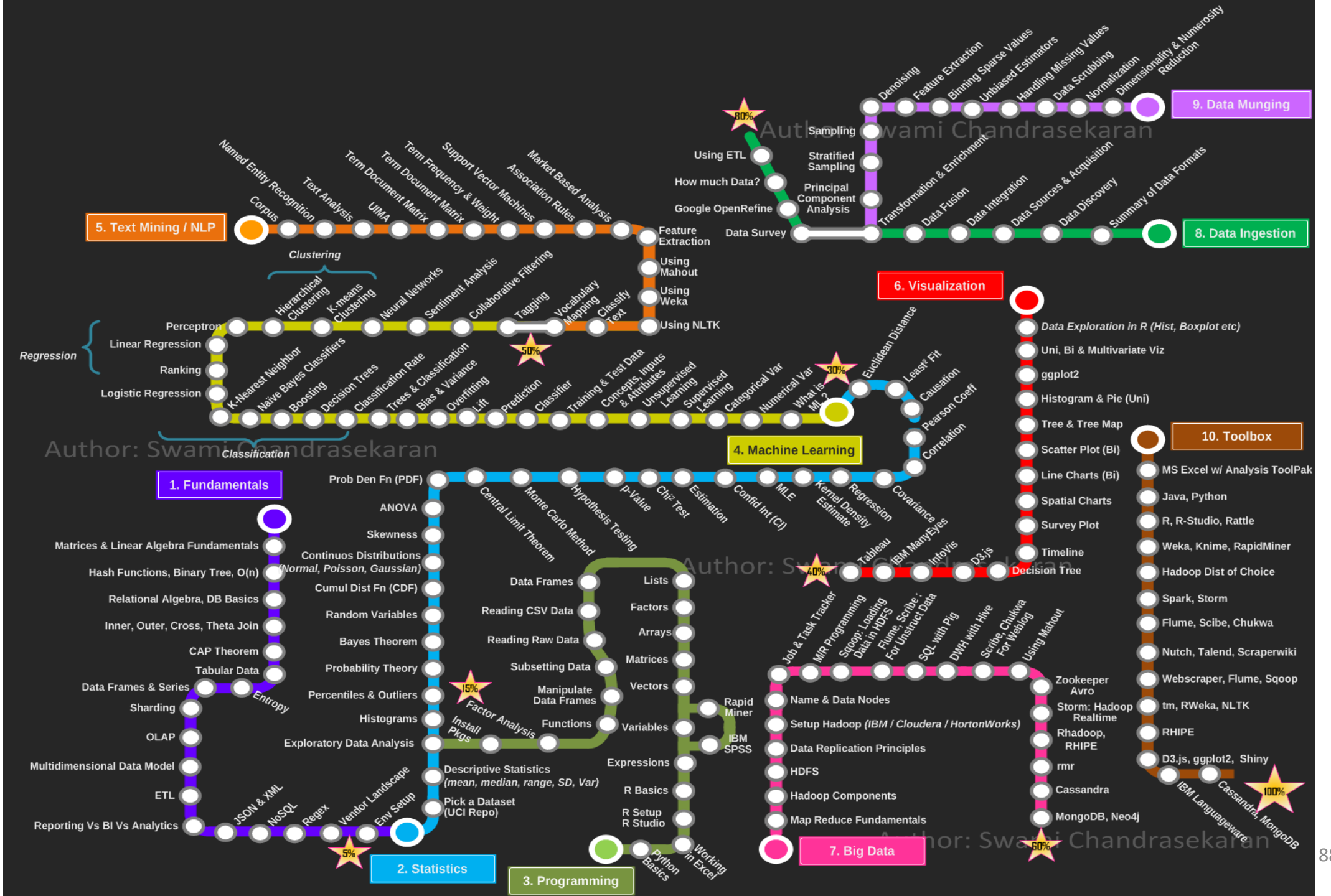


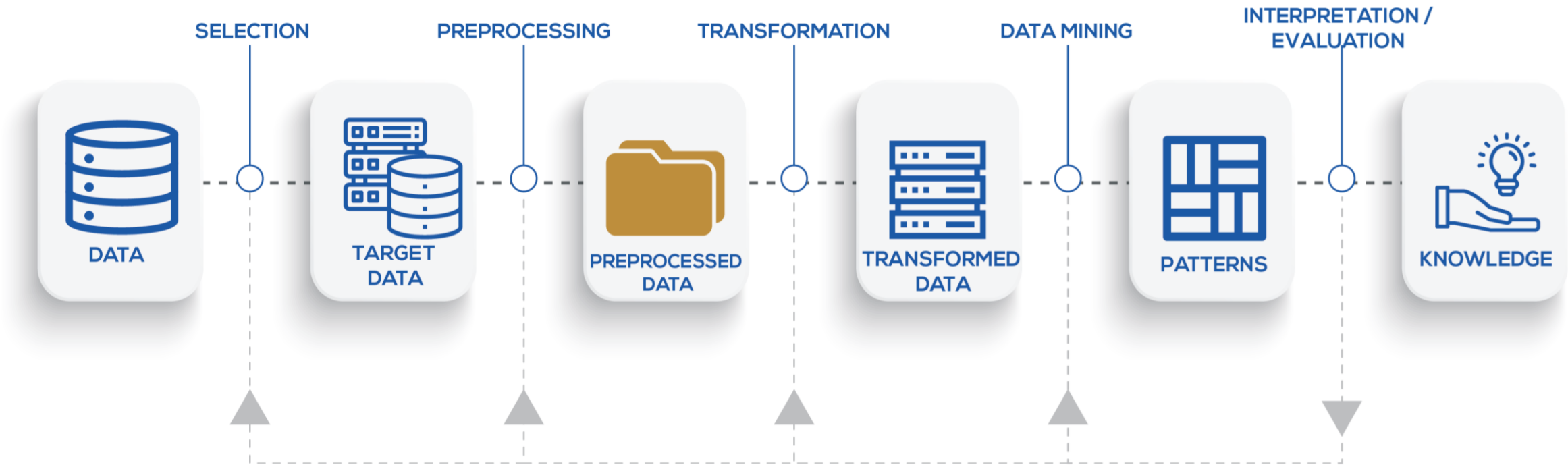
**Diagram developed by John Elder and Dustin Hux*


1. Fundamentals
Matrices & Linear Algebra Fundamentals
Hash Functions, Binary Tree, O(n)
Relational Algebra, DB Basics
Inner, Outer, Cross, Theta Join
CAP Theorem
Tabular Data
Data Frames & Series
Entropy
Sharding
OLAP
Multidimensional Data Model
ETL
Reporting Vs BI Vs Analytics
JSON & XML
NoSQL
Regex
Vendor Landscape
Env Setup
5%
2. Statistics
Prob Den Fn (PDF)
ANOVA
Skewness
Continuos Distributions (Normal, Poisson, Gaussian)
Cumul Dist Fn (CDF)
Random Variables
Bayes Theorem
Probability Theory
Percentiles & Outliers
Histograms
Exploratory Data Analysis
Descriptive Statistics (mean, median, range, SD, Var)
Pick a Dataset (UCI Repo)
Central Limit Theorem
Monte Carlo Method
Hypothesis Testing
p-Value
Chi² Test
Estimation
Confid Int (CI)
MLE
Kernel Density Estimate
Regression
Covariance
15%
Factor Analysis
Install Pkgs
3. Programming
Data Frames
Reading CSV Data
Reading Raw Data
Subsetting Data
Manipulate Data Frames
Functions
Lists
Factors
Arrays
Matrices
Vectors
Variables
Expressions
R Basics
R Setup
R Studio
Rapid Miner
IBM SPSS
Python Basics
Working in Excel
4. Machine Learning
Regression
Perceptron
Linear Regression
Ranking
Logistic Regression
Classification
K-Nearest Neighbor
Naive Bayes Classifiers
Boosting
Decision Trees
Classification Rate
Trees & Classification
Bias & Variance
Overfitting
Lift
Prediction
Classifier
Training & Test Data
Concepts, Inputs & Attributes
Unsupervised Learning
Supervised Learning
Categorical Var
Numerical Var
What is ML?
50%
Clustering
Hierarchical Clustering
K-means Clustering
Neural Networks
Sentiment Analysis
Collaborative Filtering
Tagging
Vocabulary Mapping
Classify Text
Using NLTK
Using Mahout
Using Weka
5. Text Mining / NLP
Corpus
Named Entity Recognition
Text Analysis
UIMA
Term Document Matrix
Term Document Matrix
Term Frequency & Weight
Support Vector Machines
Association Rules
Market Based Analysis
Feature Extraction
30%
Euclidean Distance
Least² Fit
Causation
Pearson Coeff
Correlation
6. Visualization
Data Exploration in R (Hist, Boxplot etc)
Uni, Bi & Multivariate Viz
ggplot2
Histogram & Pie (Uni)
Tree & Tree Map
Scatter Plot (Bi)
Line Charts (Bi)
Spatial Charts
Survey Plot
Timeline
Decision Tree
40%
Tableau
IBM ManyEyes
InfoVis
D3.js
7. Big Data
Job & Task Tracker
MR Programming
Sqoop: Loading Data in HDFS
Flume, Scribe : For Unstruct Data
SQL with Pig
DWH with Hive
Scribe, Chukwa For Weblog
Using Mahout
Name & Data Nodes
Setup Hadoop (IBM / Cloudera / HortonWorks)
Data Replication Principles
HDFS
Hadoop Components
Map Reduce Fundamentals
Zookeeper Avro
Storm: Hadoop Realtime
Rhadoop, RHIPE
rmr
Cassandra
MongoDB, Neo4j
50%
8. Data Ingestion
Sampling
Stratified Sampling
Principal Component Analysis
Transformation & Enrichment
Data Fusion
Data Integration
Data Sources & Acquisition
Data Discovery
Summary of Data Formats
Data Survey
80%
Using ETL
How much Data?
Google OpenRefine
9. Data Munging
Denoising
Feature Extraction
Binning Sparse Values
Unbiased Estimators
Handling Missing Values
Data Scrubbing
Normalization
Dimensionality & Numerosity Reduction
10. Toolbox
MS Excel w/ Analysis ToolPak
Java, Python
R, R-Studio, Rattle
Weka, Knime, RapidMiner
Hadoop Dist of Choice
Spark, Storm
Flume, Scibe, Chukwa
Nutch, Talend, Scraperwiki
Webscraper, Flume, Sqoop
tm, RWeka, NLTK
RHIPE
D3.js, ggplot2, Shiny
IBM Languageware
Cassandra, MongoDB
100%
Author: Swami Chandrasekaran

Module 4: R for Data Science

BeVera

Knowledge Discovery Process



R is Infinitely Expandable

Applications of R normally use a package; i.e., a library of special functions designed for a specific problem.

Hundreds of packages are available, mostly written by users.

A user normally only loads a handful of packages for a particular analysis (e.g., library(caret)).

Standards determine how a package is structured, works well with other packages and creates new data types in an easily used manner.

Standardization makes it easy for users to learn new packages.

Importing your data

R allows for the import of different data formats using specific packages that can make your job easier:

Package	Purpose
readr	Importing flay files
readxl	Getting excel files into R
haven	Import SAS, STATA and SPSS data files into R.
RMySQL and RpostgreSQL	Connect to databases, access and manipulate via DBI
rvest	webscraping

Example: mydata=read.csv(file="c:/…..csv", header=TRUE)

https://www.datacamp.com/community/tutorials/r-data-import-tutorial

Manipulating your data

Package	Purpose
tidyr	tidying your data
stringr	string manipulation
dplyr	to learn data frame like objects, solve data manipulation challenges
data.table	perform heavy data wrangling tasks
zoo, xts, quantmod	performing time series analysis

Using packages

1

```
install.packages("haven")
```

Downloads files to computer

1 x per computer

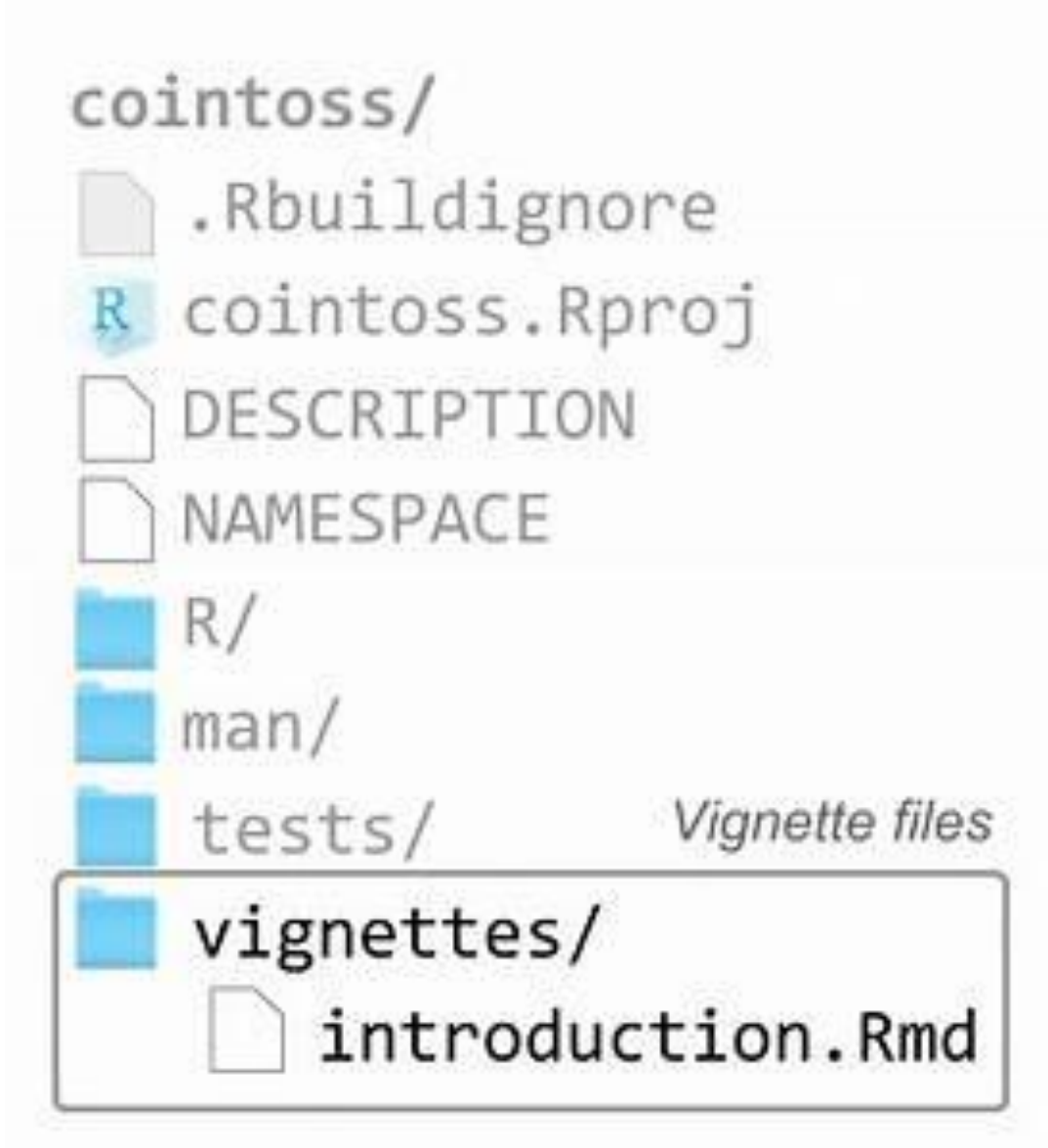
2

```
library("haven")
```

Loads package

1 x per R Session

Vignettes

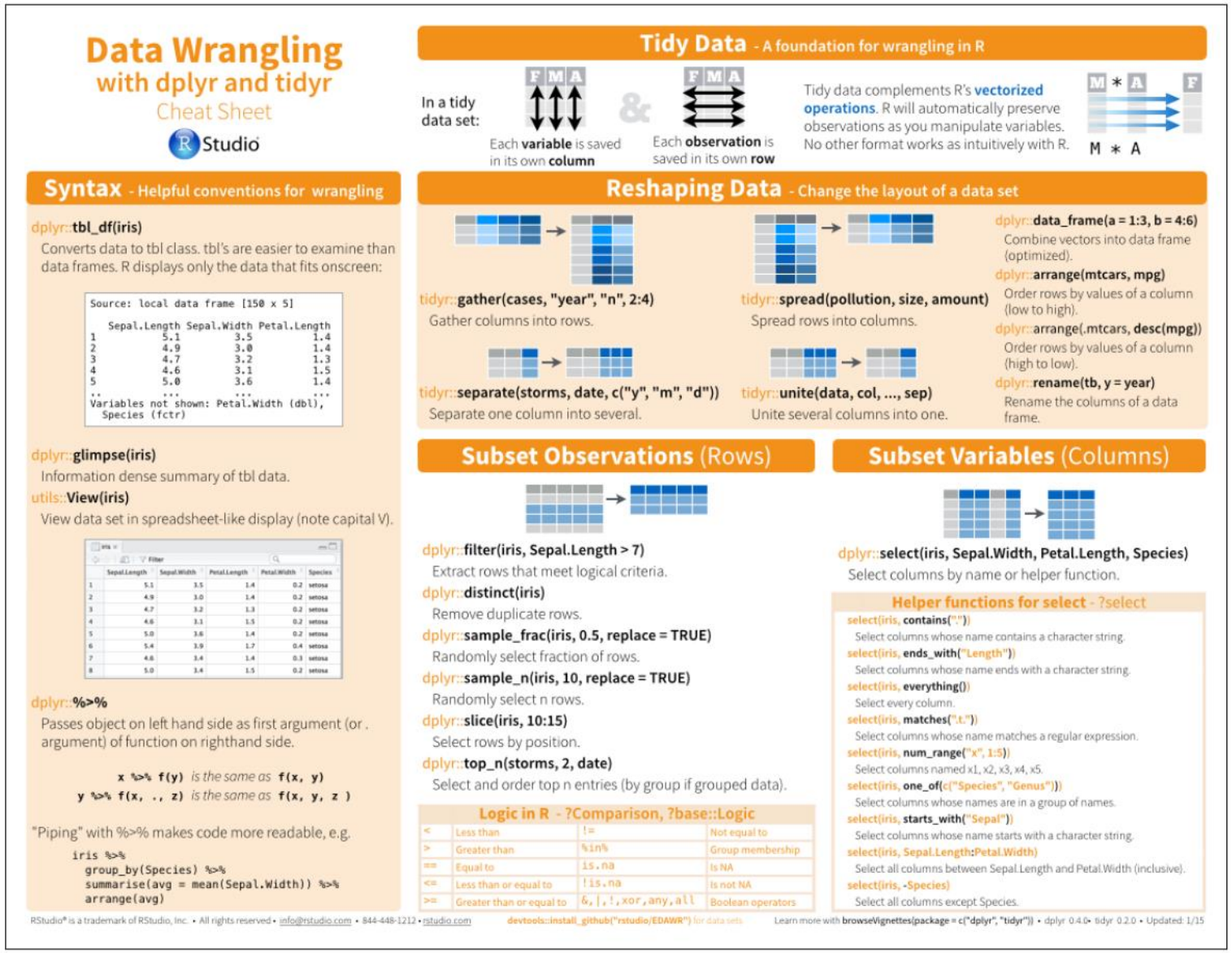

cointoss/
.Rbuildignore
cointoss.Rproj
DESCRIPTION
NAMESPACE
R/
man/
tests/
vignettes/
introduction.Rmd
Vignette files

Additional resources for R programming basics

http://www.sthda.com/english/wiki/easy-r-programming-basics

https://www.rstudio.com/resources/webinars/data-wrangling-with-r-and-rstudio/

https://github.com/rstudio/webinars/blob/master/05-Data-Wrangling-with-R-and-RStudio/wrangling-webinar.pdf



Data Wrangling with dplyr and tidyr

Cheat Sheet

RStudio

Tidy Data - A foundation for wrangling in R

In a tidy data set:

Each **variable** is saved in its own **column**

&

Each **observation** is saved in its own **row**

Tidy data complements R's **vectorized operations**. R will automatically preserve observations as you manipulate variables. No other format works as intuitively with R.

M * A → F

M * A

Syntax - Helpful conventions for wrangling

dplyr::**tbl_df(iris)**

Converts data to tbl class. tbl's are easier to examine than data frames. R displays only the data that fits onscreen:

```
Source: local data frame [150 x 5]

   Sepal.Length Sepal.Width Petal.Length
1           5.1         3.5          1.4
2           4.9         3.0          1.4
3           4.7         3.2          1.3
4           4.6         3.1          1.5
5           5.0         3.6          1.4
..          ...         ...          ...
Variables not shown: Petal.Width (dbl),
  Species (fctr)
```

dplyr::**glimpse(iris)**

Information dense summary of tbl data.

utils::**View(iris)**

View data set in spreadsheet-like display (note capital V).

dplyr::**%>%**

Passes object on left hand side as first argument (or . argument) of function on righthand side.

`x %>% f(y)` *is the same as* `f(x, y)`

`y %>% f(x, ., z)` *is the same as* `f(x, y, z )`

"Piping" with %>% makes code more readable, e.g.

```
iris %>%
  group_by(Species) %>%
  summarise(avg = mean(Sepal.Width)) %>%
  arrange(avg)
```

Reshaping Data - Change the layout of a data set

tidyr::**gather(cases, "year", "n", 2:4)**

Gather columns into rows.

tidyr::**spread(pollution, size, amount)**

Spread rows into columns.

tidyr::**separate(storms, date, c("y", "m", "d"))**

Separate one column into several.

tidyr::**unite(data, col, ..., sep)**

Unite several columns into one.

dplyr::**data_frame(a = 1:3, b = 4:6)**

Combine vectors into data frame (optimized).

dplyr::**arrange(mtcars, mpg)**

Order rows by values of a column (low to high).

dplyr::**arrange(.mtcars, desc(mpg))**

Order rows by values of a column (high to low).

dplyr::**rename(tb, y = year)**

Rename the columns of a data frame.

Subset Observations (Rows)

dplyr::**filter(iris, Sepal.Length > 7)**

Extract rows that meet logical criteria.

dplyr::**distinct(iris)**

Remove duplicate rows.

dplyr::**sample_frac(iris, 0.5, replace = TRUE)**

Randomly select fraction of rows.

dplyr::**sample_n(iris, 10, replace = TRUE)**

Randomly select n rows.

dplyr::**slice(iris, 10:15)**

Select rows by position.

dplyr::**top_n(storms, 2, date)**

Select and order top n entries (by group if grouped data).

Logic in R - ?Comparison, ?base::Logic

<	Less than	!=	Not equal to
>	Greater than	%in%	Group membership
==	Equal to	is.na	Is NA
<=	Less than or equal to	!is.na	Is not NA
>=	Greater than or equal to	&, |, !, xor, any, all	Boolean operators

Subset Variables (Columns)

dplyr::**select(iris, Sepal.Width, Petal.Length, Species)**

Select columns by name or helper function.

Helper functions for select - ?select

select(iris, contains("."))
Select columns whose name contains a character string.

select(iris, ends_with("Length"))
Select columns whose name ends with a character string.

select(iris, everything())
Select every column.

select(iris, matches(".t."))
Select columns whose name matches a regular expression.

select(iris, num_range("x", 1:5))
Select columns named x1, x2, x3, x4, x5.

select(iris, one_of(c("Species", "Genus")))
Select columns whose names are in a group of names.

select(iris, starts_with("Sepal"))
Select columns whose name starts with a character string.

select(iris, Sepal.Length:Petal.Width)
Select all columns between Sepal.Length and Petal.Width (inclusive).

select(iris, -Species)
Select all columns except Species.

devtools::install_github("rstudio/EDAWR") for data sets

Learn more with browseVignettes(package = c("dplyr", "tidyr")) • dplyr 0.4.0 • tidyr 0.2.0 • Updated: 1/15

Also available for:

- Base R
- Advanced R
- Data Table
- Devtools
- ggplot2
- R Markdown
- Regular Expressions
- Rstudio IDE
- Shiny

http://www.rstudio.com/resources/cheatsheets/

R For Data Science *Cheat Sheet*

Tidyverse for Beginners

Learn More R for Data Science Interactively at www.datacamp.com

Tidyverse

The **tidyverse** is a powerful collection of R packages that are actually data tools for transforming and visualizing data. All packages of the tidyverse share an underlying philosophy and common APIs.

The core packages are:

- **ggplot2**, which implements the grammar of graphics. You can use it to visualize your data.
- **dplyr** is a grammar of data manipulation. You can use it to solve the most common data manipulation challenges.
- **tidyr** helps you to create tidy data or data where each variable is in a column, each observation is a row end each value is a cell.
- **readr** is a fast and friendly way to read rectangular data.
- **purrr** enhances R's functional programming (FP) toolkit by providing a complete and consistent set of tools for working with functions and vectors.
- **tibble** is a modern re-imaginging of the data frame.
- **stringr** provides a cohesive set of functions designed to make working with strings as easy as posssible
- **forcats** provide a suite of useful tools that solve common problems with factors.

You can install the complete tidyverse with:

```
> install.packages("tidyverse")
```

Then, load the core tidyverse and make it available in your current R session by running:

```
> library(tidyverse)
```

Note: there are many other tidyverse packages with more specialised usage. They are not loaded automatically with library(tidyverse), so you'll need to load each one with its own call to `library()`.

Useful Functions

`> tidyverse_conflicts()`	Conflicts between tidyverse and other packages
`> tidyverse_deps()`	List all tidyverse dependencies
`> tidyverse_logo()`	Get tidyverse logo, using ASCII or unicode characters
`> tidyverse_packages()`	List all tidyverse packages
`> tidyverse_update()`	Update tidyverse packages

Loading in the data

`> library(datasets)`	Load the datasets package
`> library(gapminder)`	Load the gapminder package
`> attach(iris)`	Attach iris data to the R search path

dplyr

Filter

`filter()` allows you to select a subset of rows in a data frame.

`> iris %>%` `  filter(Species=="virginica")`	Select iris data of species "virginica"

`> iris %>%`
`  filter(Species=="virginica",`
`  Sepal.Length > 6)`
	Select iris data of species "virginica" and sepal length greater than 6.

Arrange

`arrange()` sorts the observations in a dataset in ascending or descending order based on one of its variables.

`> iris %>%` `  arrange(Sepal.Length)`	Sort in ascending order of sepal length
`> iris %>%` `  arrange(desc(Sepal.Length))`	Sort in descending order of sepal length

Combine multiple `dplyr` verbs in a row with the pipe operator `%>%`:

`> iris %>%`
`  filter(Species=="virginica") %>%`
`  arrange(desc(Sepal.Length))`
	Filter for species "virginica" then arrange in descending order of sepal length

Mutate

`mutate()` allows you to update or create new columns of a data frame.

`> iris %>%` `  mutate(Sepal.Length=Sepal.Length*10)`	Change Sepal.Length to be in millimeters
`> iris %>%` `  mutate(SLMm=Sepal.Length*10)`	Create a new column called SLMm

Combine the verbs `filter()`, `arrange()`, and `mutate()`:

```
> iris %>%
  filter(Species=="Virginica") %>%
  mutate(SLMm=Sepal.Length*10) %>%
  arrange(desc(SLMm))
```

Summarize

`summarize()` allows you to turn many observations into a single data point.

`> iris %>%` `  summarize(medianSL=median(Sepal.Length))`	Summarize to find the median sepal length

`> iris %>%`
`  filter(Species=="virginica") %>%`
`  summarize(medianSL=median(Sepal.Length))`
	Filter for virginica then summarize the median sepal length

You can also summarize multiple variables at once:

```
> iris %>%
  filter(Species=="virginica") %>%
  summarize(medianSL=median(Sepal.Length),
            maxSL=max(Sepal.Length))
```

`group_by()` allows you to summarize within groups instead of summarizing the entire dataset:

`> iris %>%`
`  group_by(Species) %>%`
`  summarize(medianSL=median(Sepal.Length),`
`  maxSL=max(Sepal.Length))`
	Find median and max sepal length of each species

`> iris %>%`
`  filter(Sepal.Length>6) %>%`
`  group_by(Species) %>%`
`  summarize(medianPL=median(Petal.Length),`
`  maxPL=max(Petal.Length))`
	Find median and max petal length of each species with sepal length > 6

ggplot2

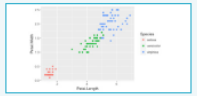
Scatter plot

Scatter plots allow you to compare two variables within your data. To do this with ggplot2, you use `geom_point()`

`> iris_small <- iris %>%`
`  filter(Sepal.Length > 5)`
`> ggplot(iris_small, aes(x=Petal.Length,`
`  y=Petal.Width)) +`
`  geom_point()`
	Compare petal width and length

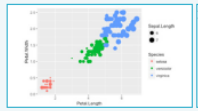
Additional Aesthetics

- Color



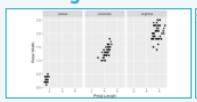
```
> ggplot(iris_small, aes(x=Petal.Length,
                         y=Petal.Width,
                         color=Species)) +
  geom_point()
```

- Size



```
> ggplot(iris_small, aes(x=Petal.Length,
                         y=Petal.Width,
                         color=Species,
                         size=Sepal.Length)) +
  geom_point()
```

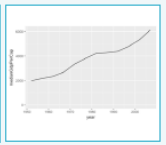
Faceting



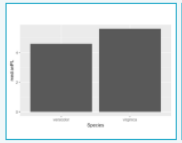
```
> ggplot(iris_small, aes(x=Petal.Length,
                         y=Petal.Width)) +
  geom_point()+
  facet_wrap(~Species)
```

Line Plots

```
> by_year <- gapminder %>%
  group_by(year) %>%
  summarize(medianGdpPerCap=median(gdpPercap))
> ggplot(by_year, aes(x=year,
                      y=medianGdpPerCap))+
  geom_line()+
  expand_limits(y=0)
```



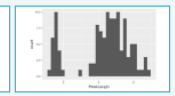
Bar Plots



```
> by_species <- iris %>%
  filter(Sepal.Length>6) %>%
  group_by(Species) %>%
  summarize(medianPL=median(Petal.Length))
> ggplot(by_species, aes(x=Species,
                         y=medianPL)) +
  geom_col()
```

Histograms

```
> ggplot(iris_small, aes(x=Petal.Length))+
  geom_histogram()
```

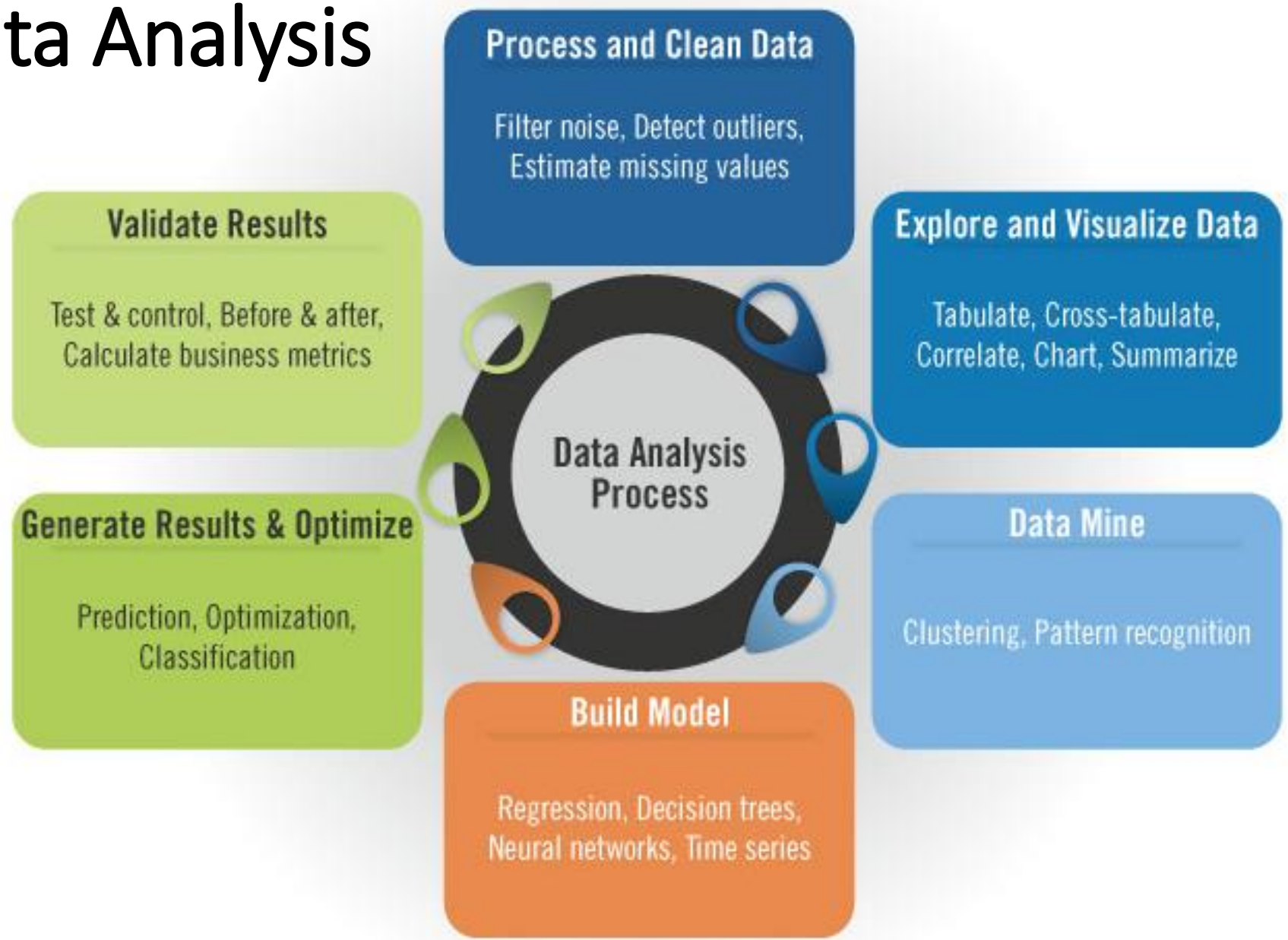


Box Plots

```
> ggplot(iris_small, aes(x=Species,
                         y=Sepal.Width))+
  geom_boxplot()
```

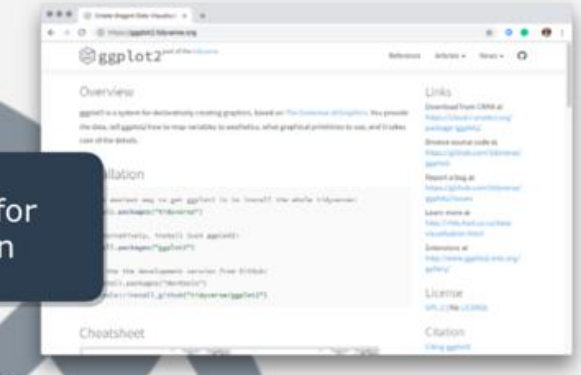
DataCamp
Learn R for Data Science Interactively

Steps of Data Analysis



Data Science with R Workflow

The Data Science With R Workflow is available in the book: R For Data Science. If you want to learn R and this workflow *for business analysis*, take the **R For Business Analysis (DS4B 101-R) course** through Business Science University.



Click the links for Documentation

ggplot2 (CS)

dplyr (CS)
stringr (CS)
lubridate (CS)
forcats
Base R (CS)

Import → Tidy → Transform → Visualize → Model → Communicate

readr (CS)
readxl / writexl
odbc / DBI
rvest

tibble (CS)
tidyr (CS)

purrr (CS)
(iteration)

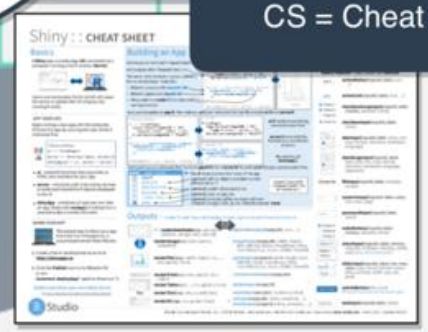
RMarkdown (CS)
Shiny (CS)

RStudio IDE (CS)

fs (file system)

recipes **broom**
rsample **yardstick**
parsnip **dials**

CS = Cheat Sheet



Important Resources

- **R For Data Science Book:** http://r4ds.had.co.nz/
- **Rmarkdown Book:** https://bookdown.org/yihui/rmarkdown/
- **Data Visualization Book**: https://rkabacoff.github.io/datavis/
- **More Cheatsheets:** https://www.rstudio.com/resources/cheatsheets/
- **tidyverse packages:** https://www.tidyverse.org/
- **Connecting to databases:** https://db.rstudio.com/
- **RMarkdown website:** https://rmarkdown.rstudio.com/
- **Shiny web applications website:** http://shiny.rstudio.com/
- **Jenny Bryan's purrr tutorial:** https://jennybryan.org/

"Data Science Education for the Enterprise"

Business Science University
university.business-science.io

Learning objectives of this module:

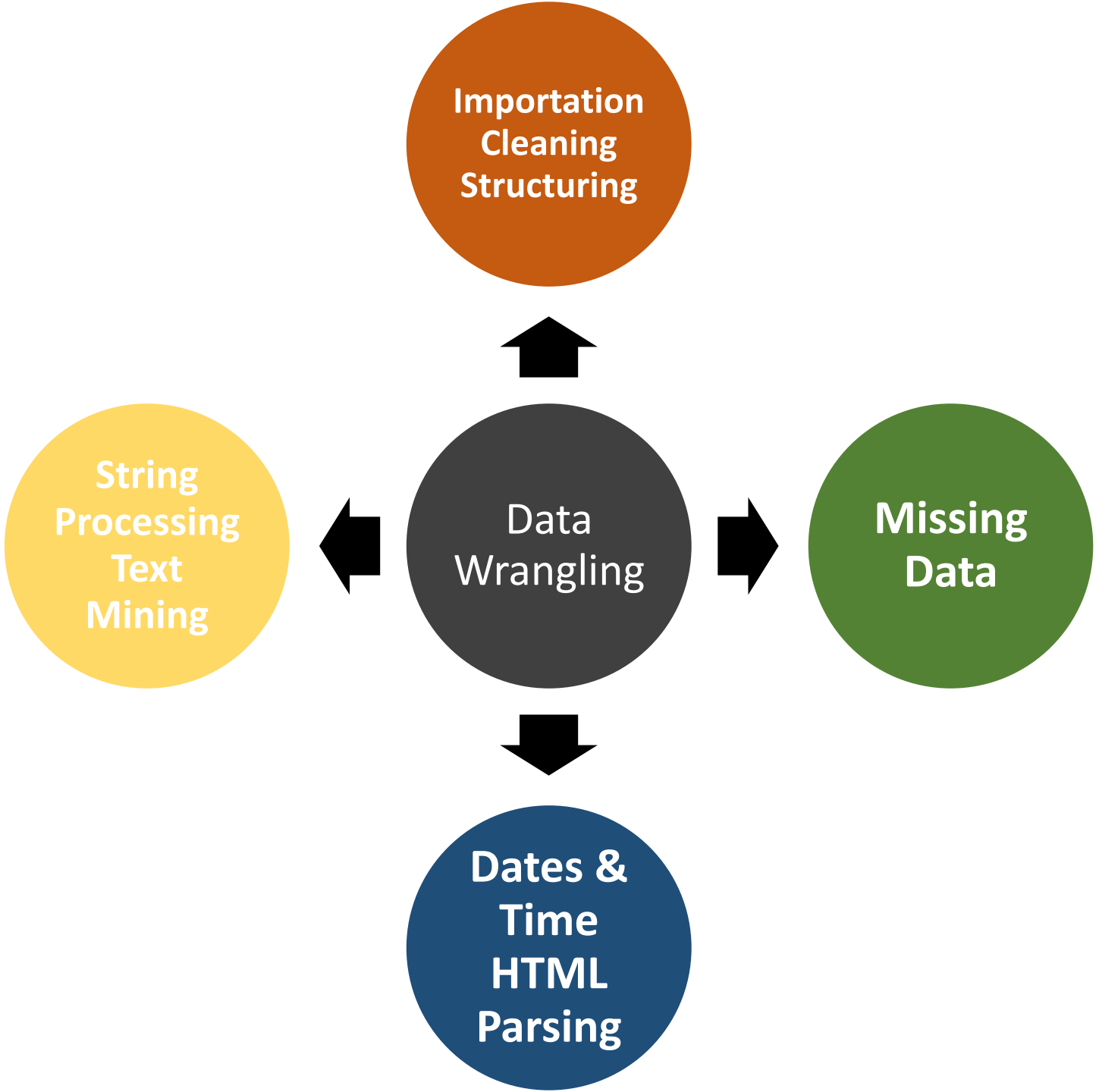
- Introduction to Data Analysis
- Learn the basic vocabulary of dplyr
- Exercise commands
- Translating questions into data manipulation statements
- Visit the tidyr package
- Learn the why and how of Exploratory Data Analysis (EDA).

Terms

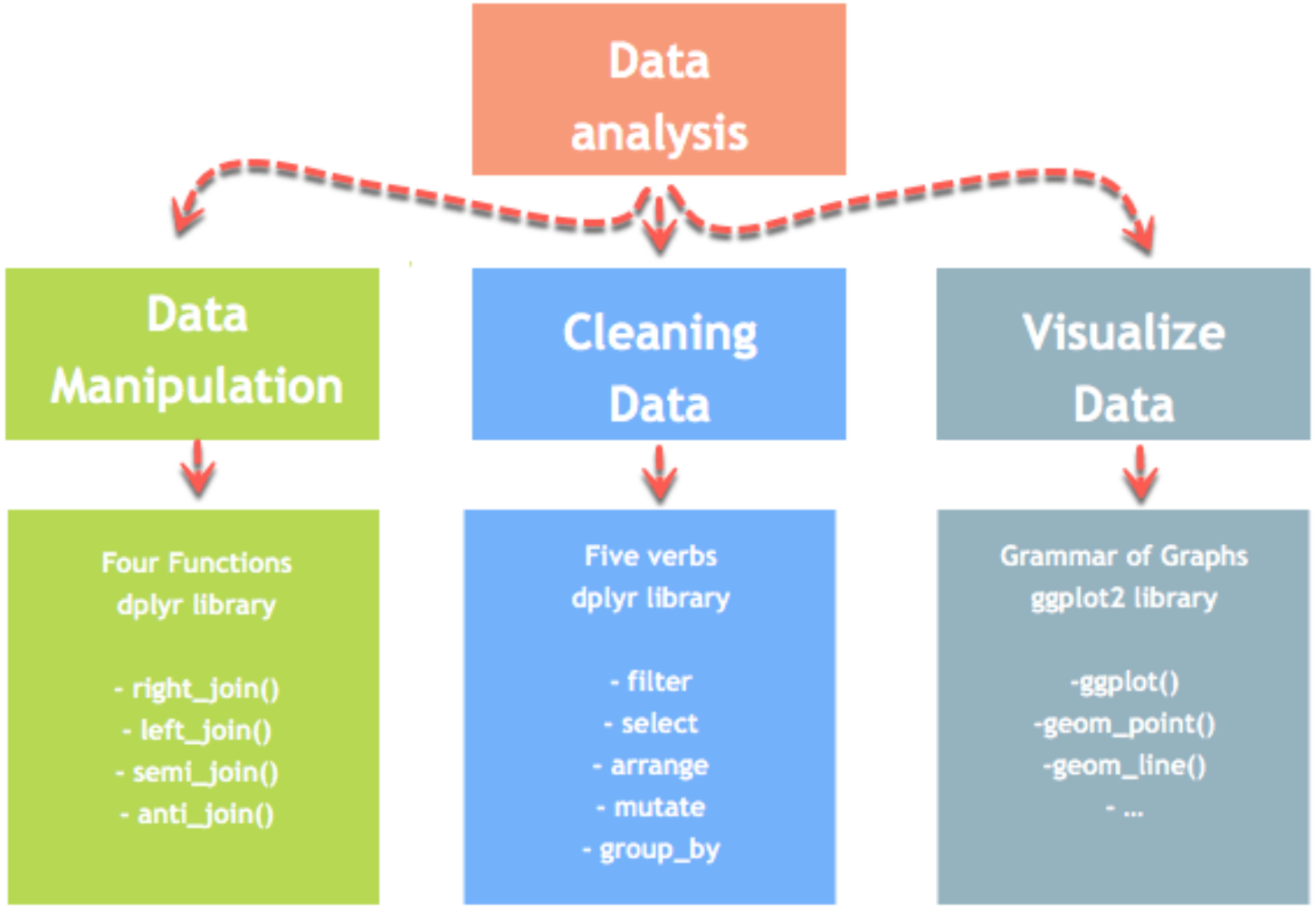
- A variable is a quantity, quality, or property that you can measure.
- A value is the state of a variable when you measure it. The value of a variable may change from measurement to measurement.
- An observation is a set of measurements made under similar conditions (you usually make all of the measurements in an observation at the same time and on the same object). An observation will contain several values, each associated with a different variable. I'll sometimes refer to an observation as a data point.
- Tabular data is a set of values, each associated with a variable and an observation. Tabular data is tidy if each value is placed in its own "cell", each variable in its own column, and each observation in its own row.

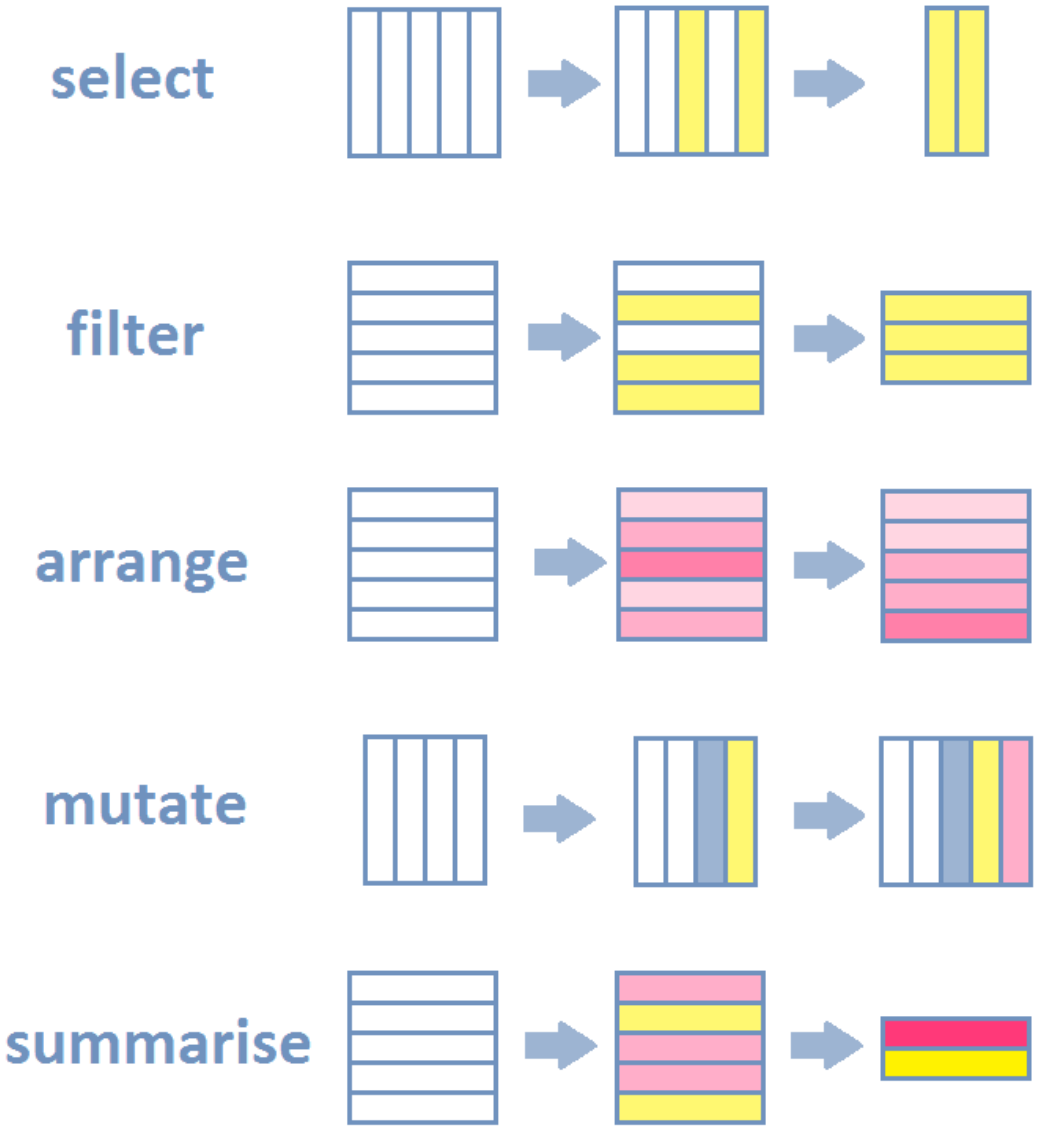
Variation

- The tendency of the values of a variable to change from measurement to measurement.
- You can see variation easily in real life; if you measure any continuous variable twice, you will get two different results. This is true even if you measure quantities that are constant, like the speed of light.
- Each of your measurements will include a small amount of error that varies from measurement to measurement.
- Categorical variables can also vary if you measure across different subjects (e.g. the eye colors of different people), or different times (e.g. the energy levels of an electron at different moments).
- Every variable has its own pattern of variation, which can reveal interesting information. The best way to understand that pattern is to visualize the distribution of the variable's values.


Importation
Cleaning
Structuring
String
Processing
Text
Mining
Data
Wrangling
Missing
Data
Dates &
Time
HTML
Parsing

Data Wrangling – Dplyr Package

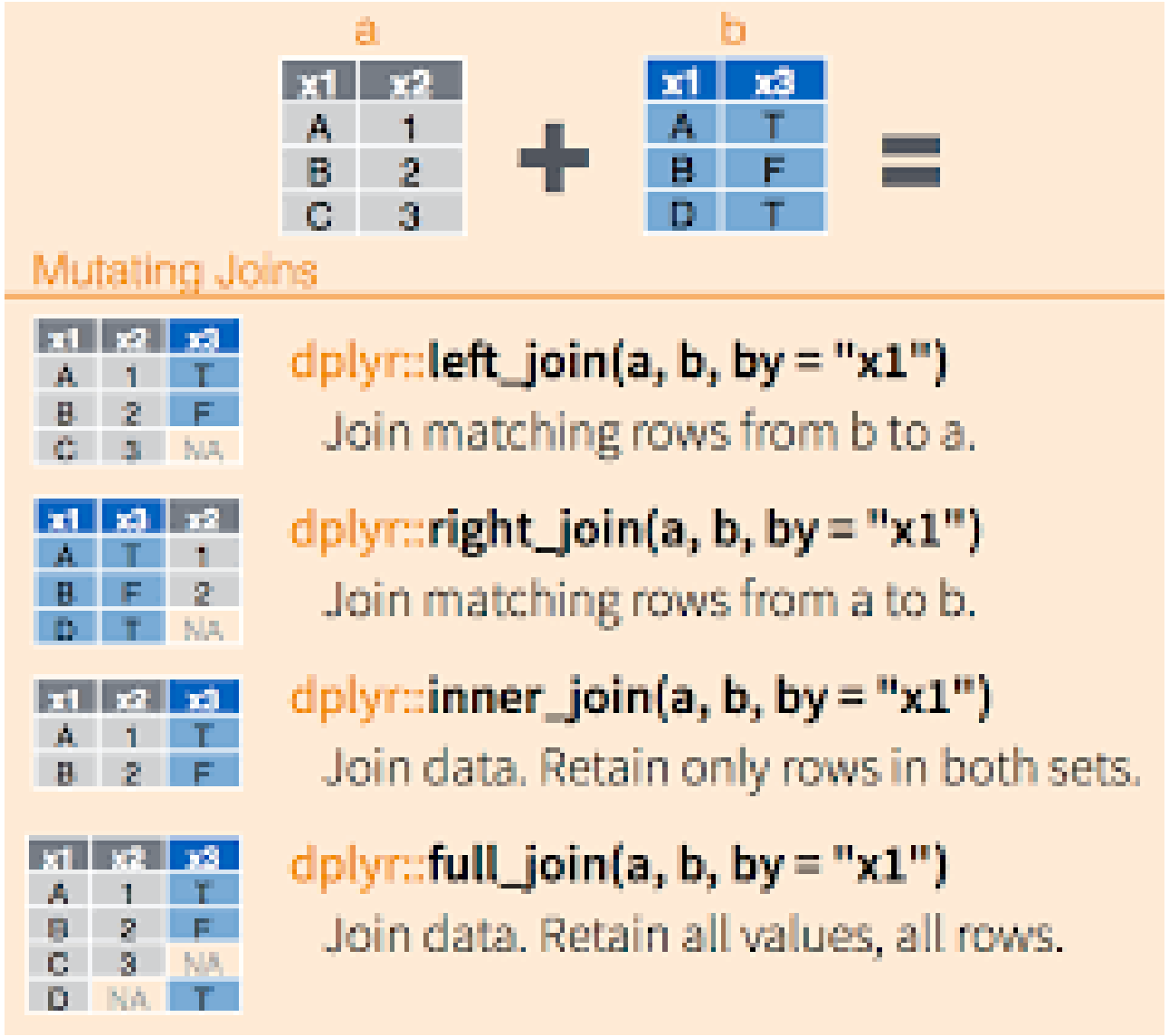




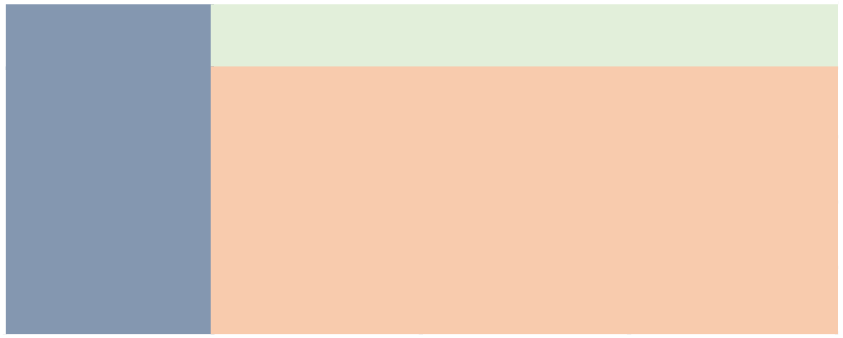
Source: http://perso.ens-lyon.fr/lise.vaudor/dplyr/

- Inspect your tibble (glimpse())
- Select specific columns (select())
- Filter out a subset of rows (filter())
- Change or add columns (mutate())
- Group observations by a grouping variable (group_by())
- Get a summary (in particular per group) (summarise())
- Join two distinct tibbles by a common column (left_join(), right_join() and full_join())

Functions that allow you to join two data frames together.

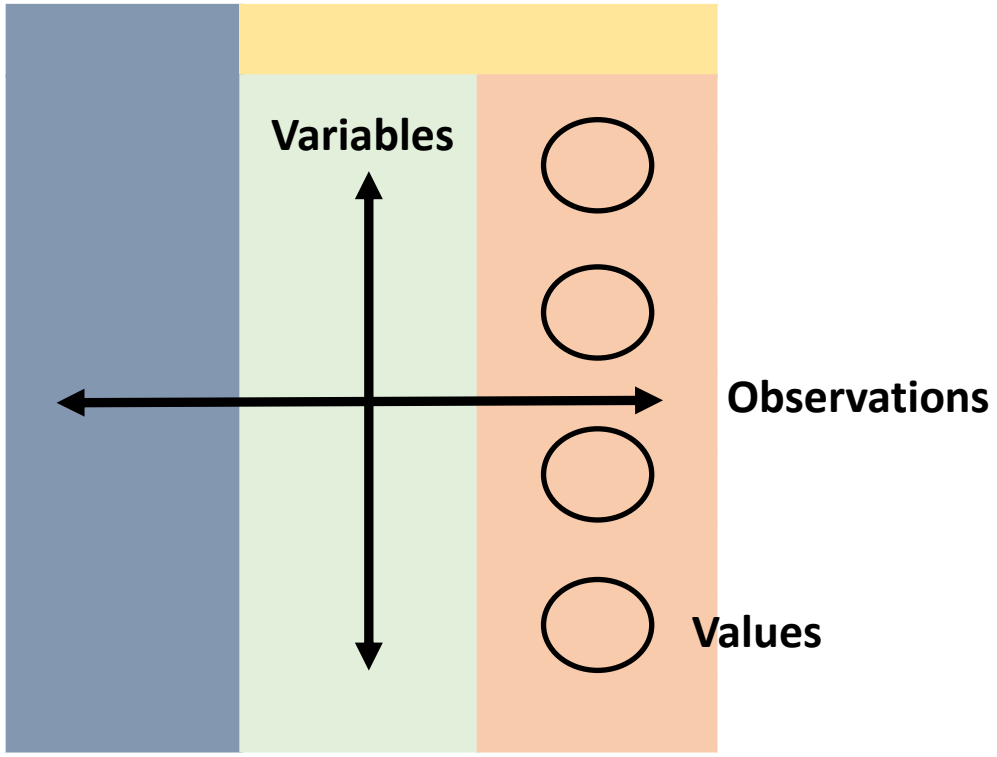


Tame Data



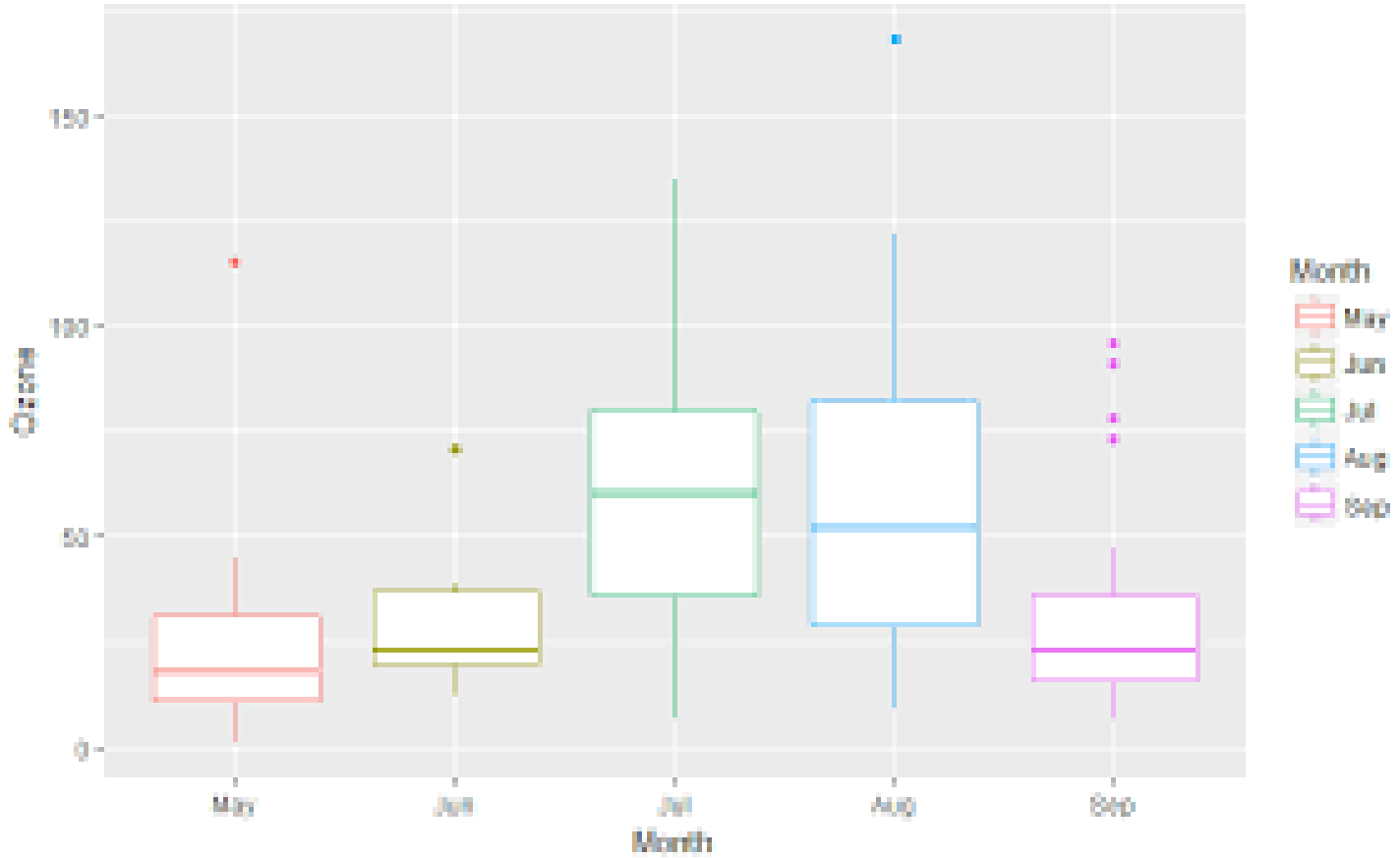
Gather()

Tidy Data



?air.quality
data(airquality)

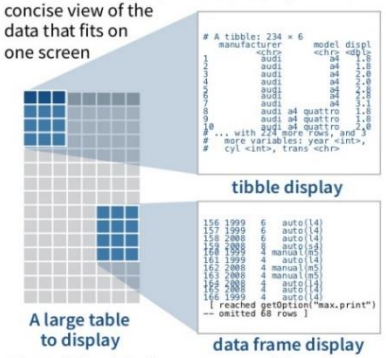
Gather() all the columns (except Month and Day) into rows.
Then spread() the resulting dataset to return the same data format as the original data.



Tibbles - an enhanced data frame

The **tibble** package provides a new S3 class for storing tabular data, the tibble. Tibbles inherit the data frame class, but improve three behaviors:

- **Subsetting** - [always returns a new tibble, [[and $ always return a vector.
- **No partial matching** - You must use full column names when subsetting
- **Display** - When you print a tibble, R provides a concise view of the data that fits on one screen



- Control the default appearance with options:

  options(tibble.print_max = n, tibble.print_min = m, tibble.width = Inf)

- View full data set with **View()** or **glimpse()**
- Revert to data frame with **as.data.frame()**

CONSTRUCT A TIBBLE IN TWO WAYS

tibble(…)
Construct by columns.
tibble(x = 1:3, y = c("a", "b", "c"))

tribble(…)
Construct by rows.

```
tribble( ~x,   ~y,
         1,   "a",
         2,   "b",
         3,   "c")
```

Both make this tibble

```
A tibble: 3 × 2
      x     y
  <int> <chr>
1     1     a
2     2     b
3     3     c
```

as_tibble(x, …) Convert data frame to tibble.

enframe(x, name = "name", value = "value")
Convert named vector to a tibble

is_tibble(x) Test whether x is a tibble.

RStudio

Tidy Data with tidyr

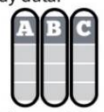
Tidy data is a way to organize tabular data. It provides a consistent data structure across packages.

A table is tidy if:

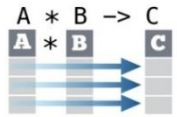
Each **variable** is in its own **column**

Each **observation**, or **case**, is in its own **row**

Tidy data:



Makes variables easy to access as vectors



Preserves cases during vectorized operations

Reshape Data - change the layout of values in a table

Use **gather()** and **spread()** to reorganize the values of a table into a new layout.

gather(data, key, value, ..., na.rm = FALSE, convert = FALSE, factor_key = FALSE)

gather() moves column names into a **key** column, gathering the column values into a single **value** column.

table4a

country	1999	2000
A	0.7K	2K
B	37K	80K
C	212K	213K

→

country	year	cases
A	1999	0.7K
B	1999	37K
C	1999	212K
A	2000	2K
B	2000	80K
C	2000	213K

key value

gather(table4a, `1999`, `2000`, key = "year", value = "cases")

spread(data, key, value, fill = NA, convert = FALSE, drop = TRUE, sep = NULL)

spread() moves the unique values of a **key** column into the column names, spreading the values of a **value** column across the new columns.

table2

country	year	type	count
A	1999	cases	0.7K
A	1999	pop	19M
A	2000	cases	2K
A	2000	pop	20M
B	1999	cases	37K
B	1999	pop	172M
B	2000	cases	80K
B	2000	pop	174M
C	1999	cases	212K
C	1999	pop	1T
C	2000	cases	213K
C	2000	pop	1T

→

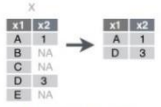
country	year	cases	pop
A	1999	0.7K	19M
A	2000	2K	20M
B	1999	37K	172M
B	2000	80K	174M
C	1999	212K	1T
C	2000	213K	1T

key value

spread(table2, type, count)

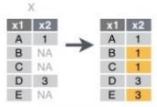
Handle Missing Values

drop_na(data, ...)
Drop rows containing NA's in … columns.



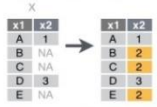
drop_na(x, x2)

fill(data, ..., .direction = c("down", "up"))
Fill in NA's in … columns with most recent non-NA values.



fill(x, x2)

replace_na(data, replace = list(), ...)
Replace NA's by column.



replace_na(x, list(x2 = 2))

Expand Tables - quickly create tables with combinations of values

complete(data, ..., fill = list())
Adds to the data missing combinations of the values of the variables listed in …
complete(mtcars, cyl, gear, carb)

expand(data, ...)
Create new tibble with all possible combinations of the values of the variables listed in …
expand(mtcars, cyl, gear, carb)

Split Cells

tidyr

Use these functions to split or combine cells into individual, isolated values.

separate(data, col, into, sep = "[^[:alnum:]]+", remove = TRUE, convert = FALSE, extra = "warn", fill = "warn", ...)

Separate each cell in a column to make several columns.

table3

country	year	rate
A	1999	0.7K/19M
A	2000	2K/20M
B	1999	37K/172M
B	2000	80K/174M
C	1999	212K/1T
C	2000	213K/1T

→

country	year	cases	pop
A	1999	0.7K	19M
A	2000	2K	20M
B	1999	37K	172
B	2000	80K	174
C	1999	212K	1T
C	2000	213K	1T

separate(table3, rate, into = c("cases", "pop"))

separate_rows(data, ..., sep = "[^[:alnum:].]+", convert = FALSE)

Separate each cell in a column to make several rows. Also **separate_rows_()**.

table3

country	year	rate
A	1999	0.7K/19M
A	2000	2K/20M
B	1999	37K/172M
B	2000	80K/174M
C	1999	212K/1T
C	2000	213K/1T

→

country	year	rate
A	1999	0.7K
A	1999	19M
A	2000	2K
A	2000	20M
B	1999	37K
B	1999	172M
B	2000	80K
B	2000	174M
C	1999	212K
C	1999	1T
C	2000	213K
C	2000	1T

separate_rows(table3, rate)

unite(data, col, ..., sep = "_", remove = TRUE)

Collapse cells across several columns to make a single column.

table5

country	century	year
Afghan	19	99
Afghan	20	0
Brazil	19	99
Brazil	20	0
China	19	99
China	20	0

→

country	year
Afghan	1999
Afghan	2000
Brazil	1999
Brazil	2000
China	1999
China	2000

unite(table5, century, year, col = "year", sep = "")

http://tidyr.tidyverse.org/ vignette("tidy-data")

tidyr functions fall into five main categories:

- “Pivotting” : converts between long and wide forms. tidyr 1.0.0 introduces pivot_longer() and pivot_wider(), replacing the older spread() and gather() functions. See vignette("pivot") for more details.

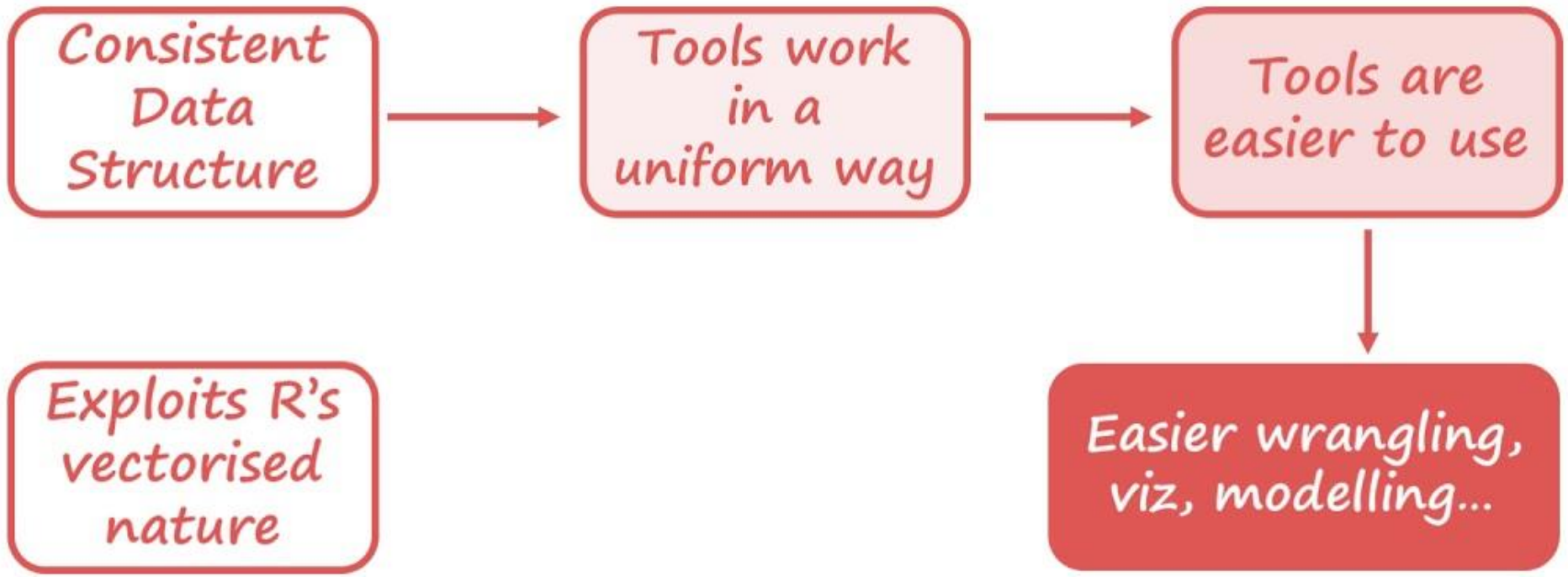
- “Rectangling” : turns deeply nested lists (as from JSON) into tidy tibbles. See unnest_longer(), unnest_wider(), hoist(), and vignette("rectangle") for more details.

- Nesting converts grouped data to a form where each group becomes a single row containing a nested data frame, and unnesting does the opposite. See nest(), unnest(), and vignette("nest") for more details.

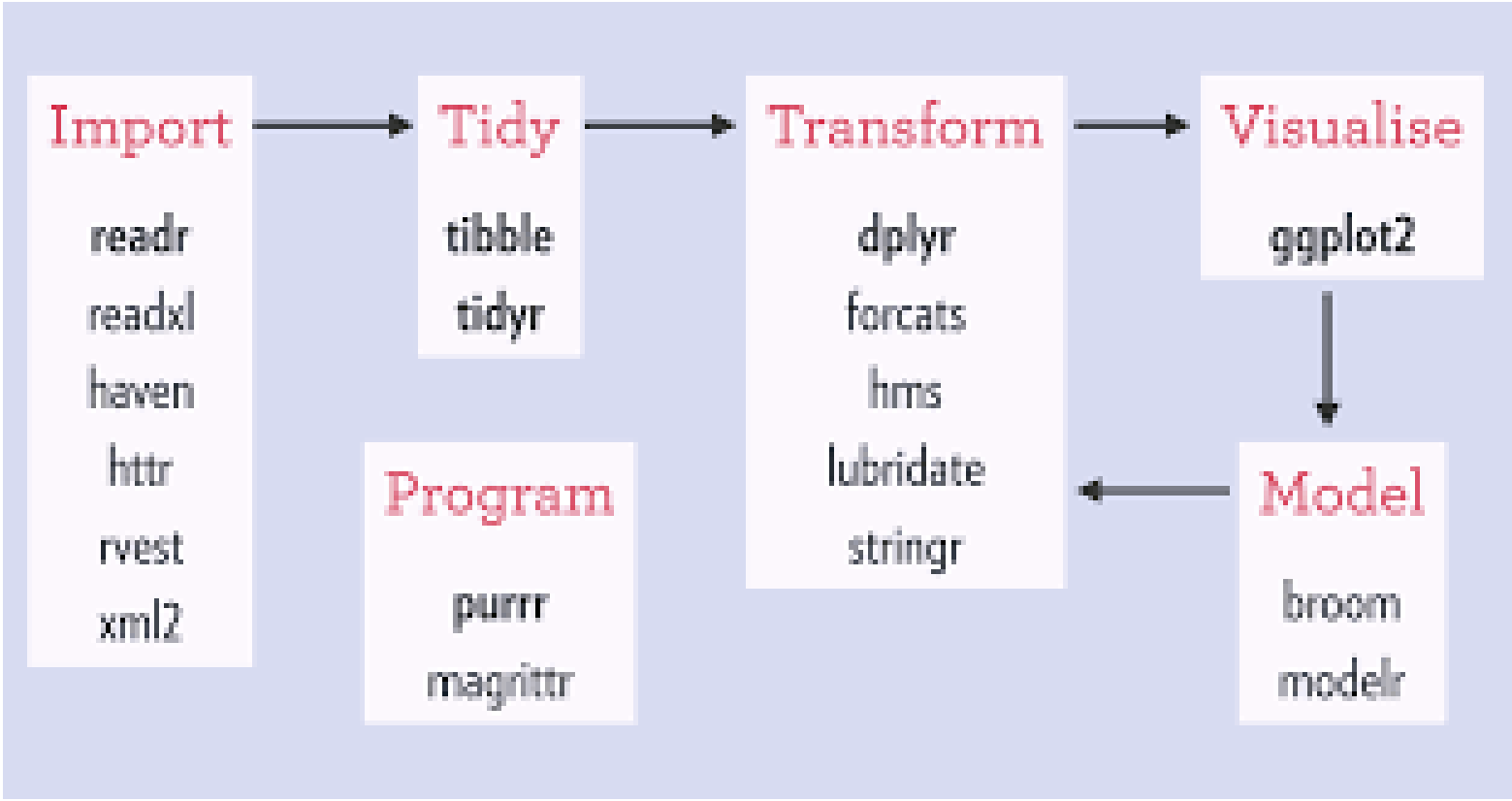
- Splitting and combining character columns. Use separate() and extract() to pull a single character column into multiple columns; use unite() to combine multiple columns into a single character column.

- Make implicit missing values explicit with complete(); make explicit missing values implicit with drop_na(); replace missing values with next/previous value with fill(), or a known value with replace_na().

Why tidy data?



https://www.tidyverse.org/



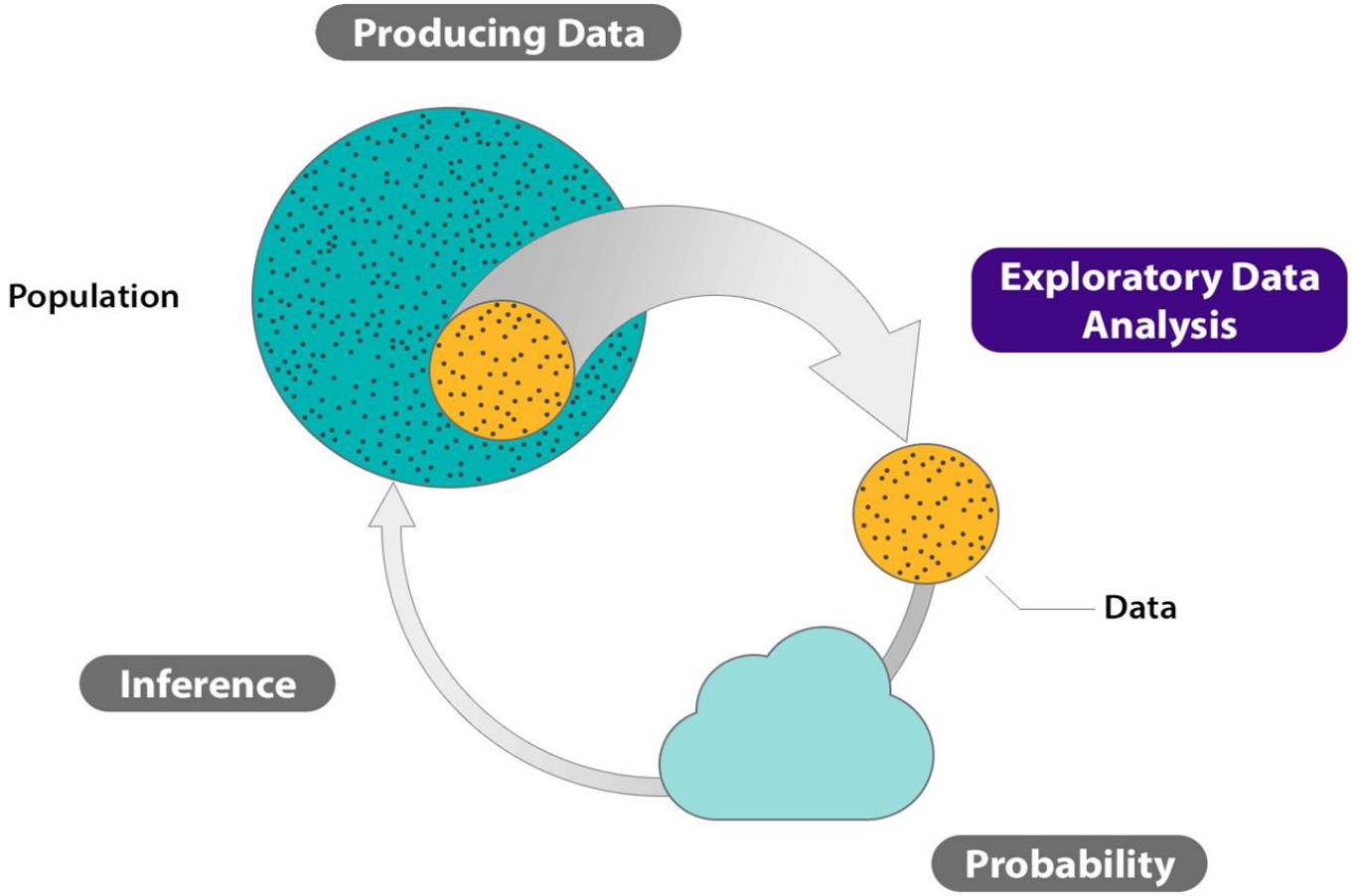

```
install.packages("tidyverse")

library("tidyverse")
```

https://github.com/rstudio/master-the-tidyverse/archive/master.zip

Tidy Data

See the paper Tidy Data by Hadley Wickham in Journal of Statistical Software (2014)

https://github.com/rstudio/master-the-tidyverse/archive/master.zip



Exploratory Data Analysis: an approach to analyzing data sets to summarize their main characteristics, often with visual methods. - Wikipedia


1
Collect the data and gain domain knowledge
2
Confirm data types and their probabilities
3
Measures of Central Tendency
a. Mean
b. Median
C. Mode
4
Measures of Dispersion
a. Variance
b. Std. Dev.
c. Range
5
Skewness right & left
Kurtosis
Thinner peck
Wider peck
6
Graphical Representation
a. Histogram
b. Boxplot
c. Dotplot
d. Stem and Leaf
EXPLORATORY DATA ANALYSIS

“Get to Know" the dataset

- Doing so upfront will make the rest of the project much smoother, in 3 main ways:
  1. You’ll gain valuable hints for Data Cleaning.
  2. You’ll think of ideas for Feature Engineering.
  3. You’ll get a "feel" for the dataset, which will help you communicate results and deliver greater impact.

- EDA should be **quick, efficient, and decisive**... not long and drawn out!
- You see, there are infinite possible plots, charts, and tables, but you only need a **handful** to "get to know" the data well enough to work with it.

What is EDA?

An approach for data analysis that employs a variety of techniques

1. Maximize insight into a data set
2. Uncover underlying structure
3. Extract important variables
4. Detect outliers and anomalies
5. Test underlying assumptions
6. Develop parsimonious models and
7. Determine optimal factor settings

EDA is a data approach.

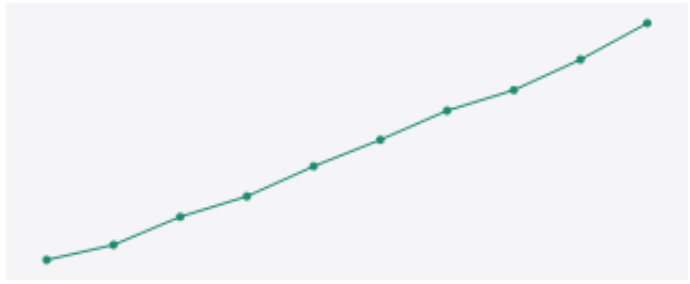
The EDA sequence is:

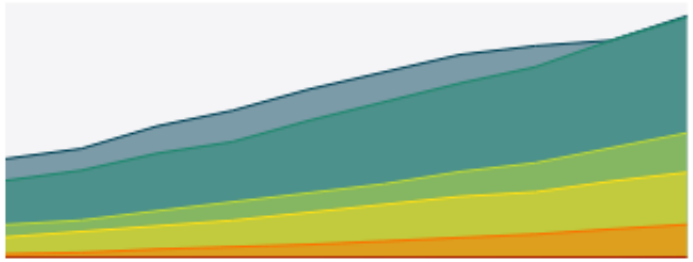
Problem => Data => Analysis=> Model=> Conclusions

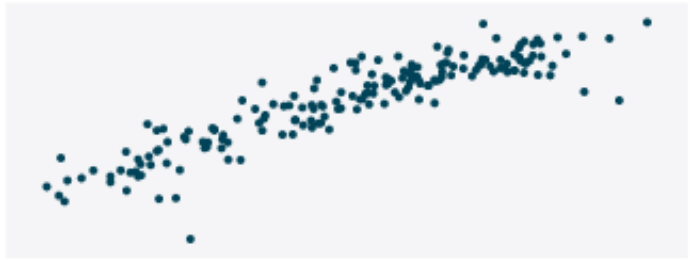
As opposed for a classical approach:

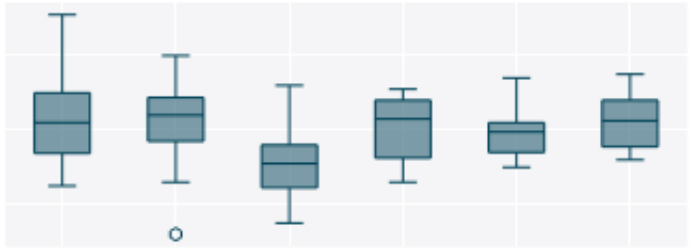
Problem => Data => Model=> Analysis=> Conclusions

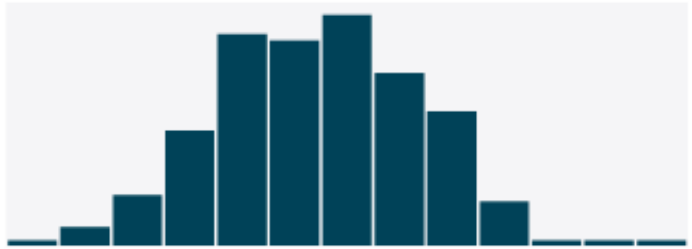
EDA Techniques are generally graphical











EDA is majorly performed using the following methods:

Univariate visualization – provides summary statistics for each field in the raw data set

Bivariate visualization – is performed to find the relationship between each variable in the dataset and the target variable of interest

Multivariate visualization – is performed to understand interactions between different fields in the dataset

Dimensionality reduction – helps to understand the fields in the data that account for the most variance between observations and allow for the processing of a reduced volume of data.

Here is a list of all graph types that are illustrated:

- Barplot
- Boxplot
- Density Plot
- Heatmap
- Histogram
- Line Plot
- Pairs Plot
- Polygon Plot
- QQplot
- Scatterplot
- Venn Diagram

Barplot	A barplot (or barchart; bargraph) illustrates the association between a numeric and a categorical variable. The barplot represents each category as a bar and reflects the corresponding numeric value with the bar's size.	barplot(x)
Boxplot	Displays the distribution of a numerical variable based on five summary statistics: minimum non-outlier; first quartile; median; third quartile; and maximum non-outlier. Furthermore, boxplots show the positioning of outliers and whether the data is skewed.	boxplot(x)
Density Plot	A density plot (or kernel density plot; density trace graph) shows the distribution of a numerical variable over a continuous interval. Peaks of a density plot visualize where the values of numerical variables are concentrated.	plot(density(x))
Heatmap	A heatmap (or shading matrix) visualizes individual values of a matrix with colors. More common values are typically indicated by brighter reddish colors and less common values are typically indicated by darker colors.	heatmap(cbind(x, y))
Histogram	A histogram groups continuous data into ranges and plots this data as bars. The height of each bar shows the number of observations within each range.	hist(x)

Line Plot	A line plot, visualizes values along a sequence (e.g., over time). Line plots consist of an x-axis and a y-axis. The x-axis usually displays the sequence and the y-axis the values corresponding to each point of the sequence.	plot(1:length(y), y, type = "l")
Pairs Plot	A pairs plot is a plot matrix, consisting of scatterplots for each variable-combination of a data frame.	pairs(data.frame(x, y))
Polygon Plot	A polygon plot displays a plane geometric figure (i.e., a polygon) within the plot	plot(1,1 col = "white", xlab="X", ylab = "Y") polygon(x= c(0.7, 1.3, 1.3, 0.8), y=c(0.6, 1.0, 1.4, 1.3), col = "#353436")
QQplot	Quantile-Quantile plot; Quantile-Quantile diagram) determines whether two data sources come from a common distribution. QQ plots draw the quantiles of the two numerical data sources against each other. If both data sources come from the same distribution, the points fall on a 45-degree angle.	qqplot(x,y)
Scatterplot	A scatterplot (or scatter plot; scatter graph; scatter chart; scattergram; scatter diagram) displays two numerical variables with points, whereby each point represents the value of one variable on the x-axis and the value of the other variable on the y-axis.	plot(x,y)
Venn Diagram	A venn diagram (or primary diagram; set diagram; logic diagram) illustrates all possible logical relations between certain data characteristics. Each characteristic is represented as a circle, whereby overlapping parts of the circles illustrate elements that have both characteristics at the same time.	Install.packages("VennDiagram") library("VennDiagram") plot.new() draw.single.venn(area = 10)

EDA of Categorical Data

Useful Packages

```
1
2  ## Needed libs for EDA
3  ## Some may need to install these packages if this is their
4  ## first time using R
5
6  library(dplyr)
7  library(ggplot2)
8  library(gapminder)
9  library(tidyr)
10 library(readr)
11 library(openintro)
12 options(scipen=999,digits=3)
13
```

Example: Coronvirus

- https://www.kaggle.com/xordux/india-corona-severity-zones
- The zones are:
  1. Green Zone: Least impacted zone, A district will be considered under green zone if there has been no confirmed cases of COVID-19 so far or there is no reported case since last 21 days in the district.
  2. Orange Zone: Districts that do not have enough confirmed cases to meet the ‘red zone’, but are being seen as potential hotspots, are part of the ‘orange zone’. A Red Zone can be categorised as a Orange Zone if no new confirmed case is reported there for 14 consecutive days.
  3. Red Zone: Districts reporting a large number of cases or high growth rates. Inclusion criteria for Red Zone:
     1. Highest case-load districts contributing to over 80 percent of cases in India, or
     2. Highest case-load districts contributing to more than 80 percent of cases for each state in the country, or
     3. Districts with doubling rate at less than four days (calculated every Monday for last seven days, to be determined by the state government).

Upload Data

```
## Download data and see what the set is composed of
## Make sure you download data in the working directory
## TO CHANGE: Toolbar: Session > Set Working Directory > Choose Directory > (select
df <- read.csv("covid_zones.csv")
glimpse(df)
```

```
> glimpse(df)
Rows: 733
Columns: 4
$ No       <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,...
$ District <chr> " South Andamans ", " Nicobars ", " North And Middle Andaman ", " ...
$ State    <chr> " Andaman And Nicobar Islands ", " Andaman And Nicobar Islands ", ...
$ Zone     <chr> " Red Zone", " Green Zone", " Green Zone", " Red Zone", " Red Zone...
```

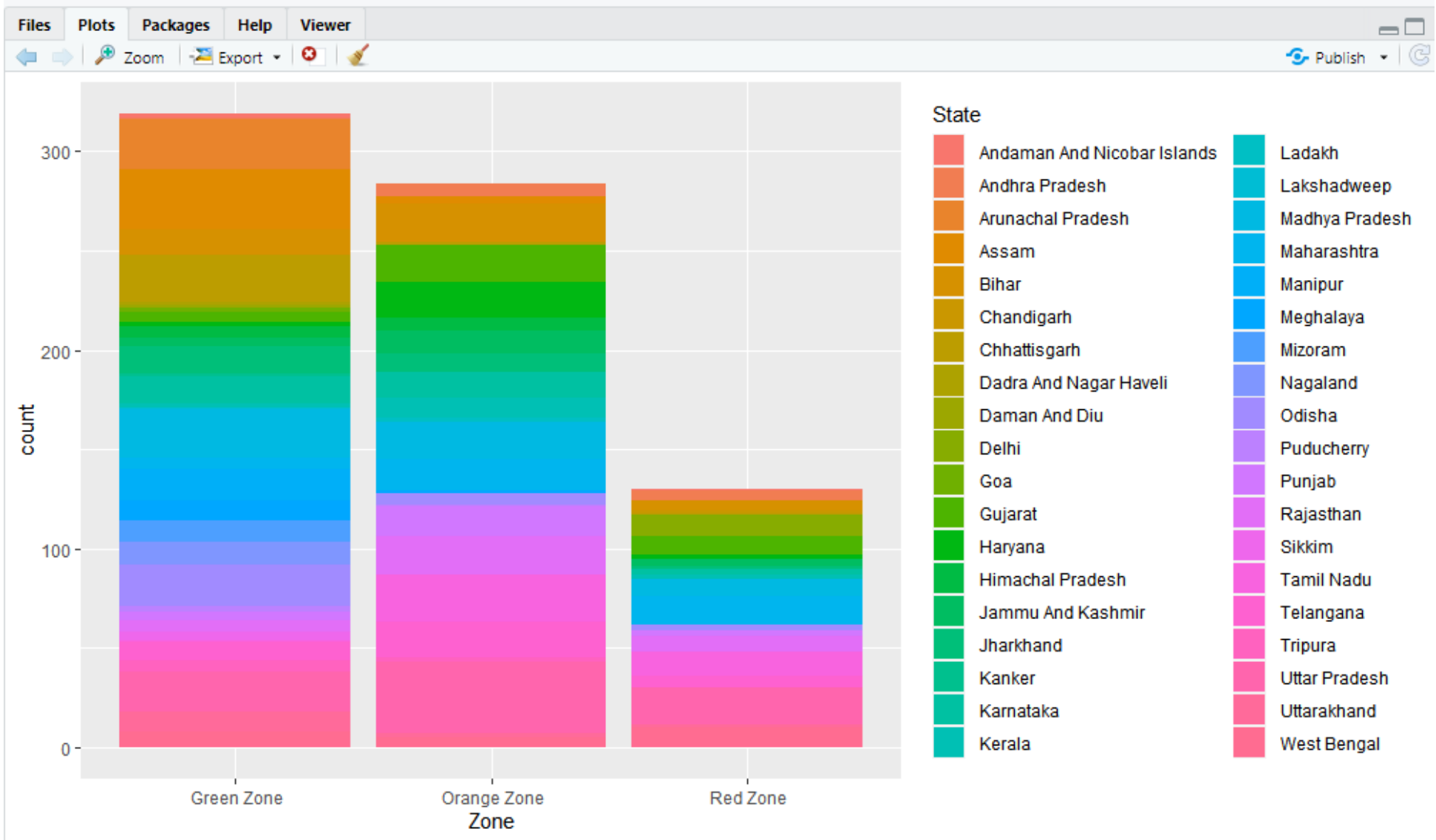
Contingency Table

```
## Contingency table: To get a frequency distribution between 2 factors variables

table(df$Zone,df$State)

## A ggplot always needs three basic inputs - 1) dataset 2) variables on axes
## 3) layer to be used. For 2 categorical variables, a stack bar chart is good.
## In this case, one categorical variable goes on x axis, in each bar,
## the other categorical variable is filled using the color.

ggplot(df,aes(x=Zone,fill=State)) + geom_bar()
```

Output for Contingency Table

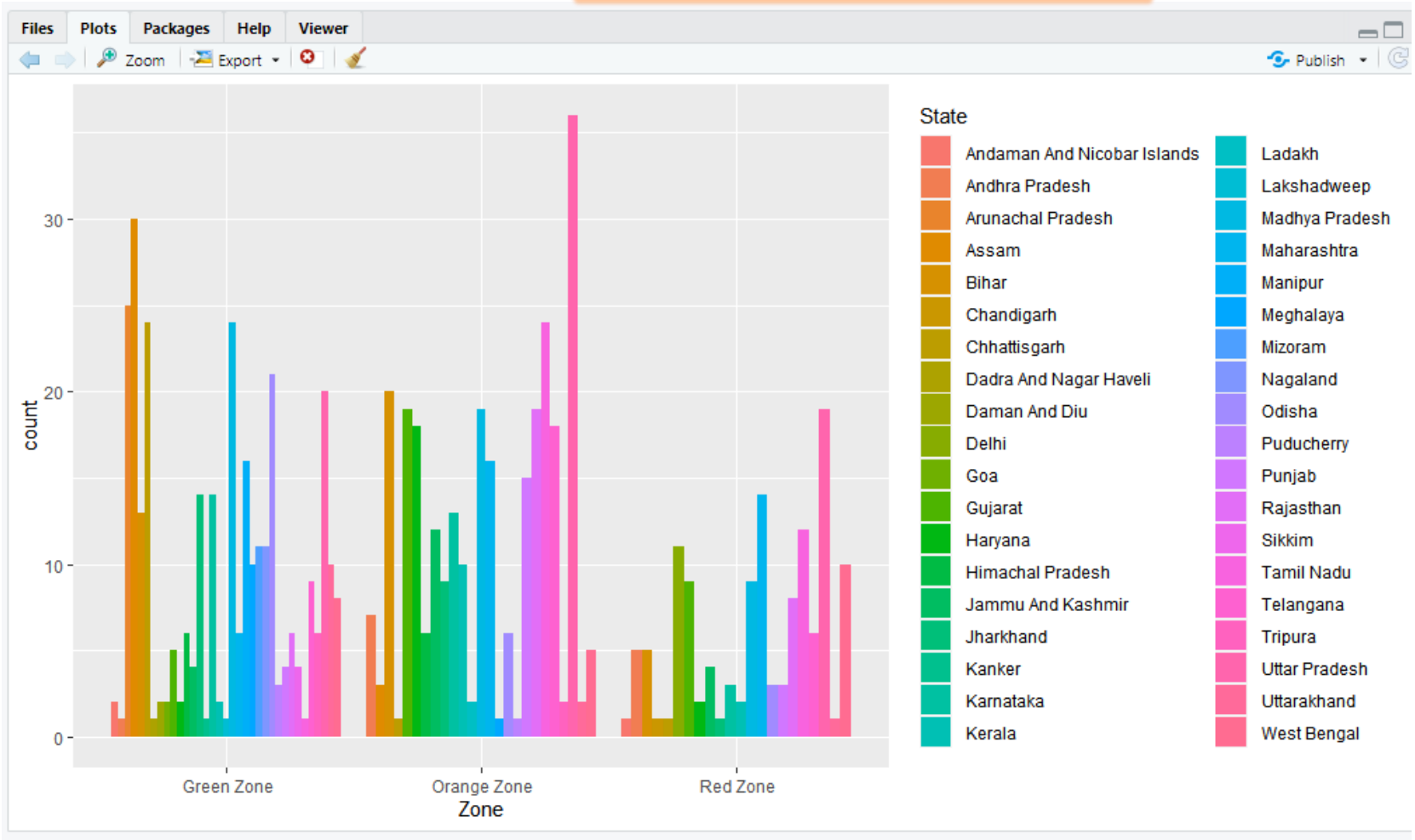
```
> table(df$Zone,df$State)

            Andaman And Nicobar Islands   Andhra Pradesh   Arunachal Pradesh
  Green Zone                           2                1                  25
  Orange Zone                          0                7                   0
  Red Zone                             1                5                   0

            Assam   Bihar   Chandigarh   Chhattisgarh   Dadra And Nagar Haveli
  Green Zone    30      13            0             24                        1
  Orange Zone    3      20            0              1                        0
  Red Zone       0       5            1              1                        0

            Daman And Diu   Delhi   Goa   Gujarat   Haryana   Himachal Pradesh
  Green Zone             2       0     2         5         2                  6
  Orange Zone            0       0     0        19        18                  6
  Red Zone               0      11     0         9         2                  0

            Jammu And Kashmir   Jharkhand   Kanker   Karnataka   Kerala   Ladakh
  Green Zone                 4          14        1          14        2        0
  Orange Zone               12           9        0          13       10        2
  Red Zone                   4           1        0           3        2        0

            Lakshadweep   Madhya Pradesh   Maharashtra   Manipur   Meghalaya
  Green Zone           1               24             6        16          10
  Orange Zone          0               19            16         0           1
  Red Zone             0                9            14         0           0

            Mizoram   Nagaland   Odisha   Puducherry   Punjab   Rajasthan   Sikkim
  Green Zone      11         11       21            3        4           6        4
  Orange Zone      0          0        6            1       15          19        0
  Red Zone         0          0        3            0        3           8        0

            Tamil Nadu   Telangana   Tripura   Uttar Pradesh   Uttarakhand
  Green Zone           1           9         6              20            10
  Orange Zone         24          18         2              36             2
  Red Zone            12           6         0              19             1

            West Bengal
  Green Zone           8
  Orange Zone          5
  Red Zone            10
>
```

Output for ggplot()



Side-by-Side?

```
31  ## When you dont want the information to be stacked, but want them side-by-side,
32  ## use the **position="dodge"** arguement in the geom_bar().
33
34  ggplot(df,aes(x=Zone,fill=State)) + geom_bar(position="dodge")
35
```



Proportions

```
36  ## Sometimes the count isn't what is important. We want proportions, so the
37  ## arguement **prop.table()** will change the contingency table to where the
38  ## values are percentages
39
40  tab_cnt <- table(df$Zone,df$State)
41  prop.table(tab_cnt)
```

Output

```
> prop.table(tab_cnt)

             Andaman And Nicobar Islands   Andhra Pradesh   Arunachal Pradesh
Green Zone                       0.00273          0.00136             0.03411
Orange Zone                      0.00000          0.00955             0.00000
Red Zone                         0.00136          0.00682             0.00000

             Assam   Bihar   Chandigarh   Chhattisgarh   Dadra And Nagar Haveli
Green Zone  0.04093 0.01774      0.00000        0.03274                  0.00136
Orange Zone 0.00409 0.02729      0.00000        0.00136                  0.00000
Red Zone    0.00000 0.00682      0.00136        0.00136                  0.00000

             Daman And Diu   Delhi     Goa   Gujarat   Haryana   Himachal Pradesh
Green Zone         0.00273 0.00000 0.00273   0.00682   0.00273            0.00819
Orange Zone        0.00000 0.00000 0.00000   0.02592   0.02456            0.00819
Red Zone           0.00000 0.01501 0.00000   0.01228   0.00273            0.00000

             Jammu And Kashmir   Jharkhand   Kanker   Karnataka   Kerala   Ladakh
Green Zone             0.00546     0.01910  0.00136     0.01910  0.00273  0.00000
Orange Zone            0.01637     0.01228  0.00000     0.01774  0.01364  0.00273
Red Zone               0.00546     0.00136  0.00000     0.00409  0.00273  0.00000

             Lakshadweep   Madhya Pradesh   Maharashtra   Manipur   Meghalaya
Green Zone       0.00136          0.03274       0.00819   0.02183     0.01364
Orange Zone      0.00000          0.02592       0.02183   0.00000     0.00136
Red Zone         0.00000          0.01228       0.01910   0.00000     0.00000

             Mizoram   Nagaland   Odisha   Puducherry   Punjab   Rajasthan   Sikkim
Green Zone   0.01501    0.01501  0.02865      0.00409  0.00546     0.00819  0.00546
Orange Zone  0.00000    0.00000  0.00819      0.00136  0.02046     0.02592  0.00000
Red Zone     0.00000    0.00000  0.00409      0.00000  0.00409     0.01091  0.00000

             Tamil Nadu   Telangana   Tripura   Uttar Pradesh   Uttarakhand
Green Zone       0.00136     0.01228   0.00819         0.02729       0.01364
Orange Zone      0.03274     0.02456   0.00273         0.04911       0.00273
Red Zone         0.01637     0.00819   0.00000         0.02592       0.00136

             West Bengal
Green Zone       0.01091
Orange Zone      0.00682
Red Zone         0.01364
>
```

Proportions (con’t)

```
43  ## This forces the rows to be to add to give 1
44  prop.table(tab_cnt,1)
45
```

```
> ## This forces the rows to be to add to give 1
> prop.table(tab_cnt,1)

            Andaman And Nicobar Islands   Andhra Pradesh   Arunachal Pradesh
Green Zone                      0.00627          0.00313             0.07837
Orange Zone                     0.00000          0.02465             0.00000
Red Zone                        0.00769          0.03846             0.00000

            Assam   Bihar   Chandigarh   Chhattisgarh   Dadra And Nagar Haveli
Green Zone  0.09404 0.04075      0.00000        0.07524                  0.00313
Orange Zone 0.01056 0.07042      0.00000        0.00352                  0.00000
Red Zone    0.00000 0.03846      0.00769        0.00769                  0.00000

            Daman And Diu   Delhi     Goa   Gujarat   Haryana   Himachal Pradesh
Green Zone        0.00627 0.00000 0.00627   0.01567   0.00627             0.01881
Orange Zone       0.00000 0.00000 0.00000   0.06690   0.06338             0.02113
Red Zone          0.00000 0.08462 0.00000   0.06923   0.01538             0.00000

            Jammu And Kashmir   Jharkhand   Kanker   Karnataka   Kerala   Ladakh
Green Zone              0.01254     0.04389  0.00313     0.04389  0.00627  0.00000
Orange Zone             0.04225     0.03169  0.00000     0.04577  0.03521  0.00704
Red Zone                0.03077     0.00769  0.00000     0.02308  0.01538  0.00000

            Lakshadweep   Madhya Pradesh   Maharashtra   Manipur   Meghalaya
Green Zone      0.00313          0.07524       0.01881   0.05016     0.03135
Orange Zone     0.00000          0.06690       0.05634   0.00000     0.00352
Red Zone        0.00000          0.06923       0.10769   0.00000     0.00000

            Mizoram   Nagaland   Odisha   Puducherry   Punjab   Rajasthan   Sikkim
Green Zone   0.03448    0.03448  0.06583      0.00940  0.01254     0.01881  0.01254
Orange Zone  0.00000    0.00000  0.02113      0.00352  0.05282     0.06690  0.00000
Red Zone     0.00000    0.00000  0.02308      0.00000  0.02308     0.06154  0.00000

            Tamil Nadu   Telangana   Tripura   Uttar Pradesh   Uttarakhand
Green Zone      0.00313     0.02821   0.01881         0.06270       0.03135
Orange Zone     0.08451     0.06338   0.00704         0.12676       0.00704
Red Zone        0.09231     0.04615   0.00000         0.14615       0.00769

            West Bengal
Green Zone      0.02508
Orange Zone     0.01761
Red Zone        0.07692
>
```

Proportions (con't)

```
46  ## This forces the columns to be to add to give 1
47  prop.table(tab_cnt,2)
```

```
> ## This forces the columns to be to add to give 1
> prop.table(tab_cnt,2)
```

	Andaman And Nicobar Islands	Andhra Pradesh	Arunachal Pradesh
Green Zone	0.6667	0.0769	1.0000
Orange Zone	0.0000	0.5385	0.0000
Red Zone	0.3333	0.3846	0.0000

	Assam	Bihar	Chandigarh	Chhattisgarh	Dadra And Nagar Haveli
Green Zone	0.9091	0.3421	0.0000	0.9231	1.0000
Orange Zone	0.0909	0.5263	0.0000	0.0385	0.0000
Red Zone	0.0000	0.1316	1.0000	0.0385	0.0000

	Daman And Diu	Delhi	Goa	Gujarat	Haryana	Himachal Pradesh
Green Zone	1.0000	0.0000	1.0000	0.1515	0.0909	0.5000
Orange Zone	0.0000	0.0000	0.0000	0.5758	0.8182	0.5000
Red Zone	0.0000	1.0000	0.0000	0.2727	0.0909	0.0000

	Jammu And Kashmir	Jharkhand	Kanker	Karnataka	Kerala	Ladakh
Green Zone	0.2000	0.5833	1.0000	0.4667	0.1429	0.0000
Orange Zone	0.6000	0.3750	0.0000	0.4333	0.7143	1.0000
Red Zone	0.2000	0.0417	0.0000	0.1000	0.1429	0.0000

	Lakshadweep	Madhya Pradesh	Maharashtra	Manipur	Meghalaya
Green Zone	1.0000	0.4615	0.1667	1.0000	0.9091
Orange Zone	0.0000	0.3654	0.4444	0.0000	0.0909
Red Zone	0.0000	0.1731	0.3889	0.0000	0.0000

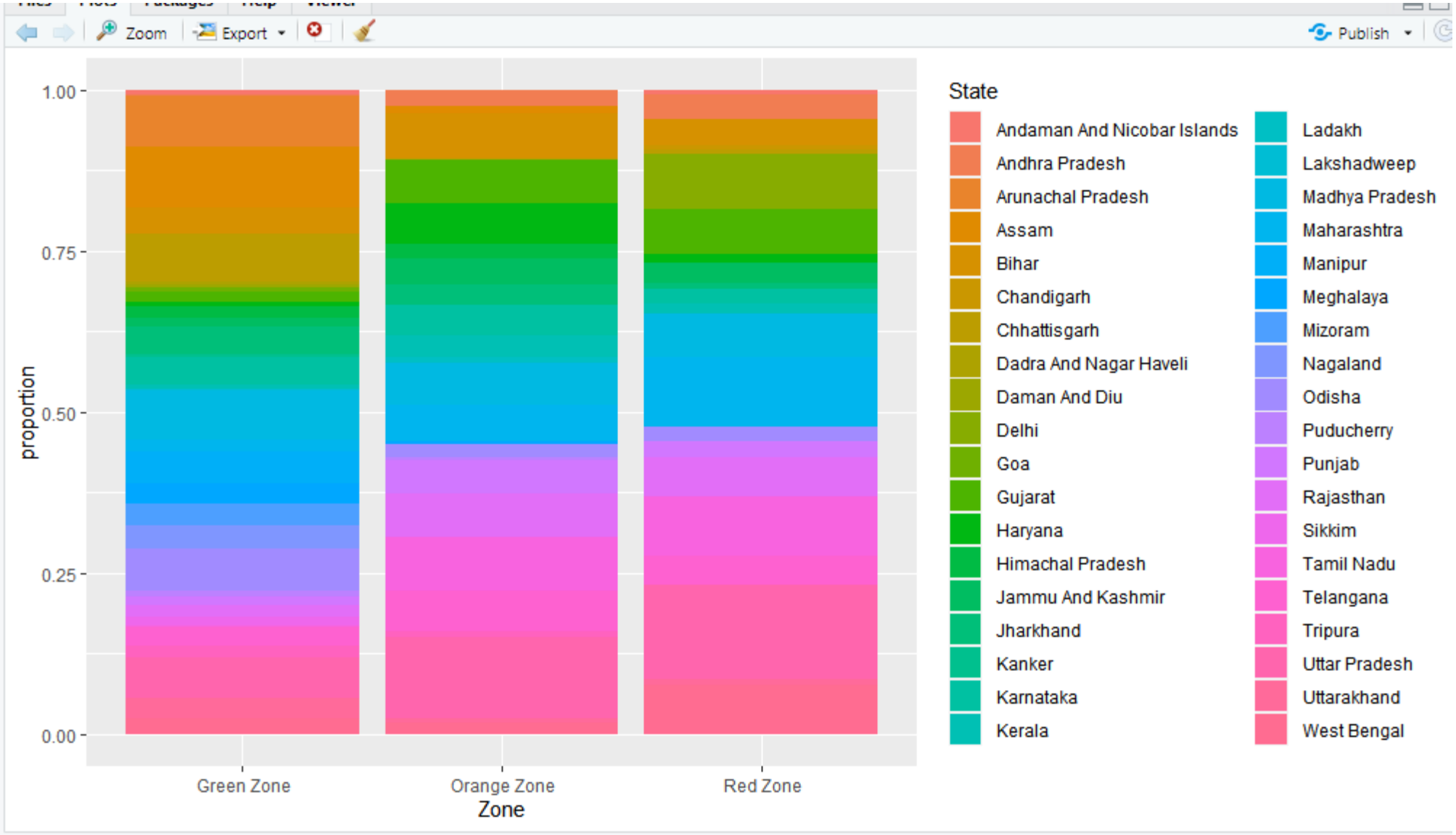
	Mizoram	Nagaland	Odisha	Puducherry	Punjab	Rajasthan	Sikkim
Green Zone	1.0000	1.0000	0.7000	0.7500	0.1818	0.1818	1.0000
Orange Zone	0.0000	0.0000	0.2000	0.2500	0.6818	0.5758	0.0000
Red Zone	0.0000	0.0000	0.1000	0.0000	0.1364	0.2424	0.0000

	Tamil Nadu	Telangana	Tripura	Uttar Pradesh	Uttarakhand
Green Zone	0.0270	0.2727	0.7500	0.2667	0.7692
Orange Zone	0.6486	0.5455	0.2500	0.4800	0.1538
Red Zone	0.3243	0.1818	0.0000	0.2533	0.0769

	West Bengal
Green Zone	0.3478
Orange Zone	0.2174
Red Zone	0.4348

100% stack chart, conditioned on Zone

```
49  ## Stacked 100% bar chart. This is called 100% stack chart, conditioned on Zone
50  ggplot(df,aes(x=Zone,fill=State)) + geom_bar(position="fill") + ylab("proportion")
```

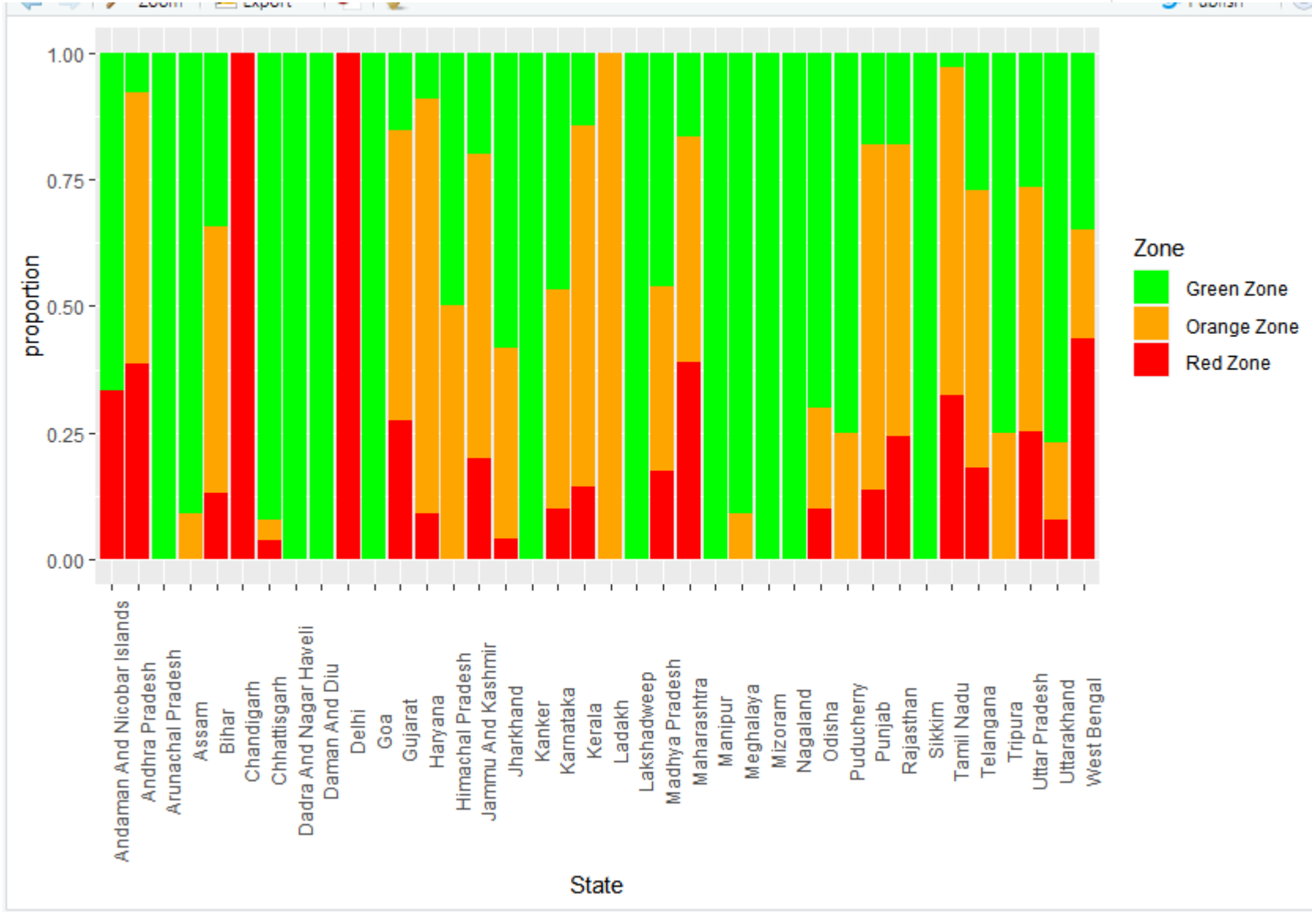


100% stacked bar chart, conditioned on State

```
52  ## 100% stacked bar chart, conditioned on State
53  ggplot(df,aes(x=State,fill=Zone)) + geom_bar(position="fill") + ylab("proportion")+ scale_fill_manual(values = c("Green", "Orange", "Red"))
54
```

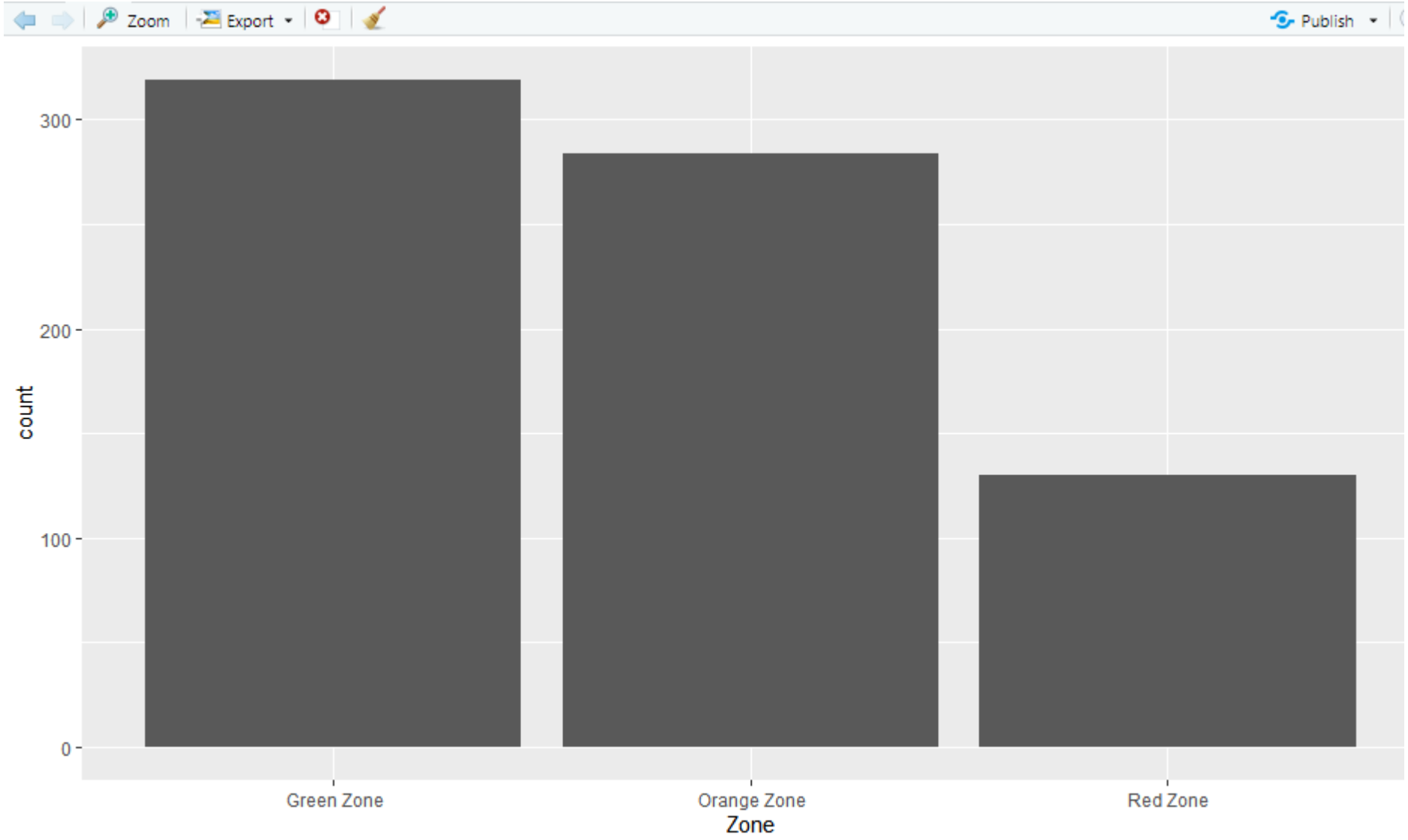
Let's Clean the labels up!

```r
## Rotates x labels by a 90 degree angle.
########## They cannot space/enter here. otherwise an error will occur..
########## It must all stay on one line.
ggplot(df,aes(x=State,fill=Zone)) + geom_bar(position="fill") + ylab("proportion")+
  scale_fill_manual(values = c("Green", "Orange", "Red")) + theme(axis.text.x = element_text(angle=90))
```



Marginal Distribution

```
61  ## Contingency table of marginal distributions.
62  table(df$Zone)
63  ggplot(df,aes(x=Zone)) + geom_bar()
64
```



Distribution against one variable

```
65  ## If concerned with analyzing one variable's distribution against only one value
66  ## of another variable.
67  ggplot(df,aes(x=Zone)) + geom_bar() + facet_wrap(~State)+ theme(axis.text.x = element_text(angle=90))
68
```

EDA of Numeric Data

Example: Air Quality

- https://stat.ethz.ch/R-manual/R-devel/library/datasets/html/airquality.html
- Daily air quality measurements in New York, May to September 1973.
- Daily readings of the following air quality values for May 1, 1973 (a Tuesday) to September 30, 1973.
  - Ozone: Mean ozone in parts per billion from 1300 to 1500 hours at Roosevelt Island
  - Solar.R: Solar radiation in Langleys in the frequency band 4000–7700 Angstroms from 0800 to 1200 hours at Central Park
  - Wind: Average wind speed in miles per hour at 0700 and 1000 hours at LaGuardia Airport
  - Temp: Maximum daily temperature in degrees Fahrenheit at La Guardia Airport.
  - Month: Numeric value between 1-12
  - Day: Numeric value between 1-31

Upload Numeric Data

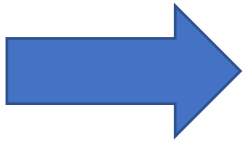
```
14  ## Download data from R using data() and see what the set is composed of
15  ## Make sure you download data in the working directory
16
17  data("airquality")
18  str(airquality)
```

```
> str(airquality)
'data.frame':   153 obs. of  6 variables:
 $ Ozone  : int  41 36 12 18 NA 28 23 19 8 NA ...
 $ Solar.R: int  190 118 149 313 NA NA 299 99 19 194 ...
 $ Wind   : num  7.4 8 12.6 11.5 14.3 14.9 8.6 13.8 20.1 8.6 ...
 $ Temp   : int  67 72 74 62 56 66 65 59 61 69 ...
 $ Month  : int  5 5 5 5 5 5 5 5 5 5 ...
 $ Day    : int  1 2 3 4 5 6 7 8 9 10 ...
>
```

Data Cleaning

```
20  ## To remove NA values, we use complete.cases() which will assign all NA as False,
21  ## else, True.
22  complete.cases(airquality)
23
24  ## To drop values option 1:
25  x <- airquality[complete.cases(airquality), ]
26  str(x)
27
28  ## To drop values option 2:
29  y <- na.omit(airquality)
30  str(y)
31
```

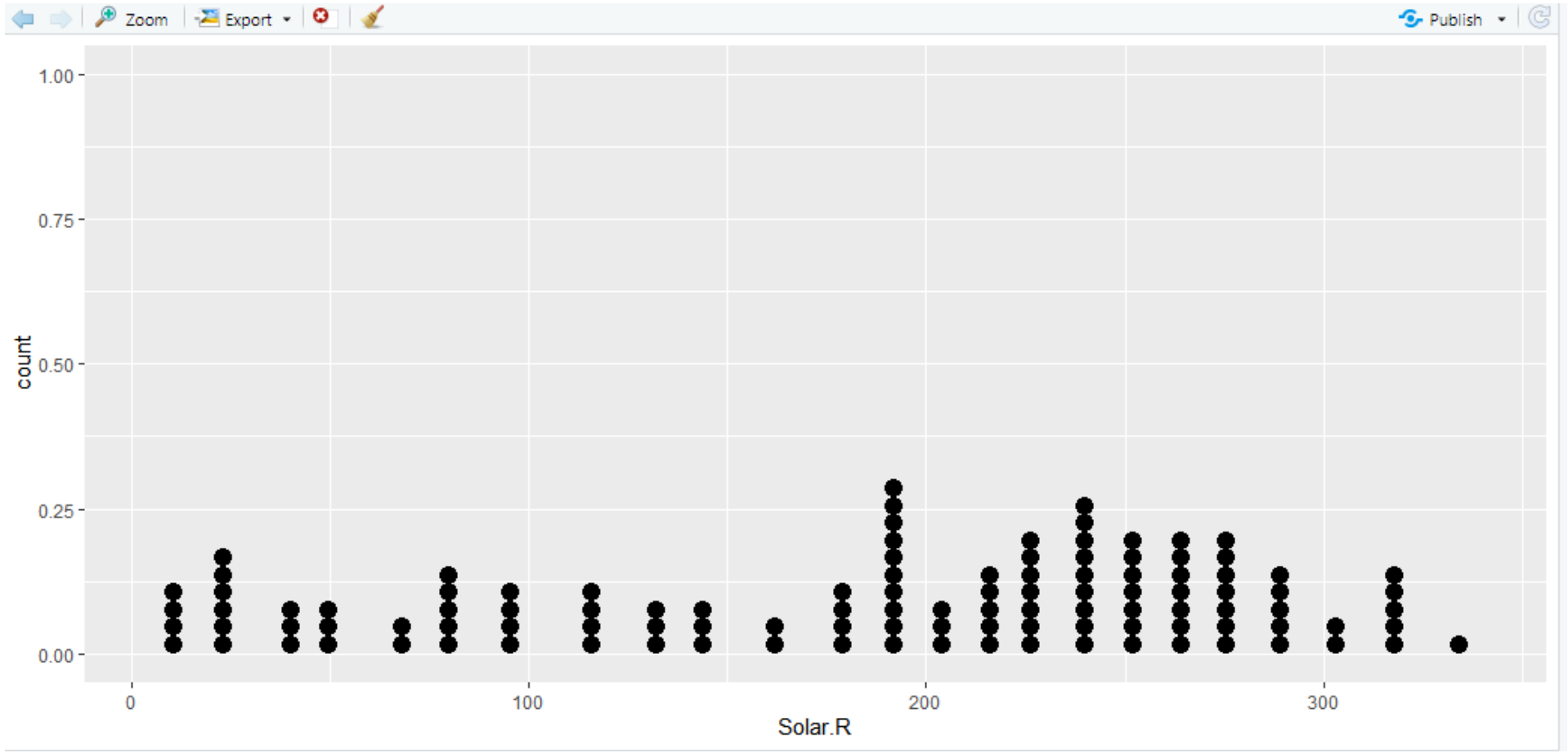
Output



```
> str(x)
'data.frame':   111 obs. of  6 variables:
 $ Ozone  : int  41 36 12 18 23 19 8 16 11 14 ...
 $ Solar.R: int  190 118 149 313 299 99 19 256 290 274 ...
 $ Wind   : num  7.4 8 12.6 11.5 8.6 13.8 20.1 9.7 9.2 10.9 ...
 $ Temp   : int  67 72 74 62 65 59 61 69 66 68 ...
 $ Month  : int  5 5 5 5 5 5 5 5 5 5 ...
 $ Day    : int  1 2 3 4 7 8 9 12 13 14 ...
> ## To drop values option 2:
> y <- na.omit(airquality)
> str(y)
'data.frame':   111 obs. of  6 variables:
 $ Ozone  : int  41 36 12 18 23 19 8 16 11 14 ...
 $ Solar.R: int  190 118 149 313 299 99 19 256 290 274 ...
 $ Wind   : num  7.4 8 12.6 11.5 8.6 13.8 20.1 9.7 9.2 10.9 ...
 $ Temp   : int  67 72 74 62 65 59 61 69 66 68 ...
 $ Month  : int  5 5 5 5 5 5 5 5 5 5 ...
 $ Day    : int  1 2 3 4 7 8 9 12 13 14 ...
 - attr(*, "na.action")= 'omit' Named int [1:42] 5 6 10 11 25 26 27 32 33 34 ...
  ..- attr(*, "names")= chr [1:42] "5" "6" "10" "11" ...
>
```

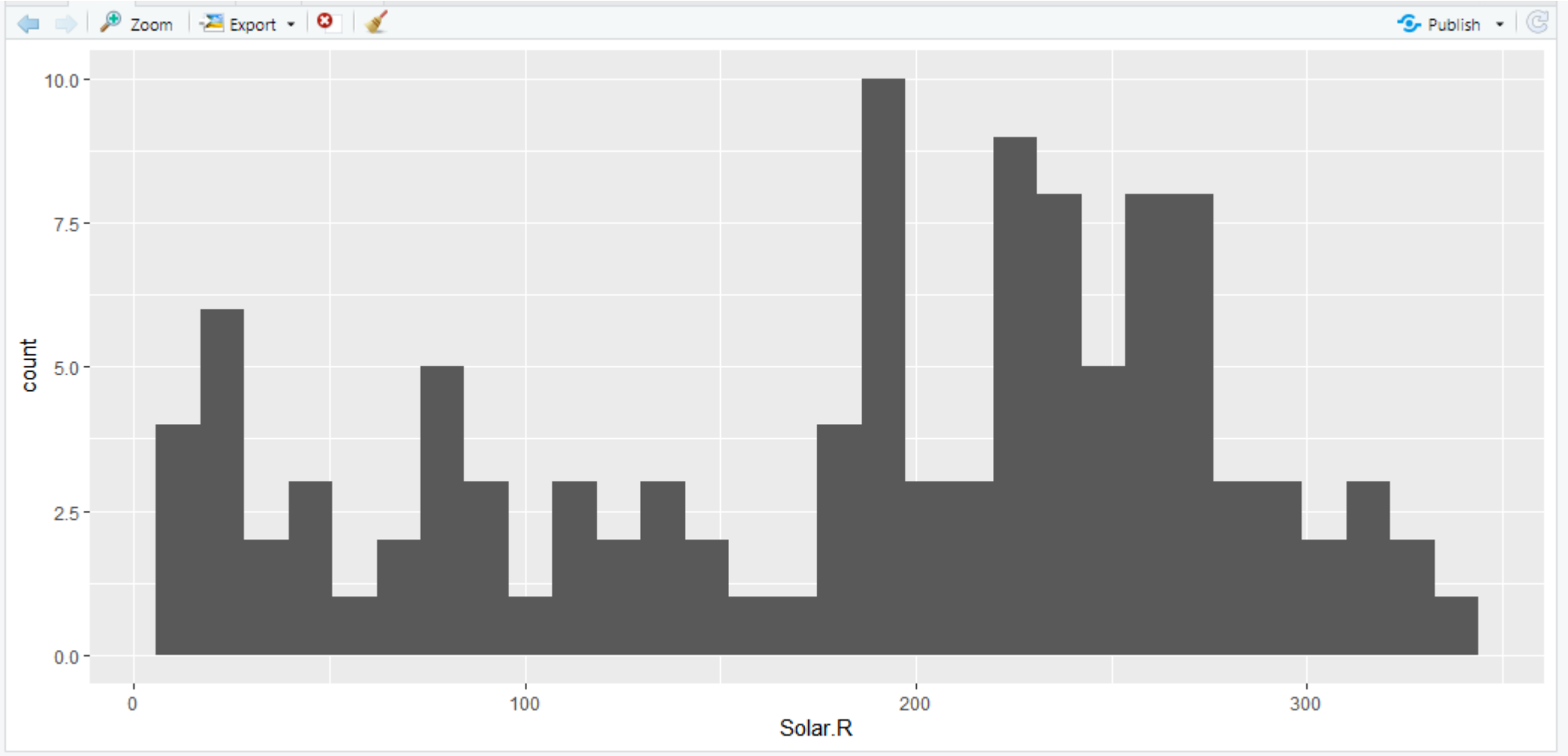
Dotplot

```
32  ## Making a dotplot to show numerical data. It's like a bar chart,
33  ## but with points stacked on top of each other
34  ggplot(y,aes(x=Solar.R)) + geom_dotplot(dotsize=0.4)
```



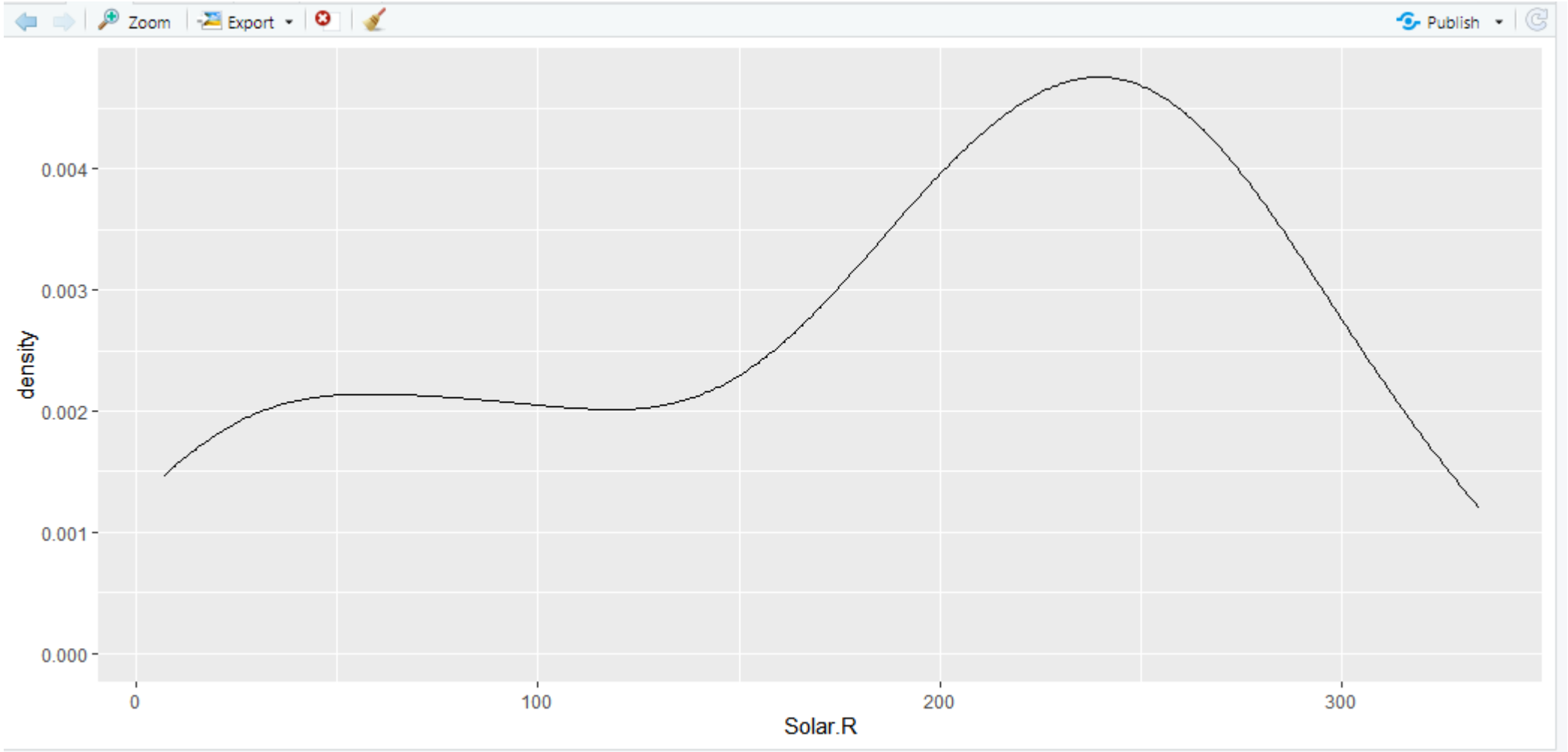
Histogram

```
36  ## Histogram combines the dots, and the y axis now shows the actual count
37  ggplot(y,aes(x=Solar.R)) + geom_histogram()
```



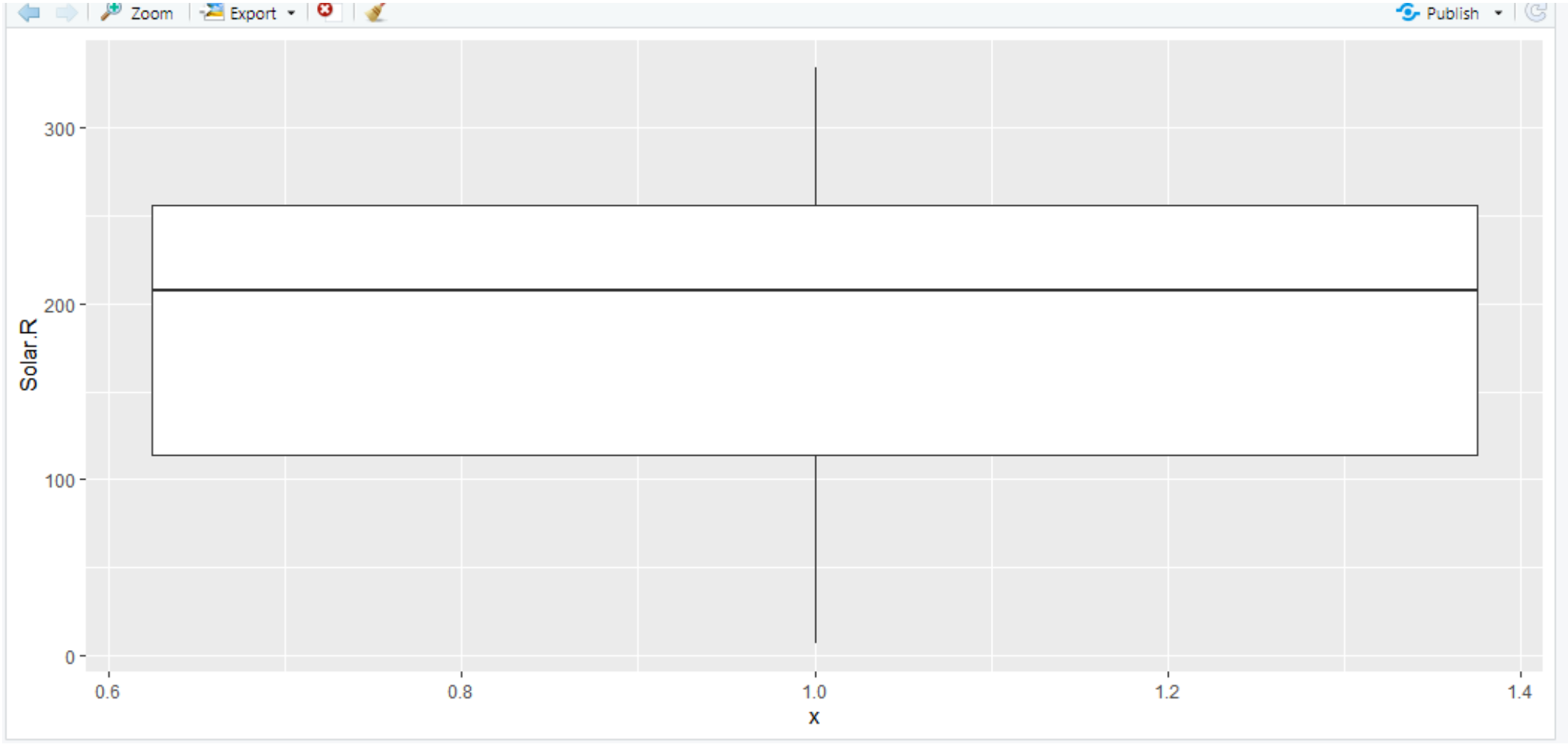
Density plot

```
39  ## The shape of the distribution can be better represented with a density plot,
40  ## without the stepwise nature of a histogram
41  ggplot(y,aes(x=Solar.R)) + geom_density()
```



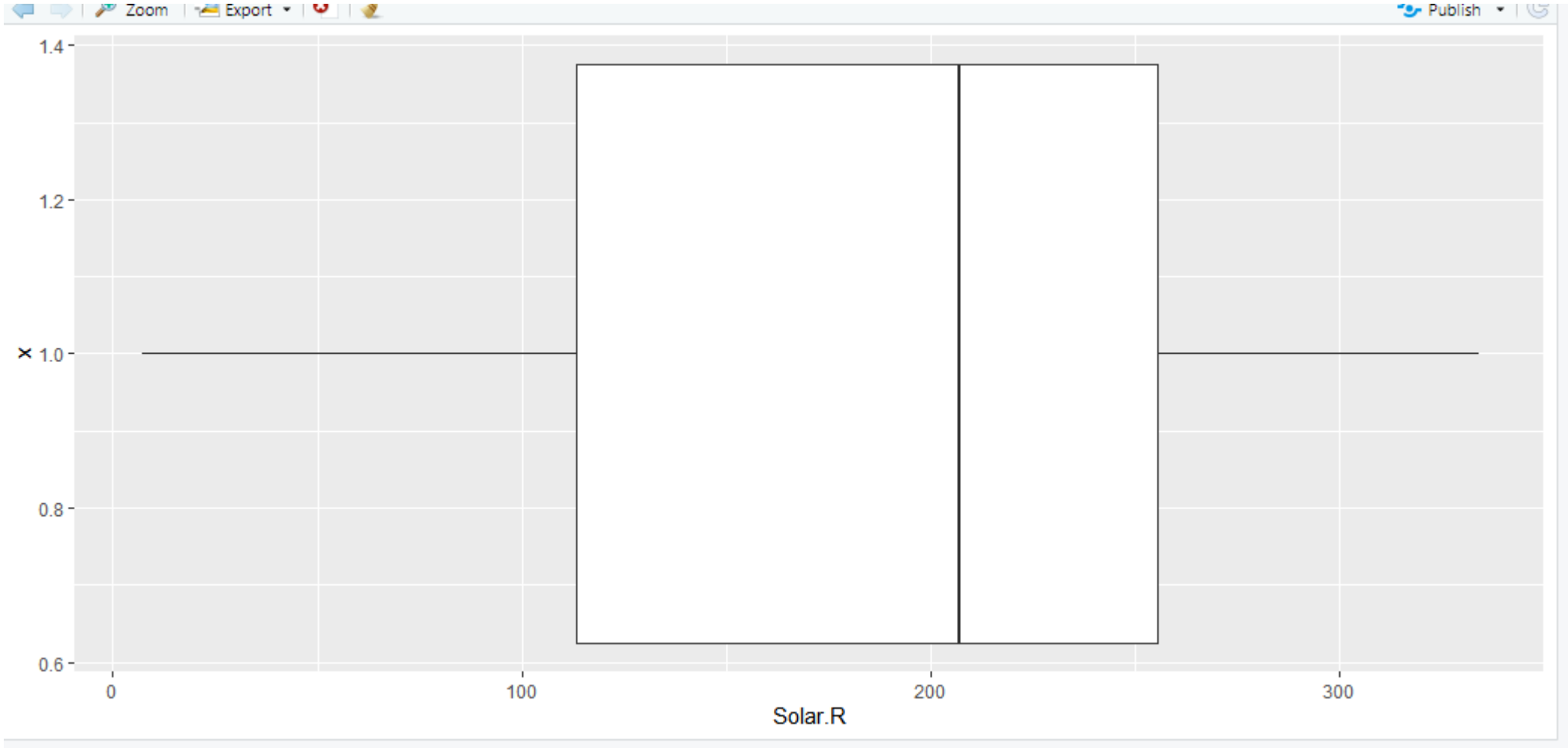
Boxplot

```
43  ## Another view of distribution where you use a boxplot
44  ggplot(y,aes(x=1,y=Solar.R)) + geom_boxplot()
```



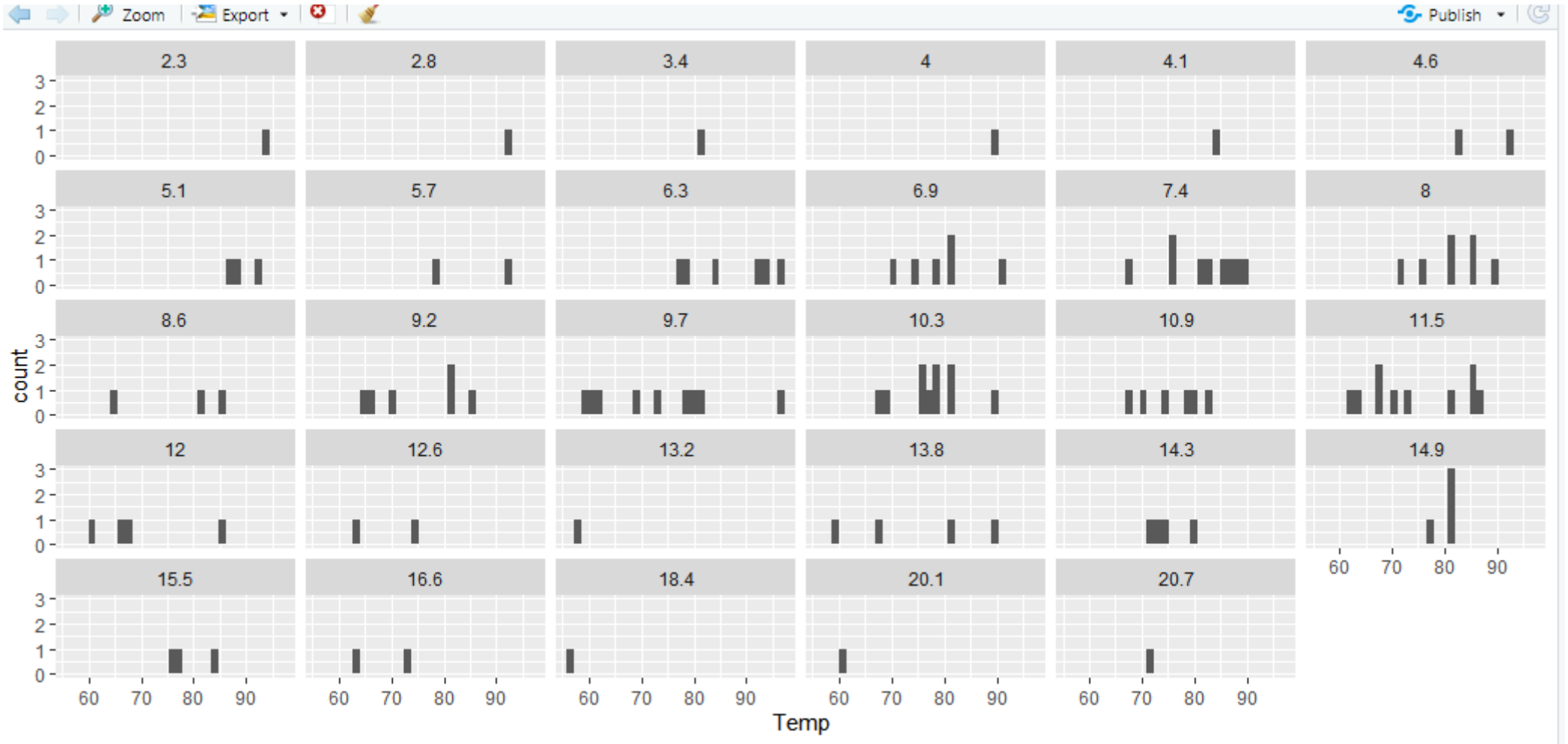
Boxplot (coord_flipped)

```
46  ggplot(y,aes(x=1,y=Solar.R)) + geom_boxplot() + coord_flip()
47
```



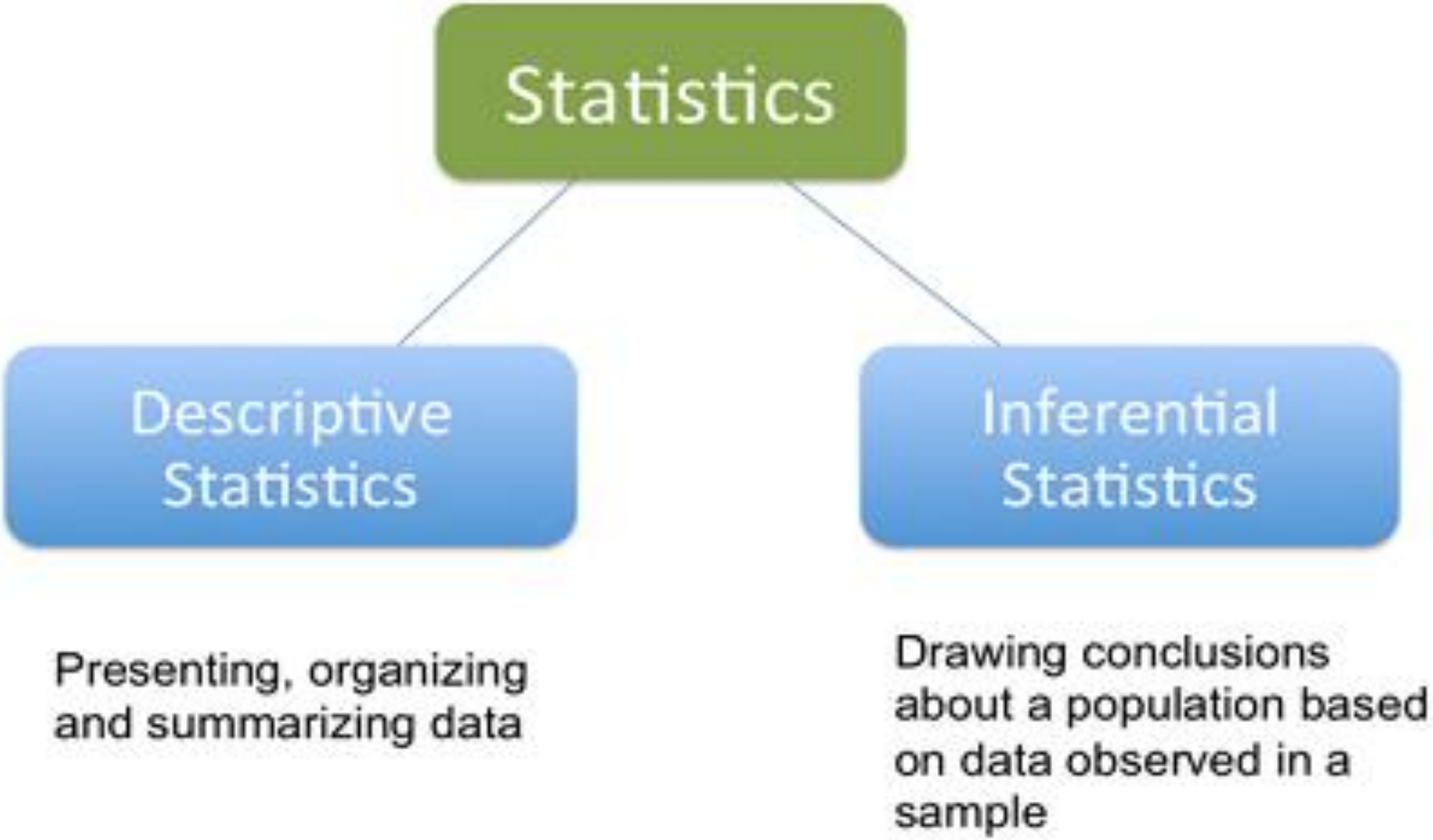
Faceted plots

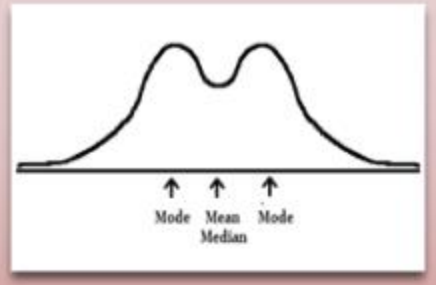
```
48  ## Temperature faceted by wind speeds
49  ggplot(y,aes(x=Temp)) + geom_histogram() + facet_wrap(~Wind)
50
```



Module 5: Statistics

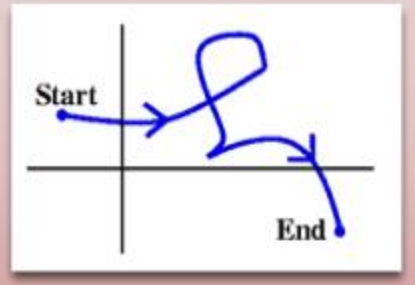
BeVera


Statistics
Descriptive Statistics
Inferential Statistics
Presenting, organizing and summarizing data
Drawing conclusions about a population based on data observed in a sample



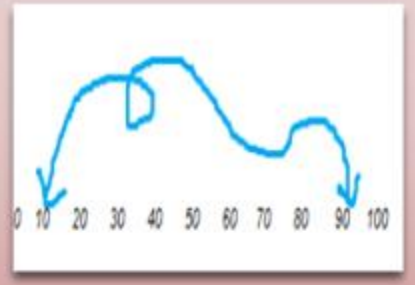
Central Tendency

Mean, Median, Mode, Outliers



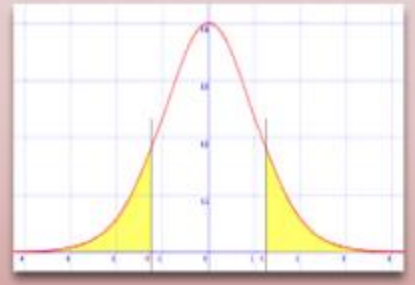
Measures of Spread

Range, Standard deviation, Variance, Quartiles



Percentiles

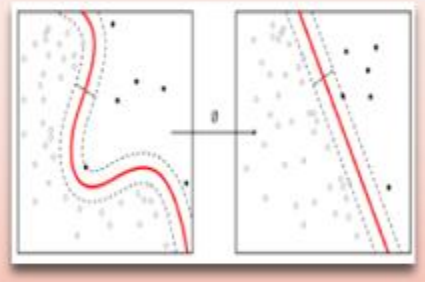
Position of data, percentile rank, percentile range



Probability Distributions:

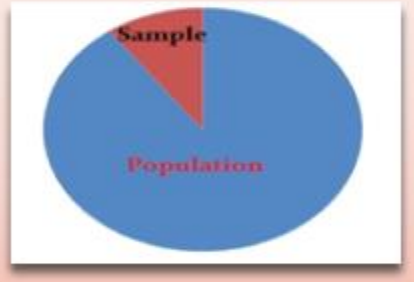
Uniform, normal (Gaussian), Poisson

Basic Probability and Statistics



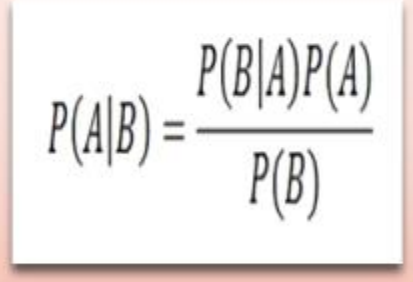
Dimensionality reduction

Pruning, PCA



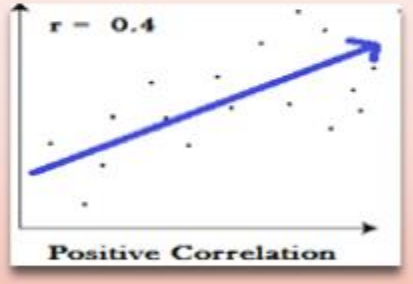
Sampling

SRS, Reservoir, Undersampling, Oversampling,



Bayesian statistics

Measuring belief or confidence



Covariance & correlation

How data is related

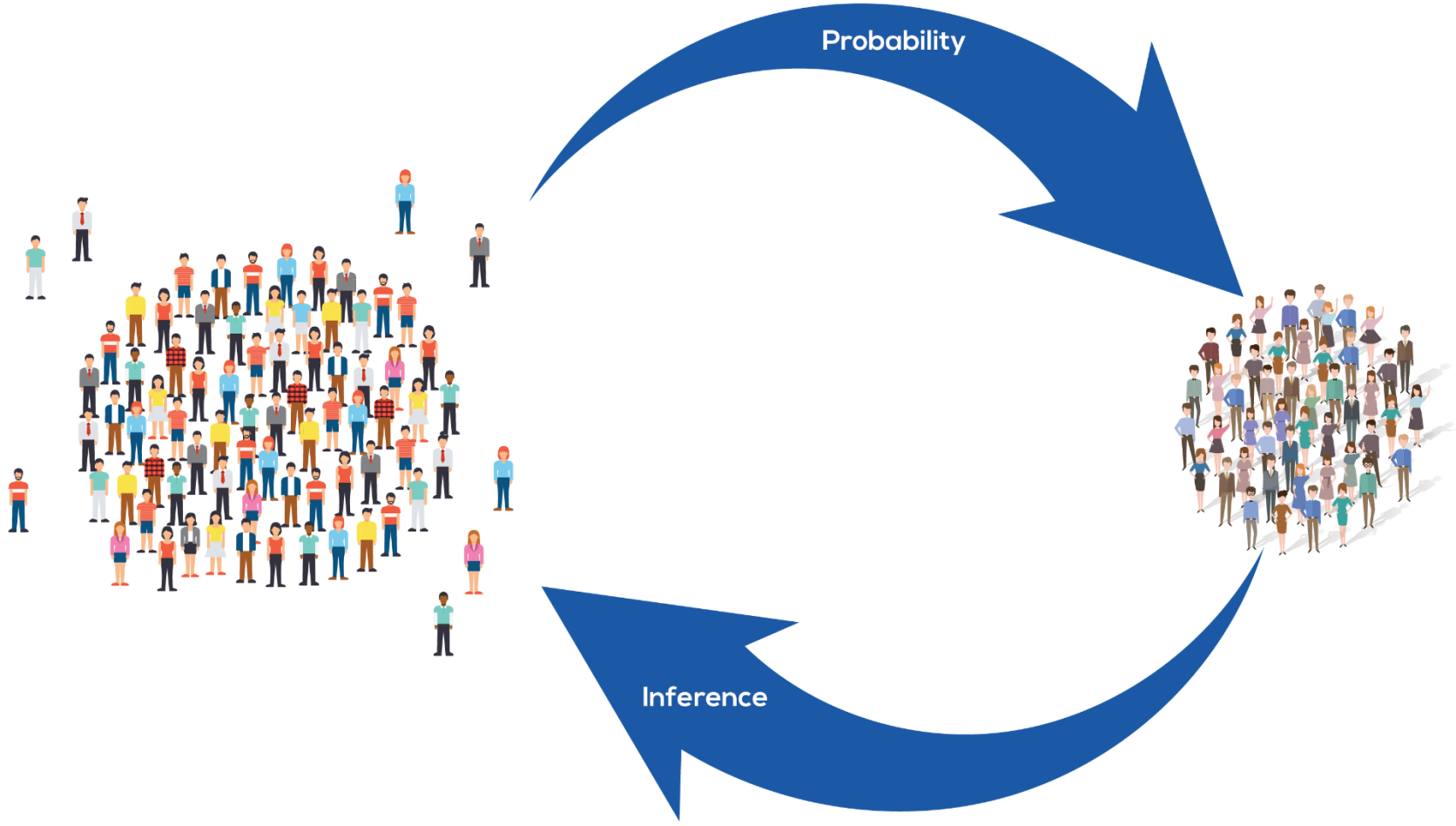
More Advanced Probability and Statistics

The area of *descriptive statistics* is concerned with meaningful and efficient ways of presenting data.

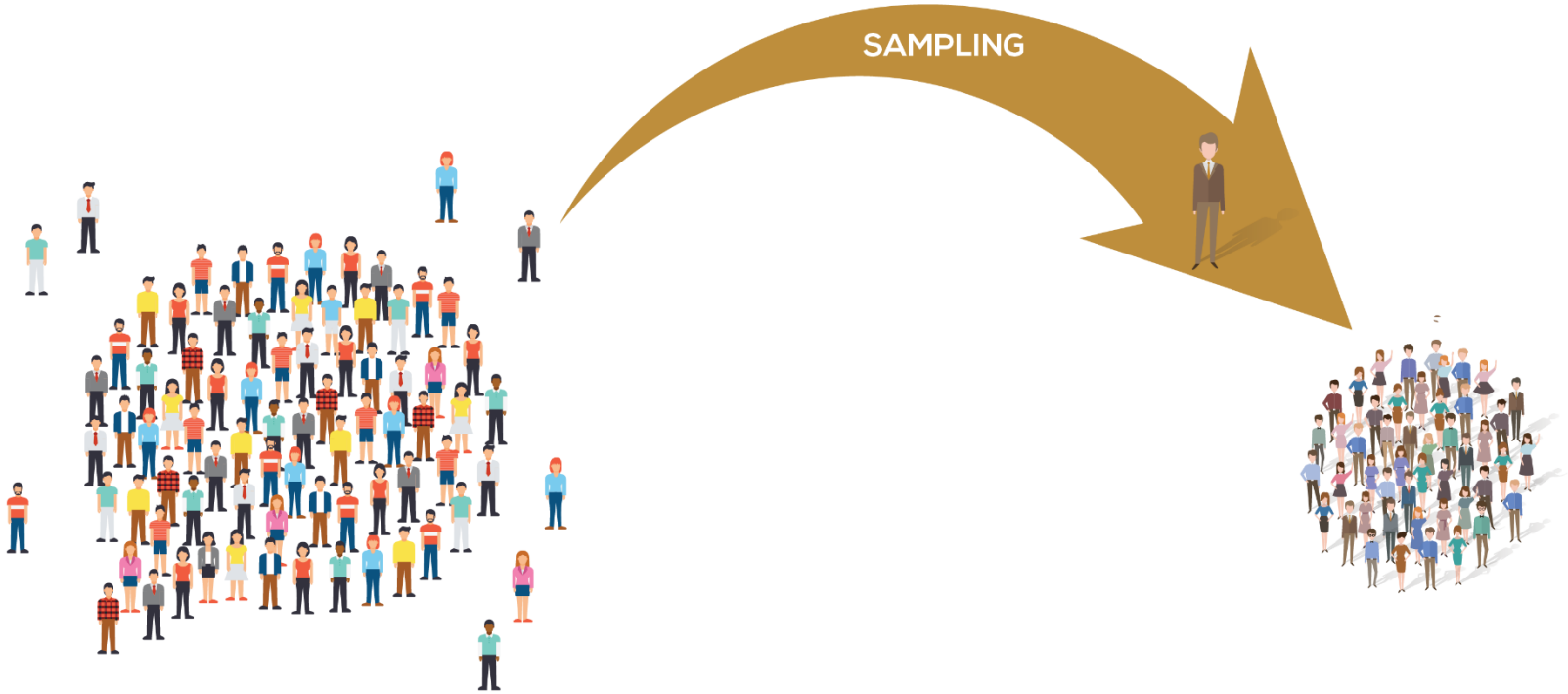
When it comes to *inferential statistics*, though, our goal is to make some statement about a characteristic of a population based on what we know about a sample drawn from that population.

S. No	Descriptive Statistics	Inferential Statistics
1	Concerned with the describing the target population	Make inferences from the sample and generalize them to the population.
2	Organize, analyze and present the data in a meaningful manner	Compares, test and predicts future outcomes.
3	Final results are shown in form of charts, tables and Graphs	Final result is the probability scores.
4	Describes the data which is already known	Tries to make conclusions about the population that is beyond the data available.
5	Tools- Measures of central tendency (mean/median/ mode), Spread of data (range, standard deviation etc.)	Tools- hypothesis tests, Analysis of variance etc.

Inferential Statistics



Sampling



Generating random data

sample(x, size, replace = FALSE, prob = NULL)

```
> set.seed(1)
> sample(1:6, 10, replace=TRUE)

> sample(1:6, 10, replace=TRUE)

> set.seed(123)
> index <- sample(1:nrow(iris), 5)
> index
> iris[index, ]
```

Simple random distributions:

1. Normal (aka Gaussian): bell-shaped, and has two parameters: a mean and a standard deviation.

- *# 6 samples from a Normal dist with mean = 0, sd = 1*

rnorm(n = 6, mean = 0, sd = 1)

- *# 4 samples from a Normal dist with mean = -10, sd = 15*

rnorm(n = 4, mean = -10, sd = 20)

2. Uniform: everything between its lower and upper bounds are equally likely to occur.

- *# 5 samples from Uniform dist with bounds at 0 and 1*

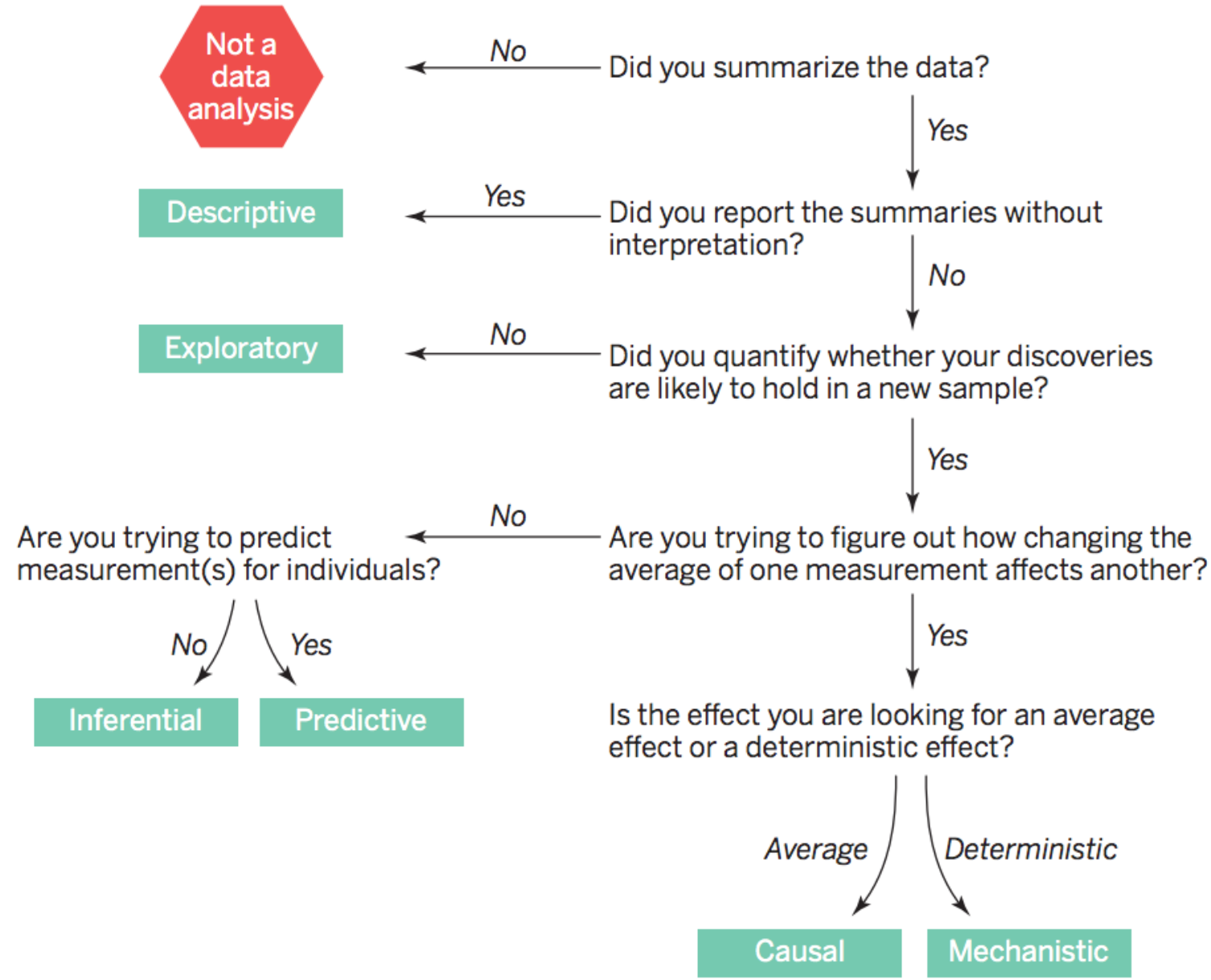
runif(n = 5, min = 0, max = 1)

- *# 10 samples from Uniform dist with bounds at -75 and +75*

runif(n = 5, min = -75, max = 75)

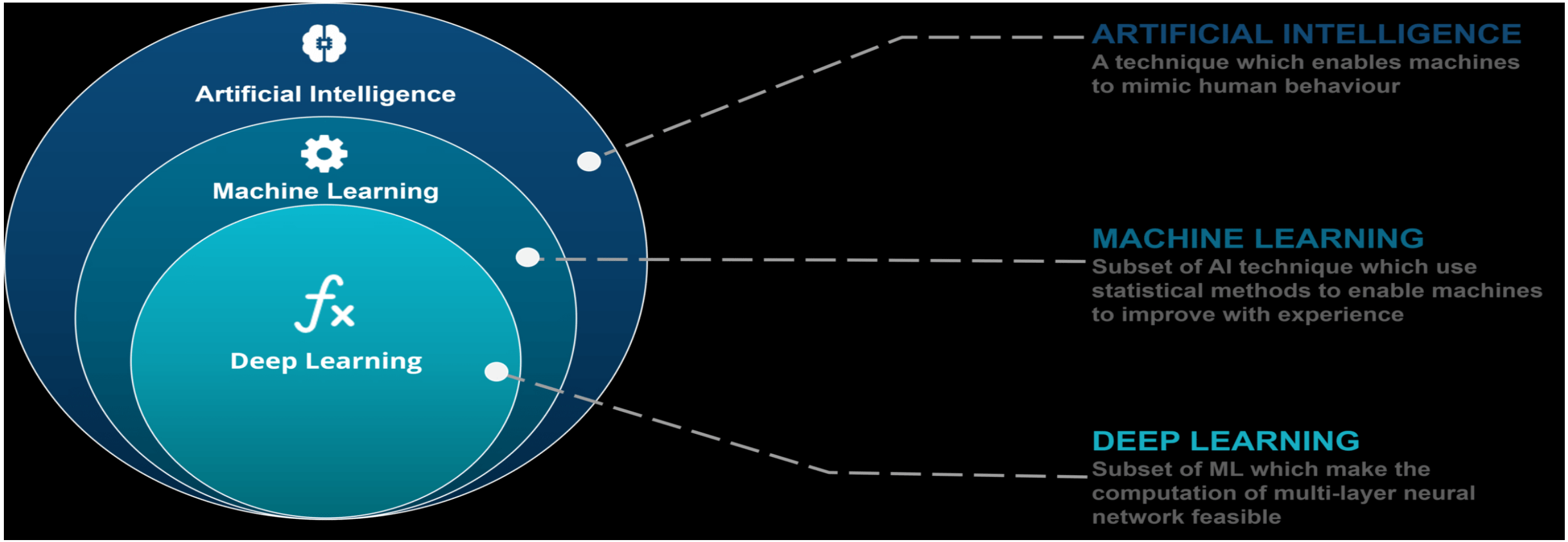

NEXT
CLASS

Data analysis flowchart



Module 5: Machine Learning

BeVera


Artificial Intelligence
Machine Learning
Deep Learning
ARTIFICIAL INTELLIGENCE
A technique which enables machines to mimic human behaviour
MACHINE LEARNING
Subset of AI technique which use statistical methods to enable machines to improve with experience
DEEP LEARNING
Subset of ML which make the computation of multi-layer neural network feasible

Applying the Data Analytic Lifecycle

In a typical Data Analytical Problem – you would have gone through:

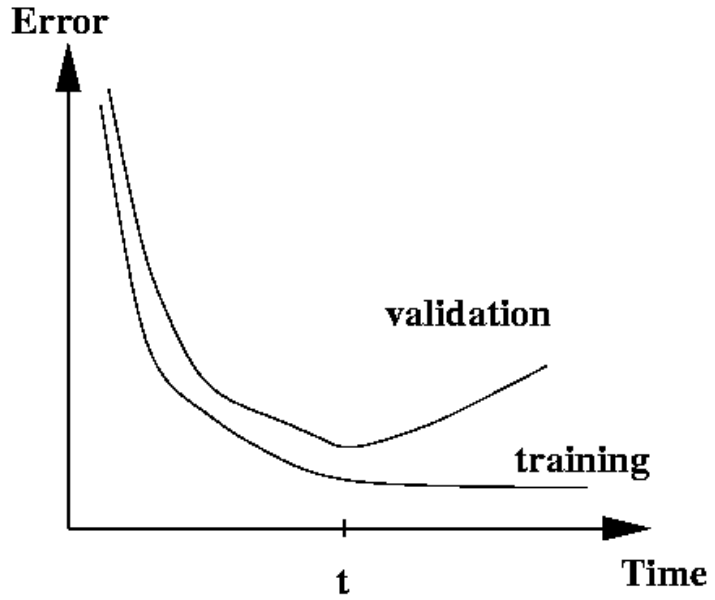
- Phase 1 – Discovery – have the problem framed
- Phase 2 – Data Preparation – have the data prepared

Now you need to plan the model and determine the method to be used.

- Phase 3 – Model Planning
  - Have do people generally solve this problem with the kind of data and resources I have?
    - Does this work well enough? Or do I need to come up with something new?
    - What are related or analogous problems? How are they solved? Can I do that?

Datasets

- **Training set**: a set of examples used for learning, where the target value is known.
- **Validation set**: a set of examples used to tune the architecture of a classifier and estimate the error.
- **Test set**: used only to assess the performances of a classifier. It is never used during the training process so that the error on the test set provides an unbiased estimate of the generalization error.



Machine Learning

- To learn: *to get knowledge of by study, experience, or being taught*.

- Types of Learning
  - Supervised
  - Unsupervised

Unsupervised Learning

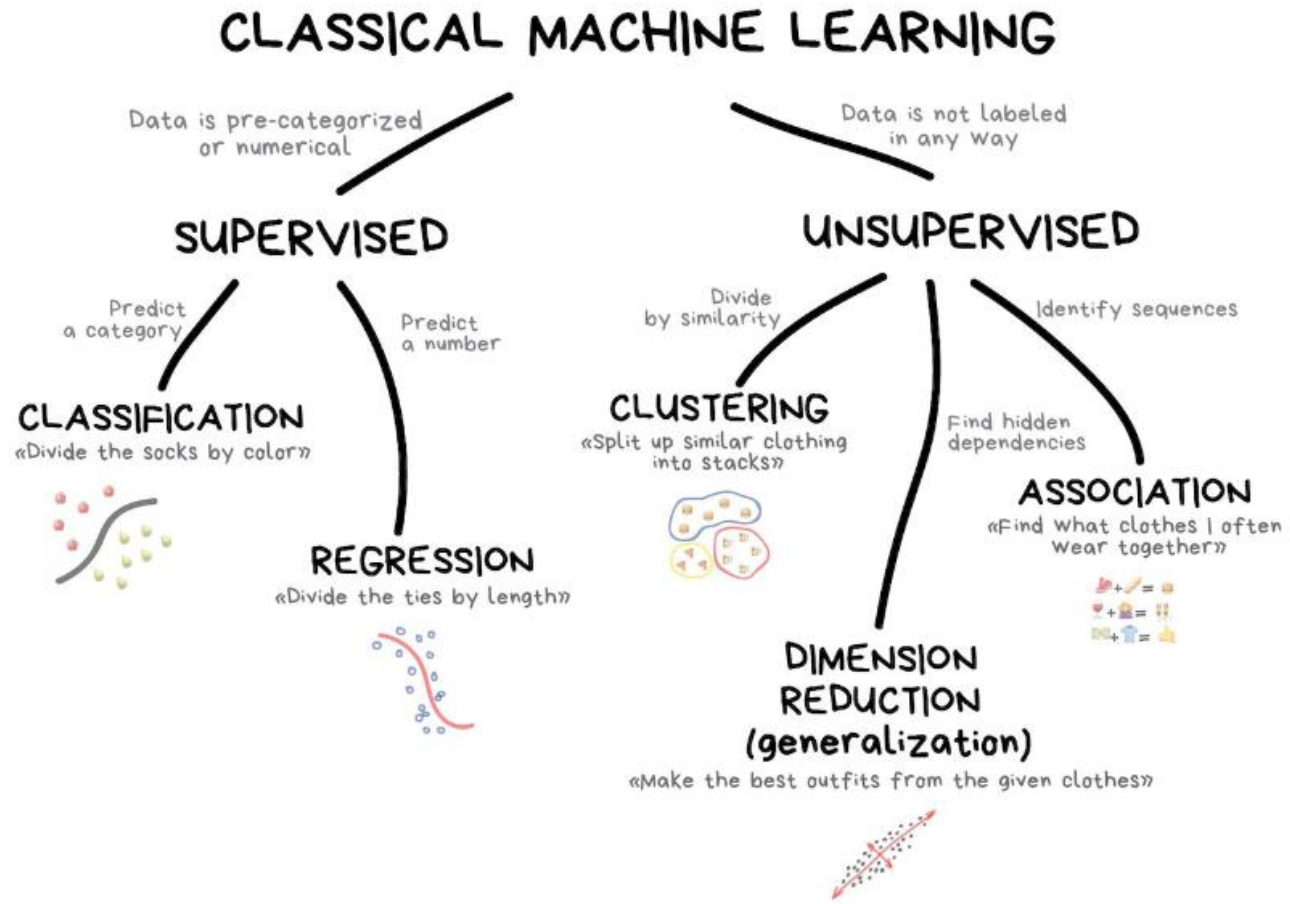
- The model is not provided with the correct results during the training.
- Can be used to cluster the input data in classes on the basis of their statistical properties only.
- Cluster significance and labeling.
- The labeling can be carried out even if the labels are only available for a small number of objects representative of the desired classes

Supervised Learning

- Training data includes both the input and the desired results.
- For some examples the correct results (targets) are known and are given in input to the model during the learning process.
- The construction of a proper training, validation and test set is crucial.
- These methods are usually fast and accurate.
- Have to be able to **generalize**: give the correct results when new data are given in input without knowing a priori the target.

Theory and Methods

- Examine analytic needs and select an appropriate technique based on business objective initial hypothesis; and the data's structure and volume
- Apply some of the more commonly used methods in Analytics solutions
- Explain the algorithms and the technical foundations for the commonly used methods
- Explain the environment (use case) in which each technique can provide the most value
- Use appropriate diagnostic methods to validate the models created
- Use R to fit, score and evaluate models



- Discovers patterns naturally from text examples.
- Using statistical methods, documents are compared to one another to determine the most important and useful patterns in the corpus for the desired behavior.
- Methods are diverse and can range from simple to complex.
- All share the same fundamental goal of learning the most valuable and distinctive patterns based on examples provided by a human.

What kind of Problem do I need to Solve?

The Problem to Solve	The Category of Techniques	Analytical Method
I want to group items by similarity. I want to find structure (commonalities) in the data.	Clustering	K-means clustering
I want to discover relationships between actions or items	Association Rules	Apriori
I want to determine the relationships, between the outcome and the input variables.	Regression	Linear Regression Logistic Regression
I want to assign (known) labels to objects.	Classification	Naïve Bayes Decision Trees
I want to find the structure in a temporal process. I want to forecast the behavior of a temporal process.	Time Series Analysis	ACF, PACF, ARIMA
I want to analyze my text data.	Text Analysis	Regular expressions, Document representation (Bag of Words), TF-IDF

Data Science with R

Special Topics

Text Analysis & NLP

- Text Mining with R (Book): tidytext
- NLP:
  - H2O word2vec: Word embeddings
  - text2vec: fast vectorization, topic modeling
  - udpipe: UDPipe C++ lib in R

Machine Learning

- Multi-Threaded/Scalable/Production ML:
  - H2O (CS)
  - Extreme Gradient Boosting: xgboost
  - R + Spark: sparklyr (CS)
  - Sparkling Water (Spark + H2O): rsparkling
- ML (Tidy): parsnip
- ML: caret (CS)

Time Series Analysis

- Time-aware tibbles: tibbletime & tsibble
- Convert between classes: timetk & tsbox
- Time Series Index Summary: timetk
- Generating Future Series: timetk

Forecasting

- ARIMA, ETS, etc: forecast & fable
- Tidy, glance, augment for forecast models: sweep
- Converting forecast prediction to tibble: sweep

Anomaly Detection

- Identify anomalies: anomalize

Network Analysis

- Network Data Transformations (Tidy): tidygraph
- Network Data Transformations: igraph

Network Viz

- Static:
  - ggraph - Graph plotting utilities for ggplot2
- Interactive (JavaScript):
  - networkD3 - D3 Networks in R
  - plotly - plotly.js (network graphs) in R

Deep Learning

- R Interface to TensorFlow Homepage:
  - Keras (CS)
  - TF Estimators
  - TensorFlow (Core)

Speed & Scale

- Fastest Single-Node Speed: data.table (CS)
- Distributed Cluster (Spark): sparklyr (CS)

Financial Analysis

- Getting financial data: tidyquant & quantmod
- Quantitative Analysis: tidyquant & xts/TTR
- Portfolio Analysis: tidyquant & PerformanceAnalytics

Financial & Time Viz

- Static:
  - tidyquant - Financial ggplot2 geoms
- Interactive:
  - highcharter - highchart.js in R
  - dygraphs - xts plotting
  - plotly - plotly.js (financial) in R

Geospatial Analysis

- Geocoding (getting lat/long, bboxes, & sf's):
  - ggmap - Google API (requires key)
  - osmdata - OpenStreet Overpass API
  - tmaptools - OpenStreet Nominatum API
- Simple Features (sf objects): sf (CS) (tidy)
- Spatial Objects (sp objects): sp (non-tidy)

Geospatial Viz

- Static:
  - ggmap - Google API (requires key)
  - osmplotr - Impressive Maps via OSM
  - tmap - Thematic Maps
  - cartography (CS) - Thematic Maps
- Interactive (JavaScript):
  - leaflet (CS) - leaflet.js in R
  - plotly - plotly.js (maps) in R

Interoperability

- Python: reticulate
- Java: rJava
- C++: Rcpp

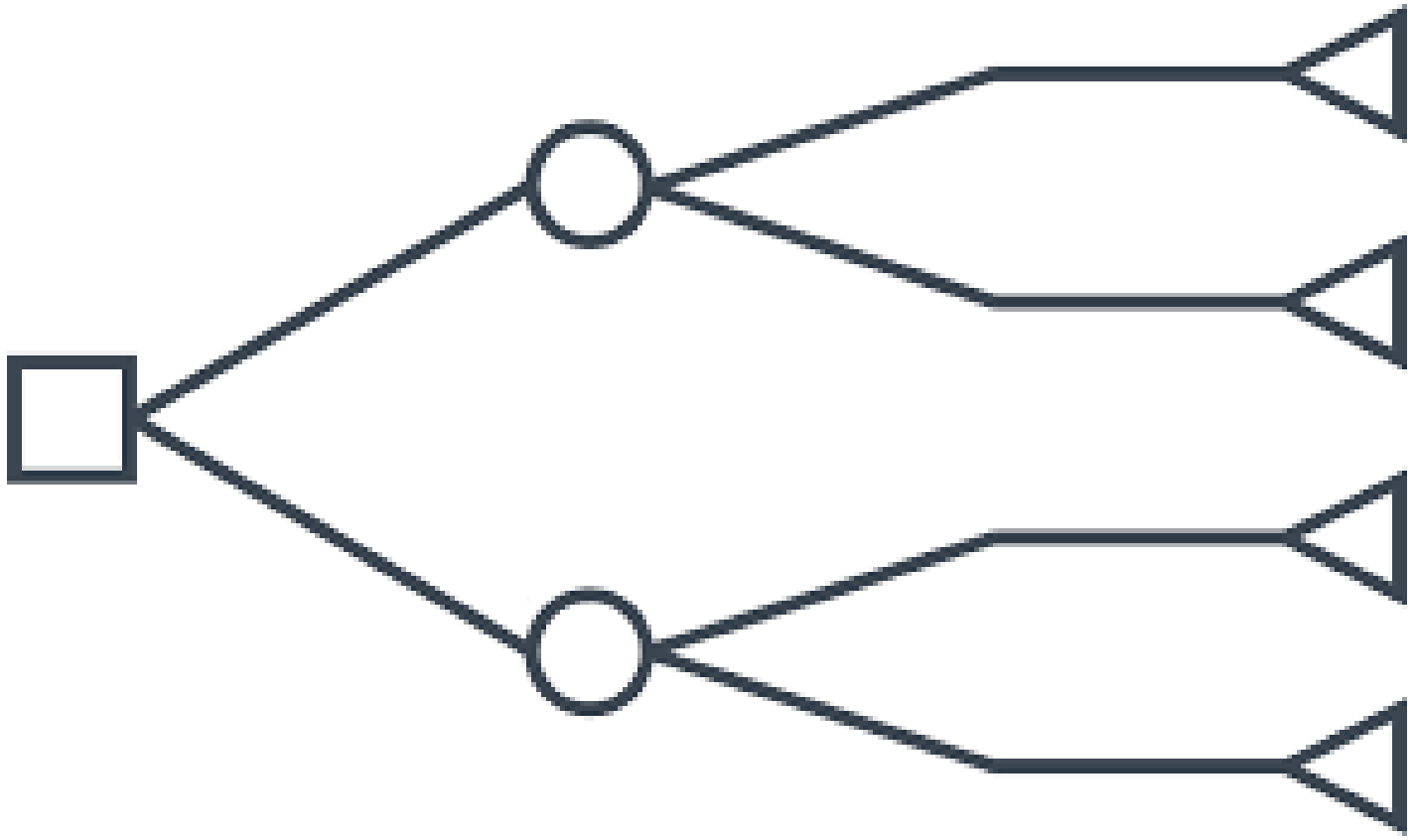
Miscellaneous Tools

- **Interactive Plotting:** htmlwidgets for R
- **Building R Packages:** R packages Book
  - Pkg Development Tools: devtools (CS)
  - R Templates: usethis
  - Build Web Doc's: pkgdown
- **Advanced Concepts** (Advanced R Book)
  - rlang & Tidy Evaluation (CS)
- **Making Blogs & Books:**
  - Make a Website/Blog: blogdown
  - Write a Web Book: bookdown
- **Posting Code (GitHub, Stack Overflow):** reprex

"Data Science Education for the Enterprise"

Business Science University
university.business-science.io

Decision Trees



Decision Tree Induction

- Many Algorithms:
  - Hunt's Algorithm (one of the earliest)
  - CART – Classification and Regression Trees
  - ID3, C4.5
  - SLIQ,SPRINT

Decision tree classifier – what is it?

- Used for classification:
- Returns probability scores of class membership
  - Well-calibrated, like logistic regression
  - Assigns labels based on highest scoring class
  - Some Decision Tree algorithms return simply the mostly class
- Regression Trees: a variation for regression
  - Returns average value at every node
  - Predications can be discontinuous at the decision boundaries
- Input variables can be continuous or discrete
- Output:
  - A tree that describes the decision flow.
  - Leaf nodes return either a probability score or simply a classification (label).
  - Trees can be converted to a set of “decision rules”.
    - If/Then statements

Identify…..

- Branch?
- Internal Nodes?
- Leaf Nodes?
- Parents?
- Children?

Tree Induction

- Greedy strategy.
  - Split the records based on an attribute test that optimizes certain criterion.
- Issues
  - Determine how to split the records
    - How to specify the attribute test condition?
    - How to determine the best split?
  - Determine when to stop splitting

Measures of node Impurity

- Gini Index
- Entropy
- Misclassification error

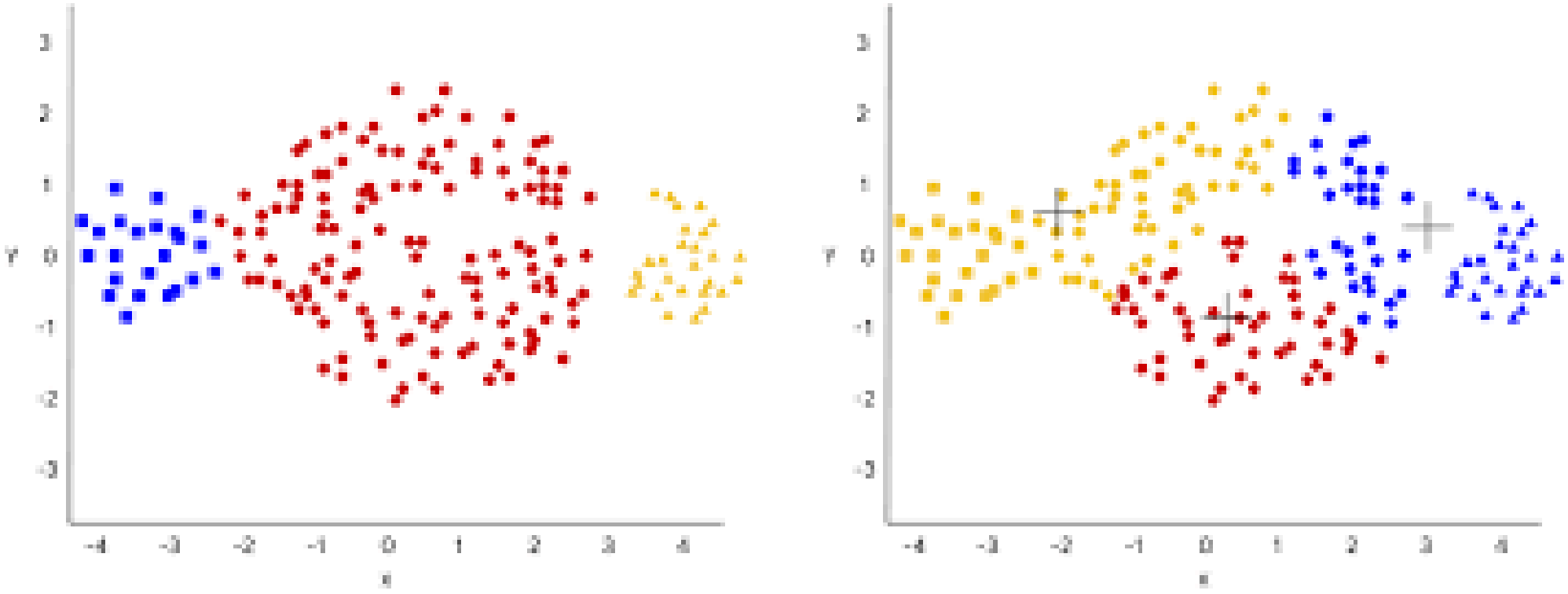
Advantages of Decision Tree Based Classification (CART)

- Simple to understand, interpret, visualize.
- Decision trees *implicitly perform variable screening or feature selection.*
- Can *handle both numerical and categorical data*. Can also *handle multi-output problems.*
- Decision trees require relatively *little effort from users for data preparation.*
- *Nonlinear relationships between parameters do not affect tree performance.*

Disadvantages of Decision Tree Based Classification (CART)

- Learners can create over-complex trees that do not generalize the data well. This is called *overfitting*.
- Unstable because small variations in the data might result in a completely different tree being generated. This is called *variance,* which needs to be lowered by methods like *bagging and boosting.*
- *Greedy algorithms cannot guarantee to return the globally optimal decision tree.* This can be mitigated by training multiple trees, where the features and samples are randomly sampled with replacement.
- Decision tree learners *create biased trees if some classes dominate*. It is therefore recommended to balance the data set prior to fitting with the decision tree.

K-Means clustering



Clustering

- How do I group these documents by topic?
- How do I group my customers by purchase patterns?
  - Sort items into groups by similarity:
    - Items in a cluster are more similar to each other than they are to items in other clusters.
    - Need to detail the properties that characterize “similarity”
      - Or of distance, then “inverse” of similarity
  - Not a predictive model; finds similarities, relationships (unsupervised)

K-Means Clustering – What is it?

- Used for clustering numerical data, usually a set of measurements about objects of interest.
- Input: numerical. There must be a distance metric defined over the variable space.
  - Euclidian distance – absolute value of the differences between two points
    - Most popular method for calculating distance
    - The square root of the sum of squares
- Output: The centers of each discovered cluster, and the assignment of each input datum to a cluster.
  - Centroid – the center of the discovered cluster.
  - K-means provides this an output

Use Cases

- Often an exploratory technique:
  - Discover structure in the data
  - Summarize the properties of each cluster
- Sometimes a prelude to classification:
  - “Discovering the classes”
- Examples
  - The height, weight and average lifespan of animals
  - Household income, yearly purchase amount in dollars, number of household members of customer households
  - Patient record with measures of BMI, HBA1C, HDL

The Algorithm

1. Choose K; then select K random “centroids”
2. Assign records to the cluster with the closest centroid
3. Recalculate the resulting centroids

   Centroid: the mean value of all the records in the cluster
4. Repeat steps 2 & 3 until record assignments no longer change

   Model Output:
   - The final cluster centers
   - The final cluster assignments of the training data

Diagnostics – evaluating the model –

How do we know we have good clusters?

- Do the clusters look separated in at least some of the plots when you do pair-wise plots of the clusters?
  - Pair-wise can be used when there are not many variables
- Do you have any clusters with few data points?
  - Try decreasing the value of K
- Are there splits on variables that you would expect, but don’t see?
  - Try increasing the value of K
- Do any of the centroids seem too close to each other?
  - Try decreasing the value of K

K-means Clustering: Advantages and Disadvantages

Advantages	Disadvantages
• Relatively simple to implement.	• Choosing manually. Use the "Loss vs. Clusters" plot to find the optimal (k), as discussed in Interpret Results.
• Scales to large data sets.	• Being dependent on initial values. Look for a discussion of k- means seeding.
• Guarantees convergence.	• Clustering data of varying sizes and density. k-means has trouble clustering data where clusters are of varying sizes and density.
• Can warm-start the positions of centroids.	• Clustering outliers. Consider removing or clipping outliers before clustering.
• Easily adapts to new examples.	• Scaling with number of dimensions.Reduce dimensionality either by using PCA on the feature data, or by using "spectral clustering" to modify the clustering algorithm as explained below.
• Generalizes to clusters of different shapes and sizes, such as elliptical clusters.	

The k-means clustering algorithms aims at partitioning n observations into a fixed number of k clusters. The algorithm will find homogeneous clusters.

- In R, we use stats::kmeans(x, centers = 4, nstart = 18)

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

#Visualize data
plot(tmp)
# create 4 clusters
kmeans=kmeans(tmp,4,15)

#Visualize the cluster centers
points(kmeans$centers, col = 1:4, pch=20)
```

Check your knowledge

1. Why do we consider K-means clustering as a unsupervised machine learning algorithm?

2. How do you use “pair wise” plots to evaluate the effectiveness of the clustering?

3. Detail the four steps in the K-means clustering algorithm.

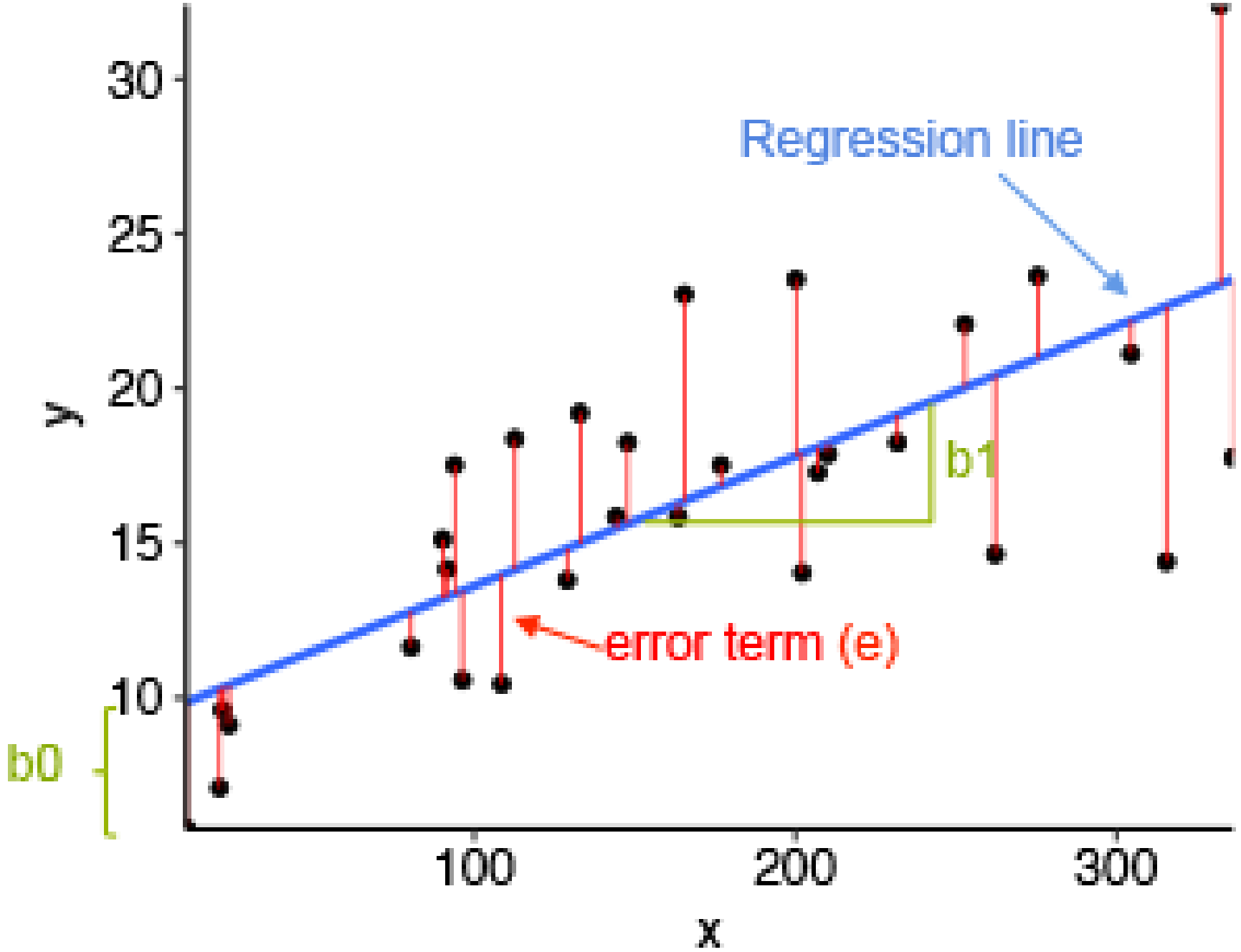
4. How do we use WSS to pick the value of K?

5. What is the most common measure of distance used with K-means clustering algorithms?

Linear Regression - What is it?

- Used to estimate a continuous value as a linear (additive) function of other variables
  - Income as a function of years of education, age, gender
  - House price as function of median home price in neighborhood, square footage, number of bedrooms/bathrooms
  - Neighborhood house sales in the past year based on unemployment, stock price etc.

- Input variables can be continuous or discrete.
- Output:
  - A set of coefficients that indicate the relative impact of each driver.
  - A linear expression for predicting outcome as a function of drivers.


Regression line
error term (e)
b1
b0
y
x
30
25
20
15
10
100
200
300

Technical Description

$$y = b_0 + b_1 x_1 + b_2 x_2 + ...$$

- Solve for the b_i
  - Ordinary Least Squares
    - storage quadratic in number of variables
    - must invert a matrix
- Categorical variables are expanded to a set of indicator variables, one for each possible value.

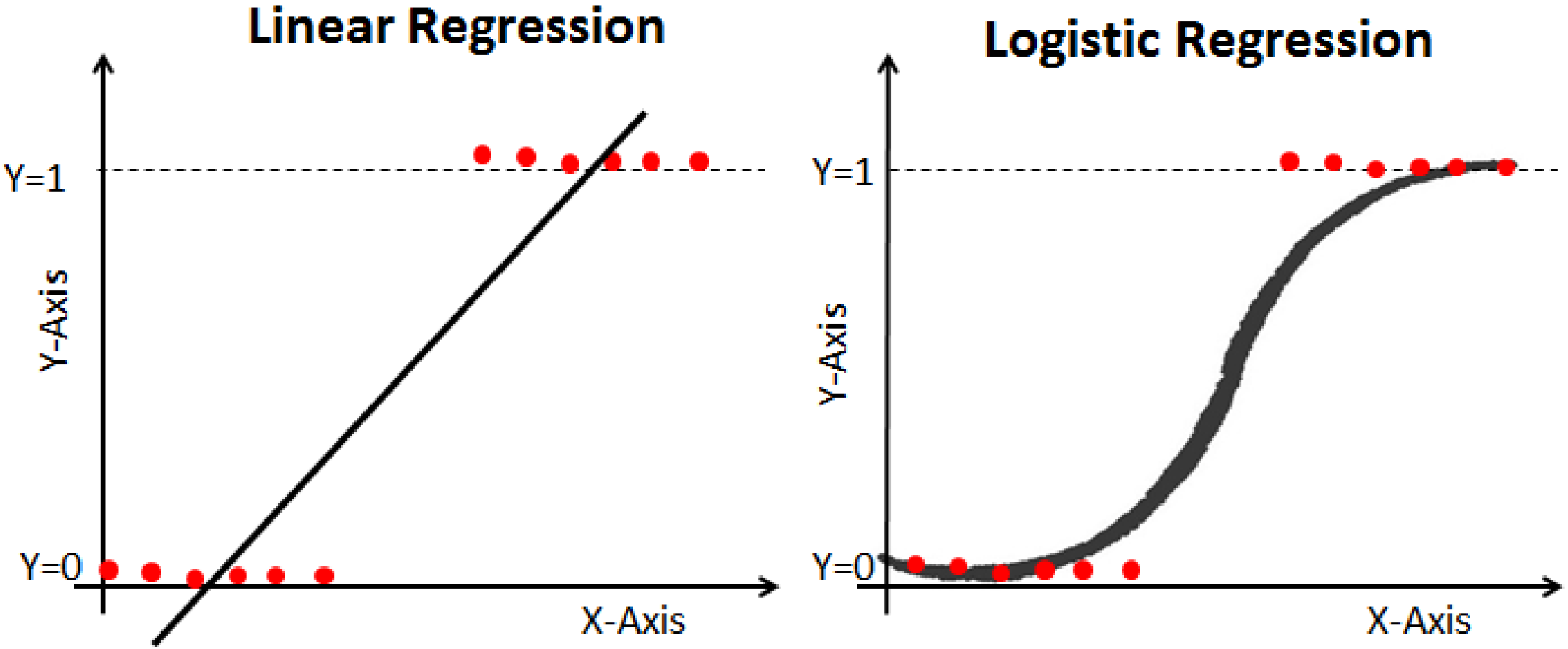
Model Performance

- R – Square (R^2), Formula for calculating R^2 is given by

$$R^2 = \frac{TSS - RSS}{TSS}$$

- Total Sum of Squares (TSS) : TSS is a measure of total variance in the response/ dependent variable Y and can be thought of as the amount of variability inherent in the response before the regression is performed.
- Residual Sum of Squares (RSS) : RSS measures the amount of variability that is left unexplained after performing the regression.

- (TSS – RSS) measures the amount of variability in the response that is explained (or removed) by performing the regression


Linear Regression
Y=1
Y=0
Y-Axis
X-Axis
Logistic Regression
Y=1
Y=0
Y-Axis
X-Axis

Types of Logistic Regression:

- Binary Logistic Regression: The target variable has only two possible outcomes such as Spam or Not Spam, Cancer or No Cancer.

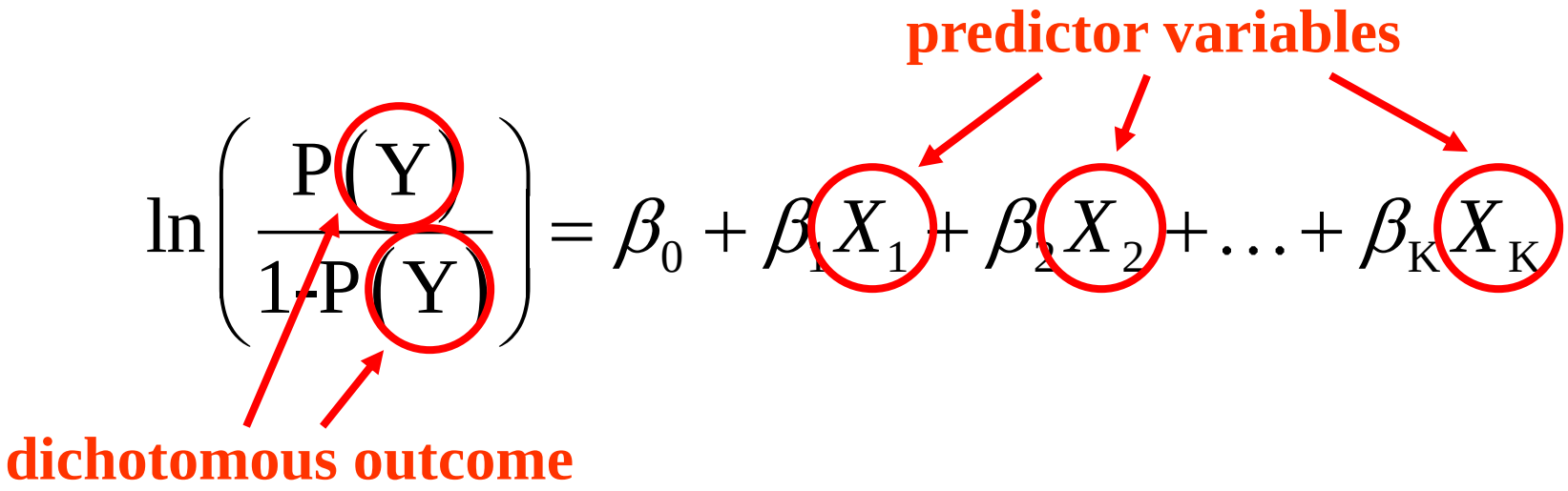
- Multinomial Logistic Regression: The target variable has three or more nominal categories such as predicting the β-turn types: Types I, II, IV and VIII.

- Ordinal Logistic Regression: the target variable has three or more ordinal categories such as product rating from 1 to 5 or outcomes from biomarkers studies; no, mild or severe disease.

Linear Regression - Reasons to Choose (+) and Cautions (-)

Reasons to Choose (+)	Cautions (-)
Concise representation (the coefficients)	Does not handle missing values well
Robust to redundant variables, correlated variables Lose some explanatory value	Assumes that each variable affects the outcome linearly and additively Variable transformations and modeling variable interactions can alleviate this. A good idea to take the log of monetary amounts or any variable with a wide dynamic range
Explanatory value Relative impact of each variable on the outcome	Can't handle variables that affect the outcome in a discontinuous way \Step functions
Easy to score data	Doesn't work well with discrete drivers that have a lot of distinct values, for example, ZIP code

The Logistic Regression Model

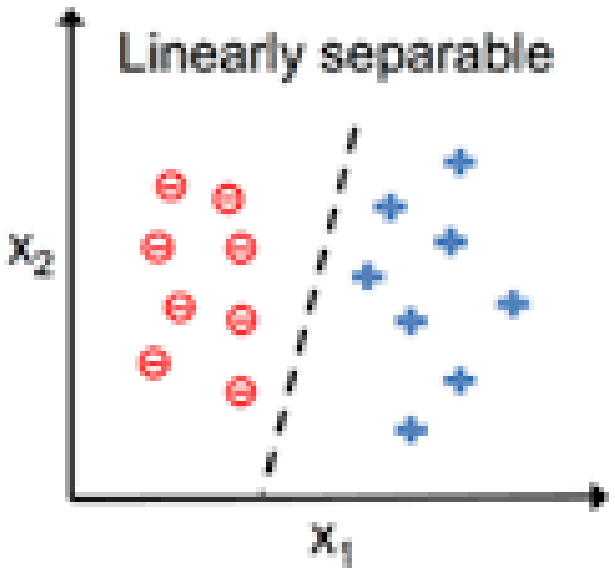


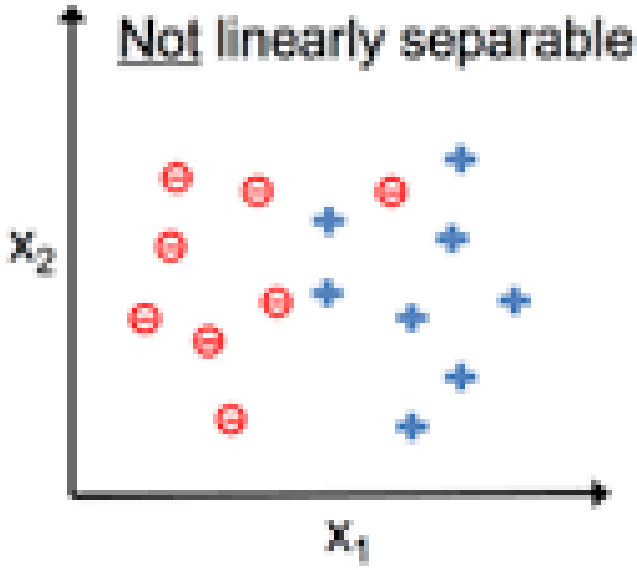
$\ln\left(\frac{P(Y)}{1-P(Y)}\right)$ is the log(odds) of the outcome.

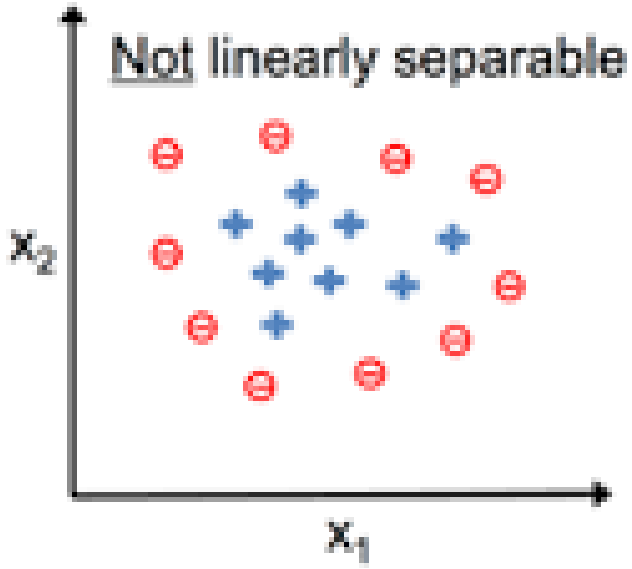
Logistic Regression - Reasons to Choose (+) and Cautions (-)

Reasons to Choose (+)	Cautions (-)
Explanatory value: Relative impact of each variable on the outcome in a more complicated way than linear regression	Does not handle missing values well
Robust with redundant variables, correlated variables Lose some explanatory value	Assumes that each variable affects the log-odds of the outcome linearly and additively Variable transformations and modeling variable interactions can alleviate this A good idea to take the log of monetary amounts or any variable with a wide dynamic range
Concise representation with the the coefficients	Cannot handle variables that affect the outcome in a discontinuous way Step functions
Easy to score data	Doesn't work well with discrete drivers that have a lot of distinct values For example, ZIP code
Returns good probability estimates of an event	
Preserves the summary statistics of the training data "The probabilities equal the counts"	

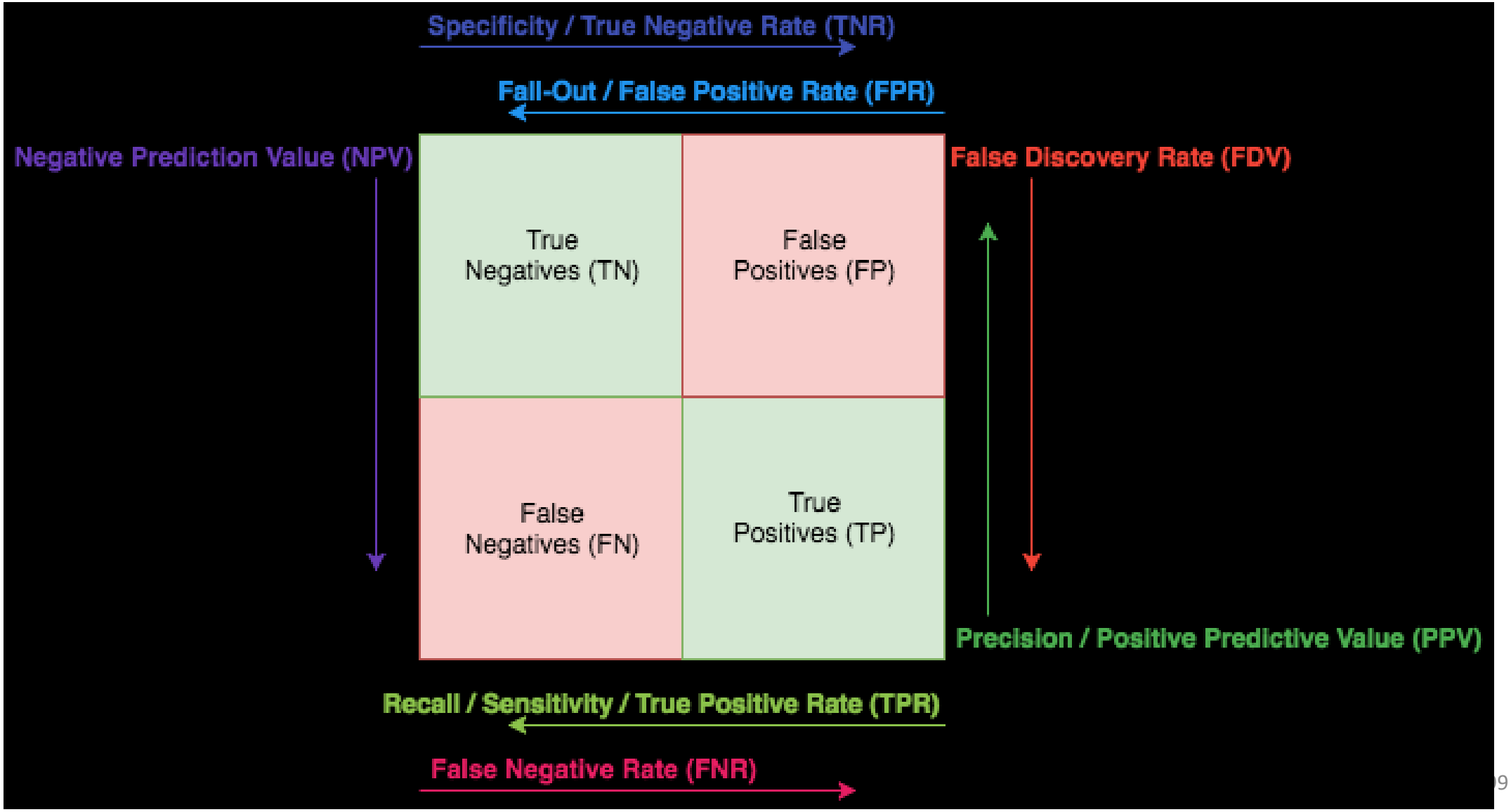
Classification







The Confusion Matrix: How many times the predicted category mapped to the various true categories?



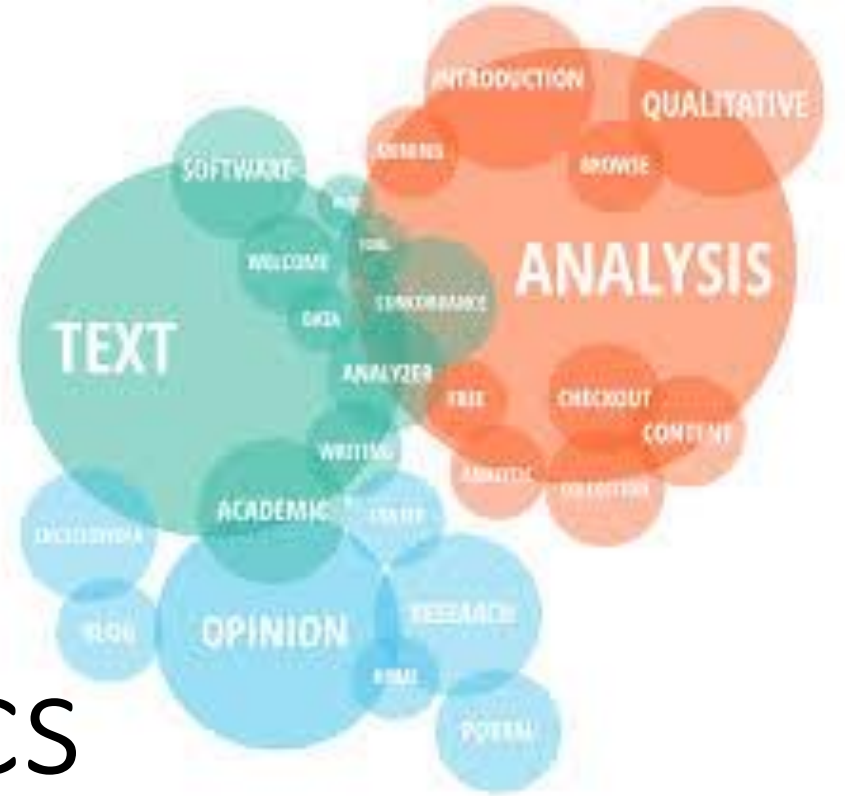
Key Differences Between Classification and Regression

- The *Classification* process models a function through which the data is predicted in discrete class labels. On the other hand, *regression* is the process of creating a model which predict continuous quantity.
- The *classification* algorithms involve decision tree, logistic regression, etc. In contrast, regression tree (e.g., Random forest) and *linear regression* are the examples of regression algorithms.
- *Classification* predicts unordered data while regression predicts ordered data.
- *Regression* can be evaluated using root mean square error. On the contrary, classification is evaluated by measuring accuracy.

Variable selection method	Examples
Supervised variable selection outside predictive models (Figure 3)	Information value Chi-square statistics Gini index
Unsupervised variable selection/extraction outside predictive models	Correlation analysis Cluster analysis Principal component analysis Neural networks
Supervised variable selection inside predictive models	Recursive feature selection: forward, backward and stepwise Regularisation techniques (for example, AIC/BIC, lasso, ridge) Ensemble techniques (for example, random forest and gradient boosting) Cross validation

Machine Learning packages to download:

- caret
- ggplot2
- mlbench
- class
- caTools
- randomForest
- impute
- ranger
- kernlab
- class
- glmnet
- naivebayes
- rpart
- rpart.plot



Social Media Analytics

Techopedia defines Social Analytics thusly:

"Social Media Analytics (SMA) refers to the approach of collecting data from social media sites and blogs and evaluating that data to make business decisions. This process goes beyond the usual monitoring or a basic analysis of retweets or 'likes' to develop an in-depth idea of the social consumer." Note, "social media sites" encompasses not just Facebook, Twitter, and the like, but forums and review sites as well as blogs and news outlets.

Breakdown of some of the key metrics for soci media analysis:

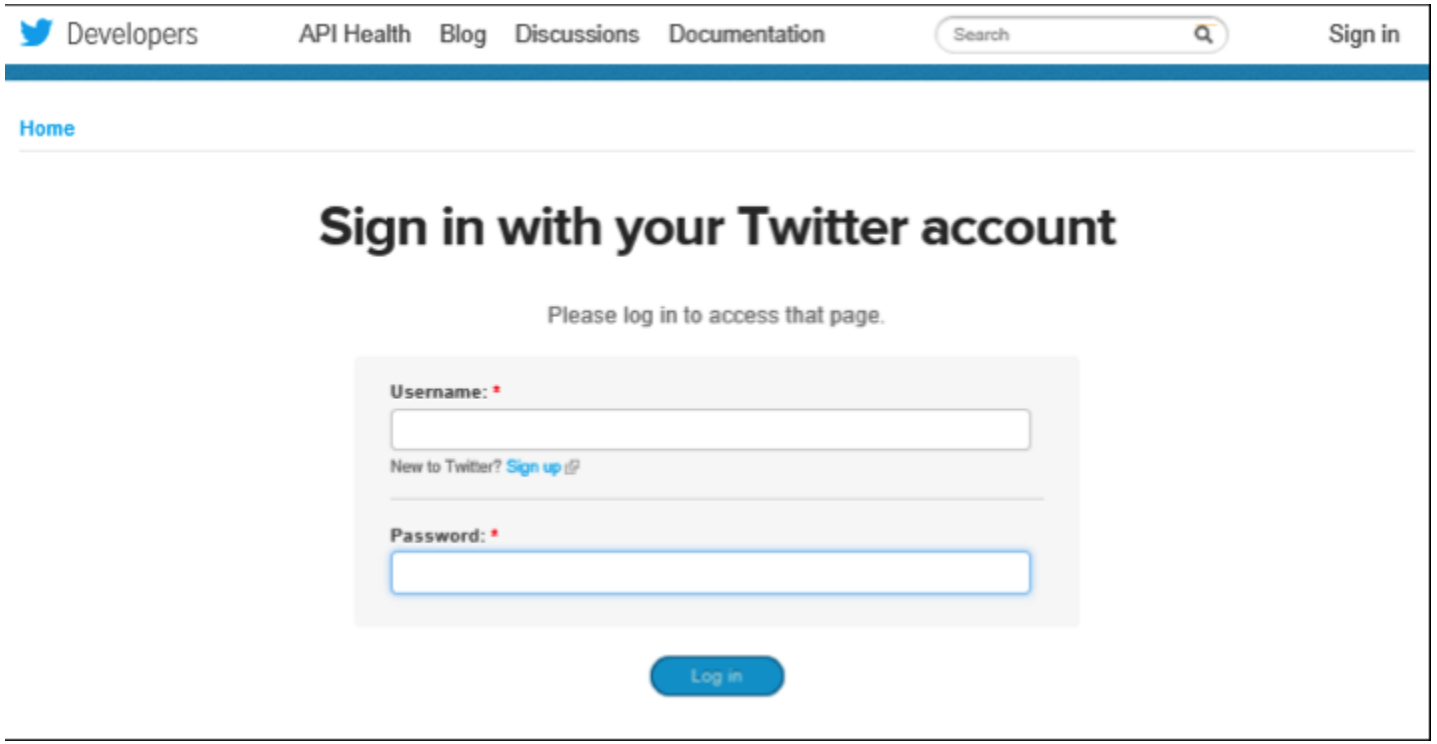
Business objective	Social media goal	Metric(s)
Grow the brand	**Awareness** *(these metrics illuminate your current and potential audience)*	Followers, shares, etc.
Turn customers into advocates	**Engagement** *(these metrics show how audiences are interacting with your content)*	Comments, likes, @mentions, etc.
Drive leads and sales	**Conversions** *(these metrics demonstrate the effectiveness of your social engagement)*	Website clicks, email signups, etc.
Improve customer retention	**Consumer** *(these metrics reflect how active customers think and feel about your brand)*	Testimonials, social media sentiment, etc.

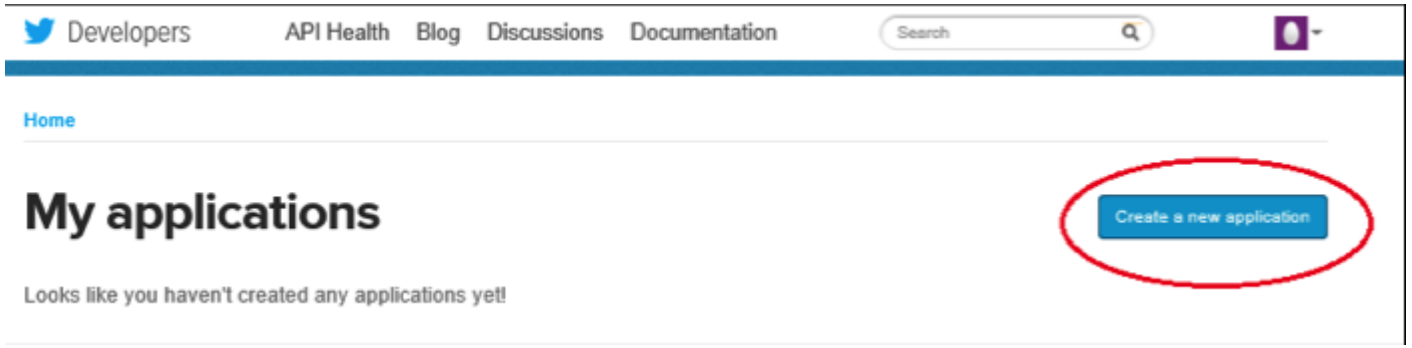
Let’s Discuss:

- How does social media influence real-world behaviors of individuals? Identify a behavior that is due to the usage of, say, Twitter.

- Identify at least three major side effects of information sharing on social media.

- How does behavior of individuals change across sites? What behaviors remain consistent and what behaviors likely change? What are possible reasons behind these differences?





https://www.credera.com/blog/business-intelligence/twitter-analytics-using-r-part-1-extract-tweets/

```
#install packages
library("openssl")
library("httpuv")
library("twitteR")
library("tm")
library("stringr")
library("dplyr")

# Setup Twitter API developer account
# Consumer_key, consume_secret, access_token, and access_secret based on your own keys
api_key <- "fXXXXXXXXXXXXXXXXXXXXXXXXXXXXGJLbXXXXM"
api_secret <- "rXXXXXXXXXXXXXXXXXXXXXXXXXXXXYVjDeGvxEqAm2lg75Ma"
access_token <- "1132454256114XXXXXXXXXXXXXXXXXXXXXXXXBu3XXXXX8t"
access_secret <- "U8cOXXXXXXXXXXXXXXXXXXXXXXXXX7F3hXXXXX6LKj"

setup_twitter_oauth(api_key,api_secret,access_token,access_secret)
origop <- options("httr_oauth_cache")
options(httr_oauth_cache = TRUE)

#Obtaining last 3200 tweets from CNN
tweets <-userTimeline("CNN", n=3200)
tweets.df <- twListToDF(tweets) #convert to data frame
hashtagtweet <- searchTwitter
write.csv(tweets.df, "C:/Users/philliyd/Documents/My Stuff/TextMining/tweetsCNN.csv")

consumer_key <- "“ consumer_secret <-"“  access_token <- "“ access_secret <- ""
setup_twitter_oauth(consumer_key, consumer_secret, access_token, access_secret)
tw = searchTwitter('@(userID)', n = 25)
d = twListToDF(tw)
brady_posts <- getPage(page='TomBrady', token=token, n = 2000, feed = FALSE,reactions = TRUE)
```

Tidbits on tweets

- Note that your tweet needs to be 140 characters or less.
- Search Twitter for Tweets
  - q: the query word that you want to look for
  - n: the number of tweets that you want returned. You can request up to a maximum of 18,000 tweets.

```
rstats_tweets <- search_tweets(q = "#rstats", n = 500)
```


  - To see what other arguments you can use with this function, use the R help:
- Retweet
  - A retweet is when you or someone else shares someone elses tweet so your / their followers can see it. It is similar to sharing in Facebook where you can add a quote or text above the retweet if you want or just share the post.

```
rstats_tweets <- search_tweets("#rstats", n = 500, include_rts = FALSE)
```


Any
Questions

BeVera

Brian Ashford
bashford@beverasolutions.com

Yvonne Phillips
yphillips@beverasolutions.com

BeVera