Text Mining and its Applications with R

For Two weeks FDP on
Hybrid Computational Intelligence (17- 27 July, 2017)

Koushik Mondal
koushik.ismd@gmail.com

July 22, 2017

Before Starting…

The whole of science is nothing more than a refinement of everyday thinking.

Albert Einstein

Outline of the Talk

- ✓ Why Using R?
- ✓ What You'll get?
- ✓ Data Types and Data Objects
- ✓ General Subsetting Rules
- ✓ Important Utilities
- ✓ Reading and Writing External Data
- ✓ Basic Graphics
- ✓ Quantitative and Qualitative Process
- ✓ Tidy data principles
- ✓ Text Mining
- ✓ Example

Why Using R?

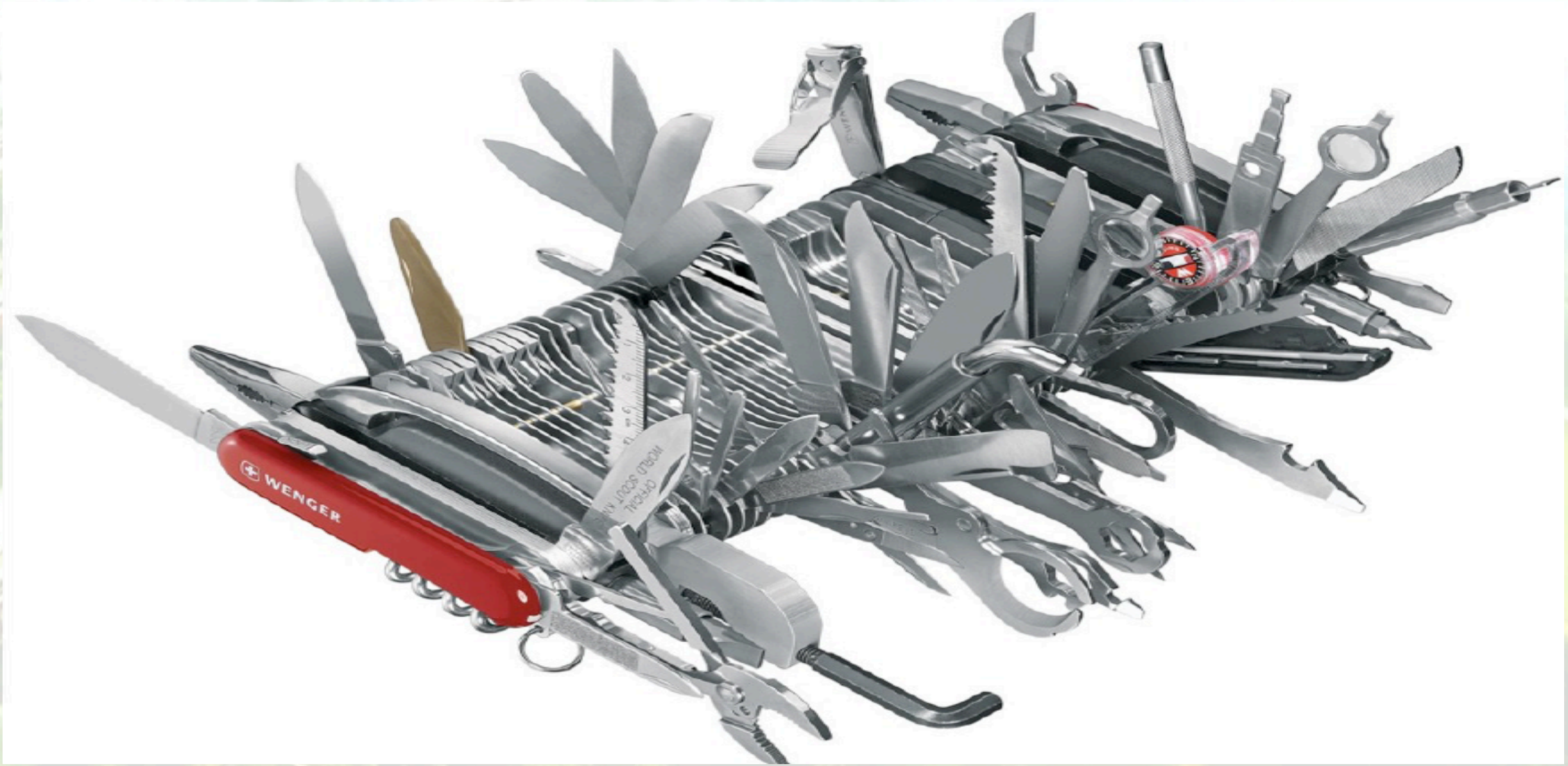
- ✓ Complete statistical environment and programming language
- ✓ Efficient functions and data structures for data analysis
- ✓ Powerful graphics
- ✓ Access to fast growing number of analysis packages
- ✓ Most widely used language in bioinformatics
- ✓ Is standard for data mining and bio-statistical analysis
- ✓ Technical advantages: free, open-source, available for all Oss
- ✓ Google it using R or CRAN (Comprehensive R Archive Network) http://www.r-project.org
- ✓ GUI Interface – RStudio (https://www.rstudio.com/)

Where is R?

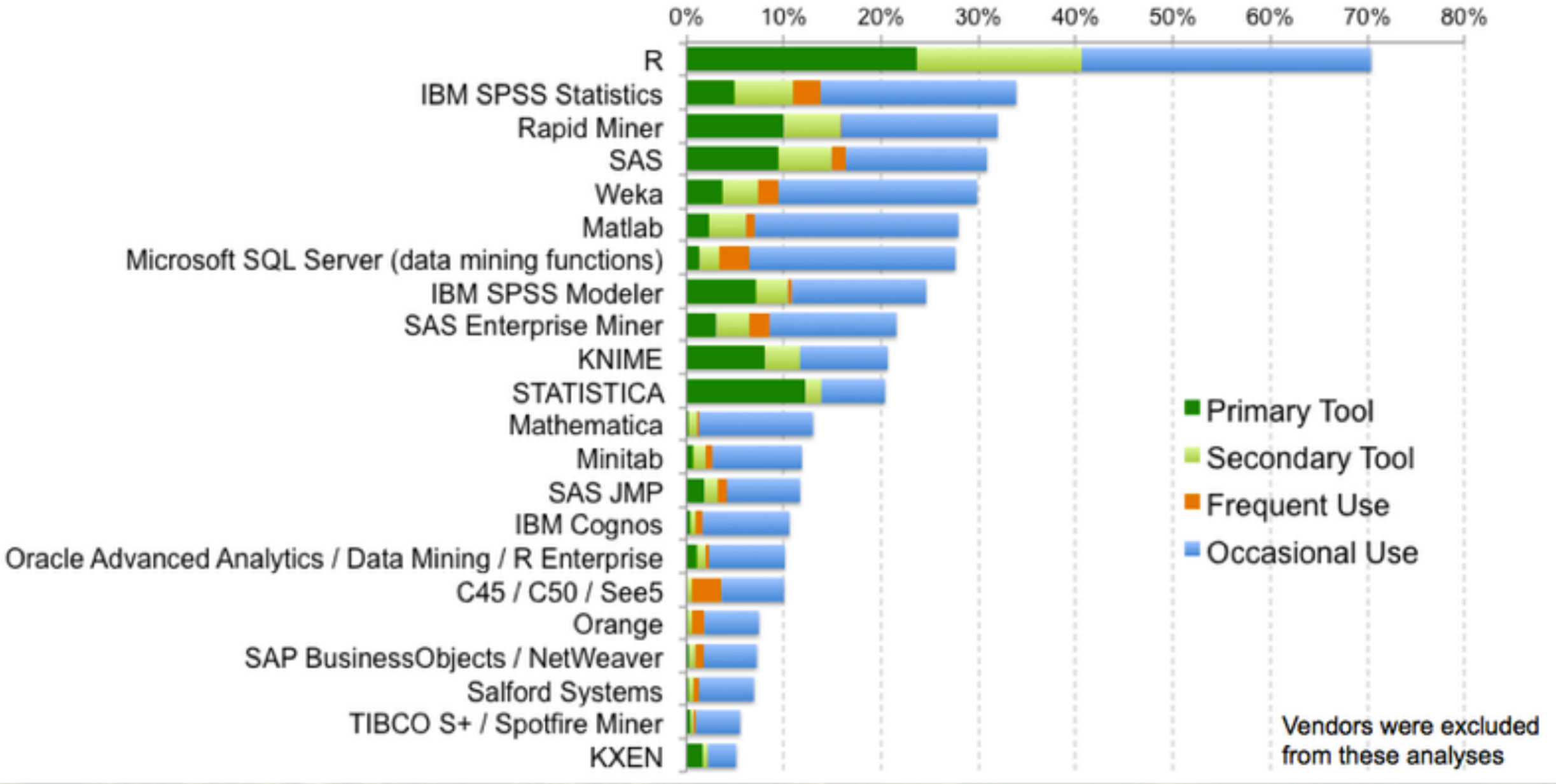
In fact, seldom I hear from a researcher who says s/he has heard about R.

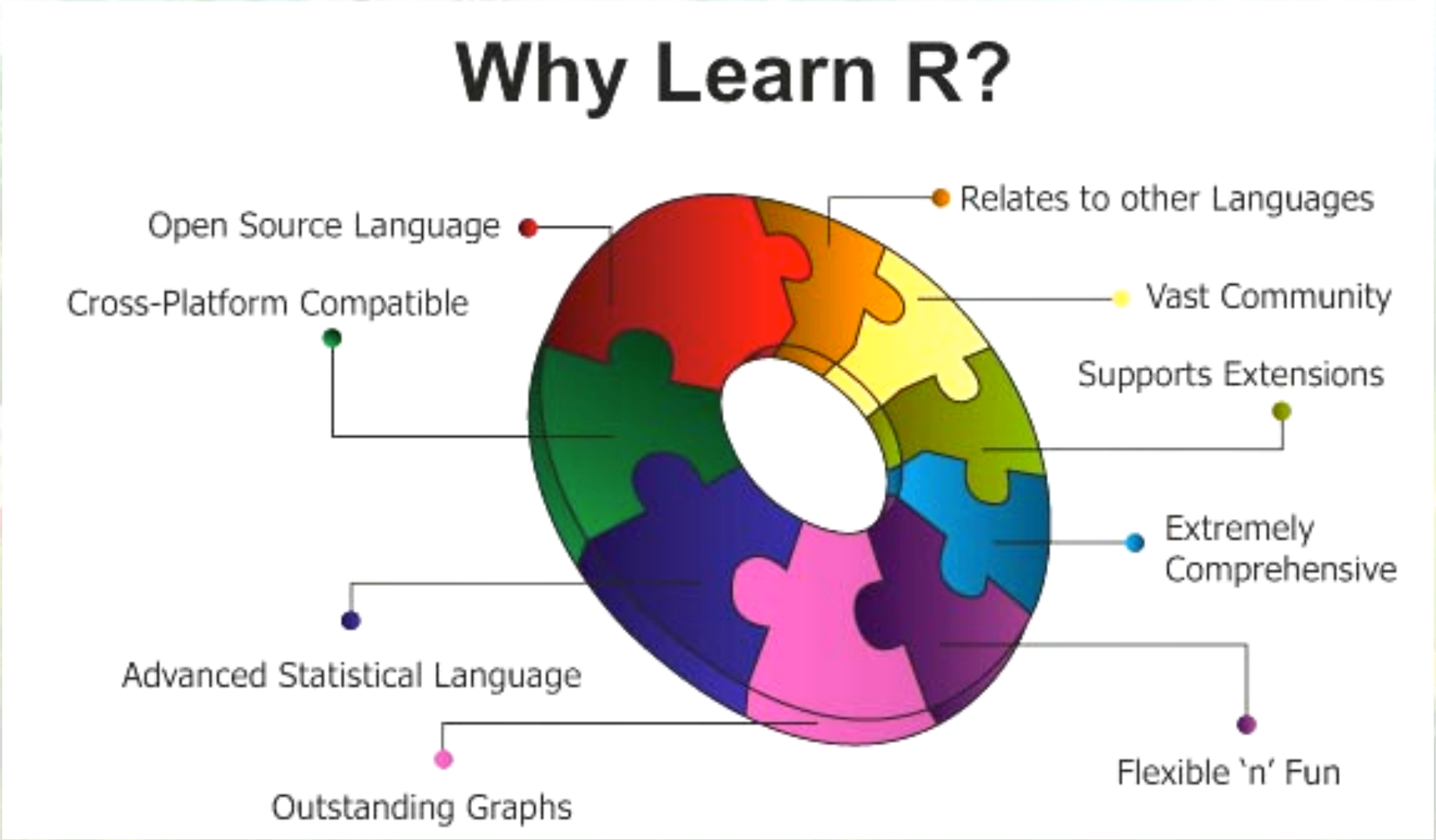
Tools: Why using R?

Tools: Why using R?

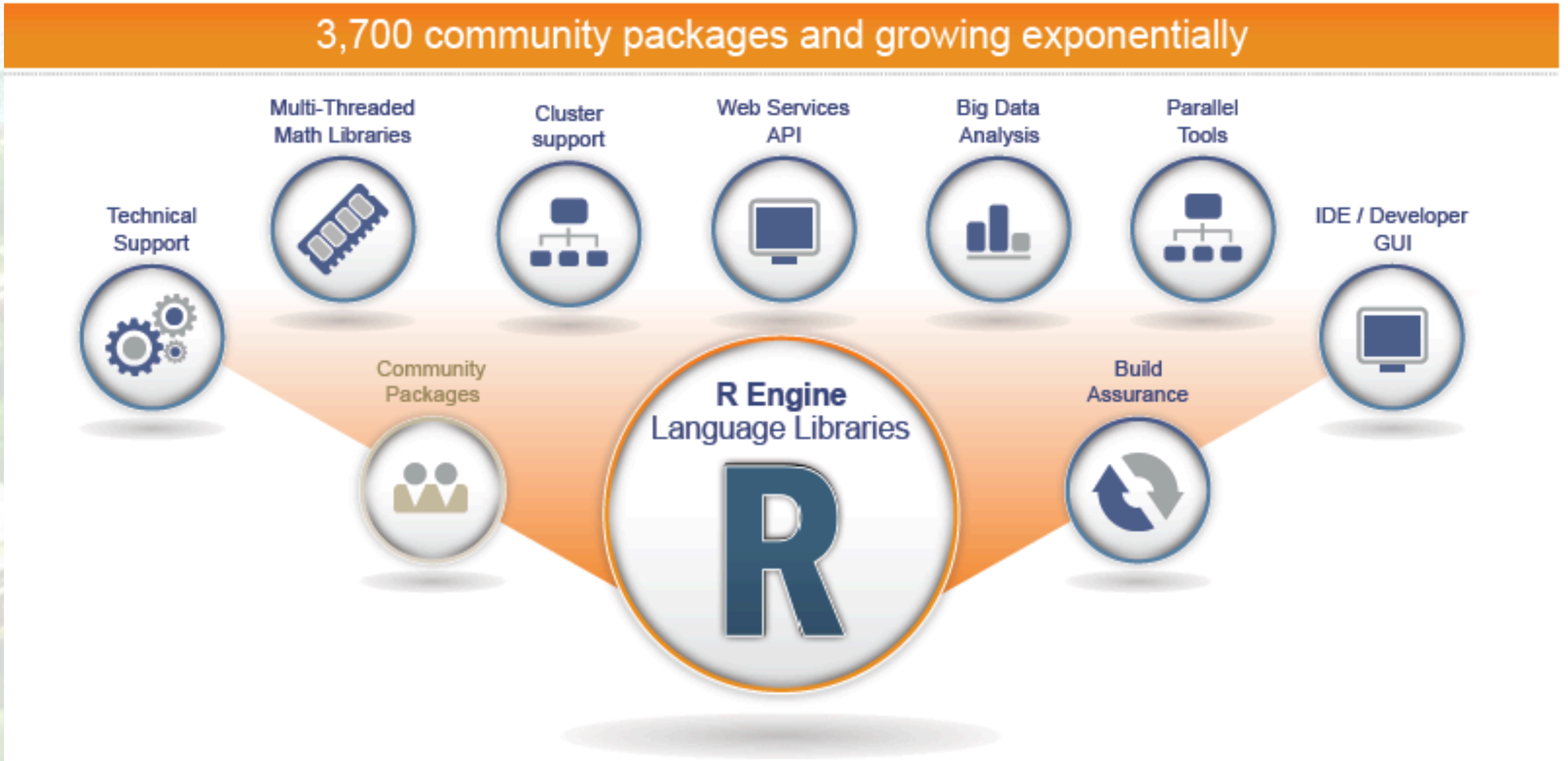


Tools: Why using R?

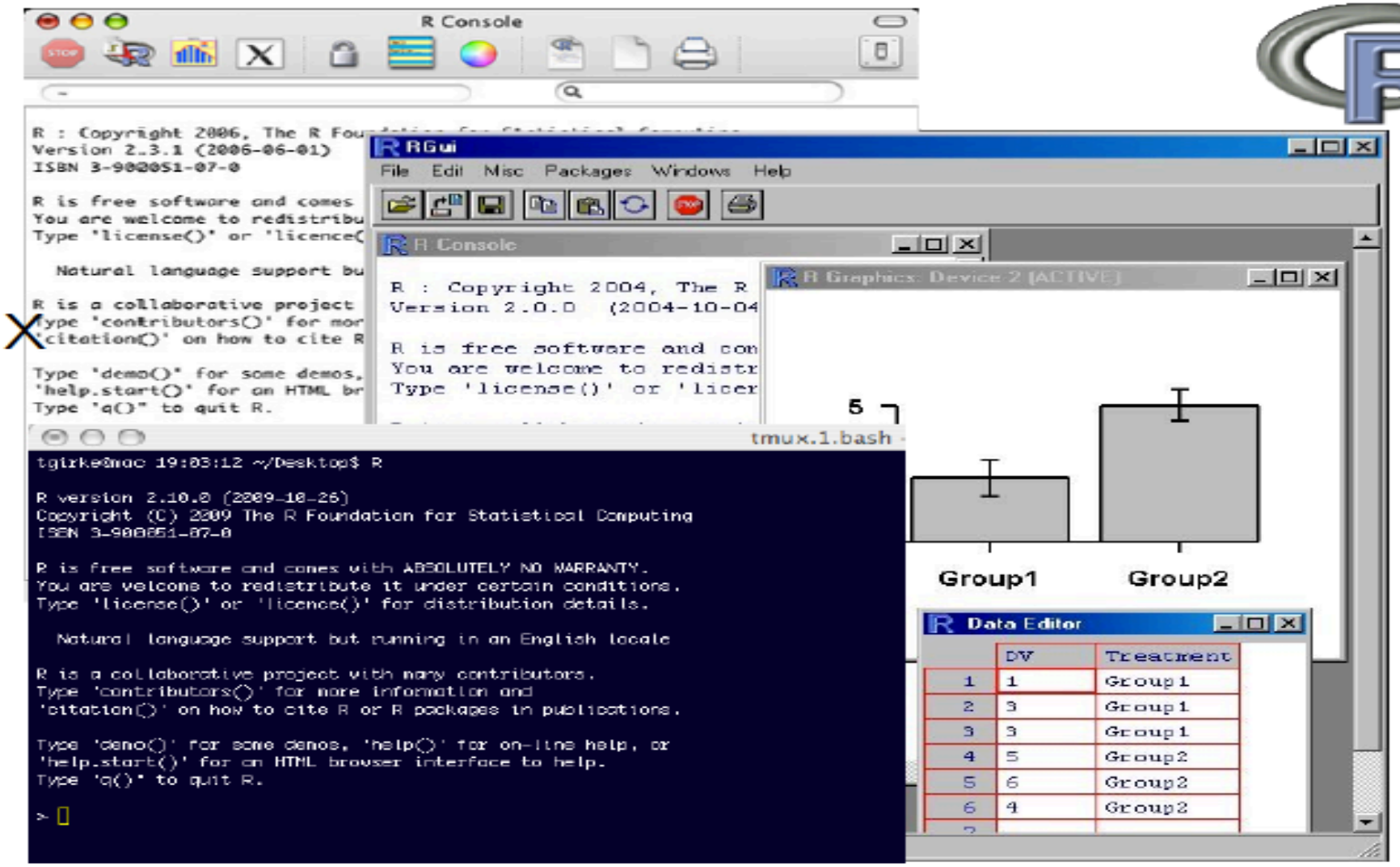



Why Learn R?
Open Source Language
Cross-Platform Compatible
Advanced Statistical Language
Outstanding Graphs
Relates to other Languages
Vast Community
Supports Extensions
Extremely Comprehensive
Flexible 'n' Fun

dim(available.packages())



What You'll Get?



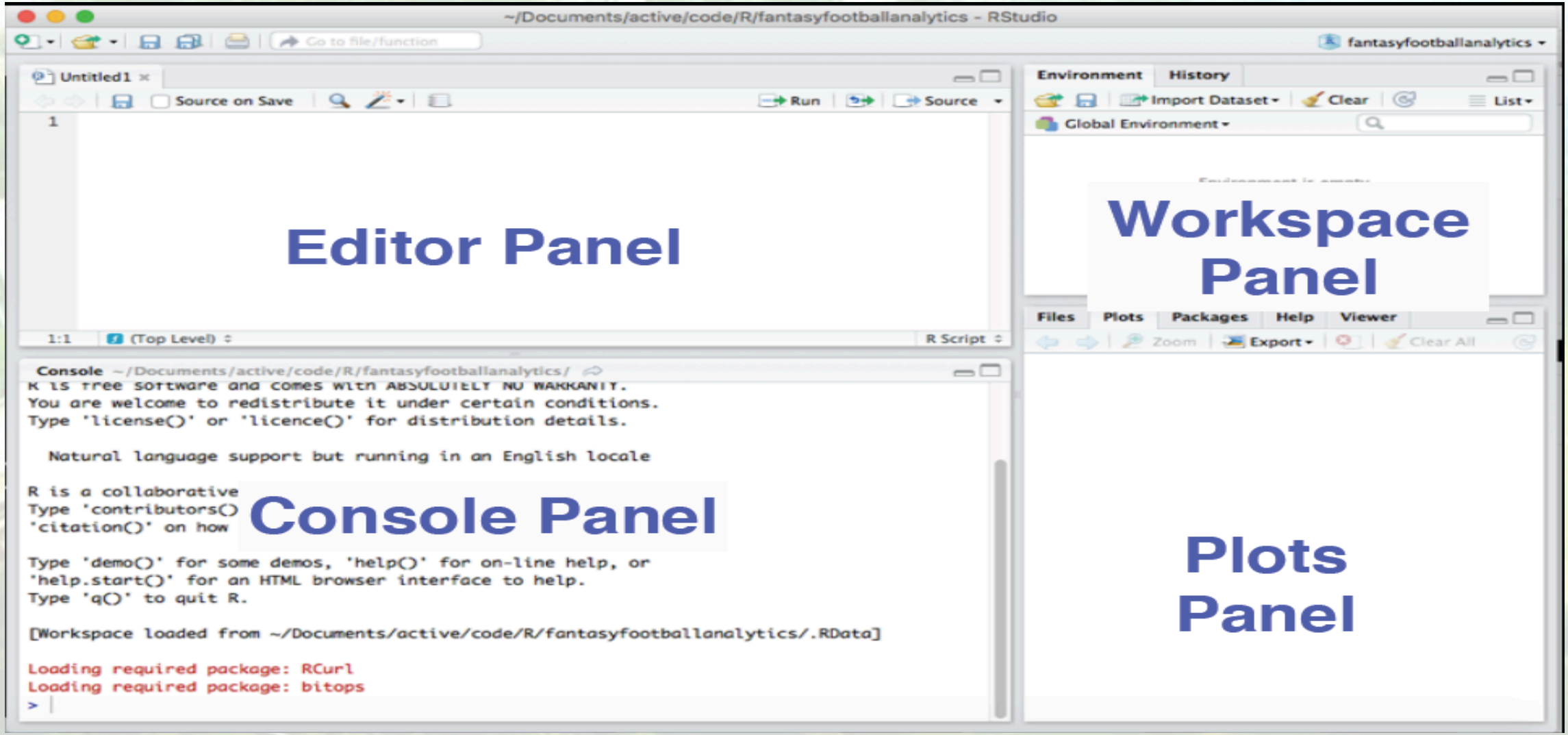
R Gui: OS X

Command-line R: Linux/OS X

R Gui: Windows

Rstudio: Alternative IDE

- ✓ New integrated development environment (IDE) for R that works well for beginners and developers.



Package Repositories

- ✓ CRAN (>4500 packages) general data analysis and mining
- ✓ Bioconductor (>700 packages) bioscience data analysis
- ✓ Omegahat (>30 packages) programming interfaces

Install R and Add-on Packages

Install R for your operating system from:

https://cran.r-project.org

Install RStudio from:

http://www.rstudio.com/ide/download

Installation of CRAN Packages

```
> install.packages(c("pkg1", "pkg2"))
> install.packages("pkg.zip", repos=NULL)
```

Installation of Bioconductor Packages

```
> source("http://www.bioconductor.org/biocLite.R")
> biocLite()
> biocLite(c("pkg1", "pkg2"))
```

Tutorials

Each of the following tutorials are in PDF format:

- ✓ P. Kuhnert & B. Venables, **An Introduction to R: Software for Statistical Modeling & Computing**
- ✓ J.H. Maindonald, **Using R for Data Analysis and Graphics**
- ✓ B. Muenchen, **R for SAS and SPSS Users**
- ✓ W.J. Owen, **The R Guide**
- ✓ D. Rossiter, **Introduction to the R Project for Statistical Computing for Use at the ITC**
- ✓ W.N. Venebles & D. M. Smith, **An Introduction to R**
- ✓ John Verzani, **simpleR - Using R for Introductory Statistics**

Startup and Closing Behavior

Starting R

The R GUI versions, including RStudio, under Windows and Mac OS X can be opened by double-clicking their icons. Alternatively, one can start it by typing 'R' in a terminal (default under Linux).

Startup/Closing Behavior

The R environment is controlled by hidden files in the startup directory:
.RData, .Rhistory and .Rprole (optional).
Closing R
> q()
Save workspace image? [y/n/c]:

Note

When responding with 'y', then the entire R workspace will be written to the .RData file which can become very large. Often it is sufficient to just save an analysis protocol in an R source le. This way one can quickly regenerate all data sets and objects.

Data Types

Numeric data: 1, 2, 3

```
> x <- c(1, 2, 3); x
[1] 1 2 3
> is.numeric(x)
[1] TRUE
> as.character(x)
[1] "1" "2" "3"
```

Character data: "a", "b", "c"

```
> x <- c("1", "2", "3"); x
[1] "1" "2" "3"
> is.character(x)
[1] TRUE
> as.numeric(x)
[1] 1 2 3
```

Complex data

```
> c(1, "b", 3)
[1] "1" "b" "3"
```

Logical data

```
> x <- 1:10 < 5
> x
[1] TRUE TRUE TRUE TRUE FALSE FALSE FALSE FALSE
FALSE FALSE
> !x
[1] FALSE FALSE FALSE FALSE TRUE TRUE TRUE TRUE
TRUE TRUE
> which(x) # Returns index for the 'TRUE' values in logical vector
[1] 1 2 3 4
```

Data Objects: Vectors and Factors

Vectors (1D)

```
> myVec <- 1:10; names(myVec) <- letters[1:10]
> myVec[1:5]
a b c d e
1 2 3 4 5
> myVec[c(2,4,6,8)]
b d f h
2 4 6 8
> myVec[c("b", "d", "f")]
b d f
2 4 6
```

Factors (1D): vectors with grouping information

```
> factor(c("dog", "cat", "mouse", "dog", "dog", "cat"))
[1] dog cat mouse dog dog cat
Levels: cat dog mouse
```

Data Objects: Matrices, Data Frames and Arrays

Matrices (2D): two dimensional structures with data of same type

```
> myMAT <- matrix(1:30, 3, 10, byrow = TRUE)
> class(myMAT)
[1] "matrix"
> myMAT[1:2,]
[,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,] 1 2 3 4 5 6 7 8 9 10
[2,] 11 12 13 14 15 16 17 18 19 20
```

Data Frames (2D): two dimensional structures with variable data types

```
> myDF <- data.frame(Col1=1:10, Col2=10:1)
> myDF[1:2, ]
Col1 Col2
1 1 10
2 2 9
```

Arrays: data structure with one, two or more dimensions

Data Objects: Lists and Functions

Lists: containers for any object type

```
> myL <- list(name="Hum", wife="Tum", no.children=3, child.ages=c(4,7,9))
> myL
$name
[1] "Hum"
$wife
[1] "Tum"
$no.children
[1] 3
$child.ages
[1] 4 7 9
> myL[[4]][1:2]
[1] 4 7
```

Functions: piece of code

```
> myfct <- function(arg1, arg2, ...) {
+ function_body
+ }
```

General Subsetting Rules

Subsetting by positive or negative index/position numbers

```
> myVec <- 1:26; names(myVec) <- LETTERS
> myVec[1:4]
A B C D
1 2 3 4
```

Subsetting by same length logical vectors

```
> myLog <- myVec > 10
> myVec[myLog]
K L M N O P Q R S T U V W X Y Z
11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
```

Subsetting by field names

```
> myVec[c("B", "K", "M")]
B K M
2 11 13
```

Calling a single column or list component by its name with the $ sign

```
> iris$Species[1:8]
[1] setosa setosa setosa setosa setosa setosa setosa setosa
Levels: setosa versicolor virginica
```

Combining Objects

The c function combines vectors and lists

```
> c(1, 2, 3)
[1] 1 2 3
> x <- 1:3; y <- 101:103
> c(x, y)
[1] 1 2 3 101 102 103
```

The cbind and rbind functions can be used to append columns and rows, respecively

```
> ma <- cbind(x, y)
> ma
x y
[1,] 1 101
…
> rbind(ma, ma)
x y
[1,] 1 101
[2,] 2 102
…
```

Accessing Name Slots and Dimensions of Objects

Length and dimension information of objects

```
> length(iris$Species)
[1] 150
> dim(iris)
[1] 150 5
```

Accessing row and column names of 2D objects

```
> rownames(iris)[1:8]
[1] "1" "2" "3" "4" "5" "6" "7" "8"
> colnames(iris)
[1] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width" "Species"
```

Return name field of vectors and lists

```
> names(myVec)
[1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S”
 > names(myL)
[1] "name" "wife" "no.children" "child.ages"
```

Sorting Objects

The function sort returns a vector in ascending or descending order

```
> sort(10:1)
[1] 1 2 3 4 5 6 7 8 9 10
```

The function order returns a sorting index for sorting an object

```
> sortindex <- order(iris[,1], decreasing = FALSE)
> sortindex[1:12]
[1] 14 9 39 43 42 4 7 23 48 3 30 12
> iris[sortindex,][1:2,]
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
14          4.3         3.0          1.1         0.1  setosa
9           4.4         2.9          1.4         0.2  setosa
> sortindex <- order(-iris[,1]) # Same as decreasing=TRUE
```

Sorting on multiple columns

```
> iris[order(iris$Sepal.Length, iris$Sepal.Width),][1:2,]
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
14          4.3         3.0          1.1         0.1  setosa
9           4.4         2.9          1.4         0.2  setosa
```

Reading and Writing External Data

Import data from tabular files into R

```
> myDF <- read.delim("myData.xls", sep="\t")
```

Export data from R to tabular files

```
> write.table(myDF, file="myfile.xls", sep="\t", quote=FALSE, col.names=NA)
```

Copy and paste (e.g. from Excel) into R

```
## On Windows/Linux systems:
> read.delim("clipboard")
## On Mac OS X systems:
> read.delim(pipe("pbpaste"))
```

Copy and paste from R into Excel or other programs

```
## On Windows/Linux systems:
> write.table(iris, "clipboard", sep="\t", col.names=NA, quote=F)
## On Mac OS X systems:
> zz <- pipe('pbcopy', 'w')
> write.table(iris, zz, sep="\t", col.names=NA, quote=F)
> close(zz)
```

Basic Graphics

Important high-level plotting functions

plot: generic x-y plotting

barplot: bar plots

boxplot: box-and-whisker plot

hist: histograms

pie: pie charts

dotchart: cleveland dot plots

image, heatmap, contour, persp: functions to generate image-like plots

qqnorm, qqline, qqplot: distribution comparison plots

pairs, coplot: display of multivariant data

Help on these functions

?myfct

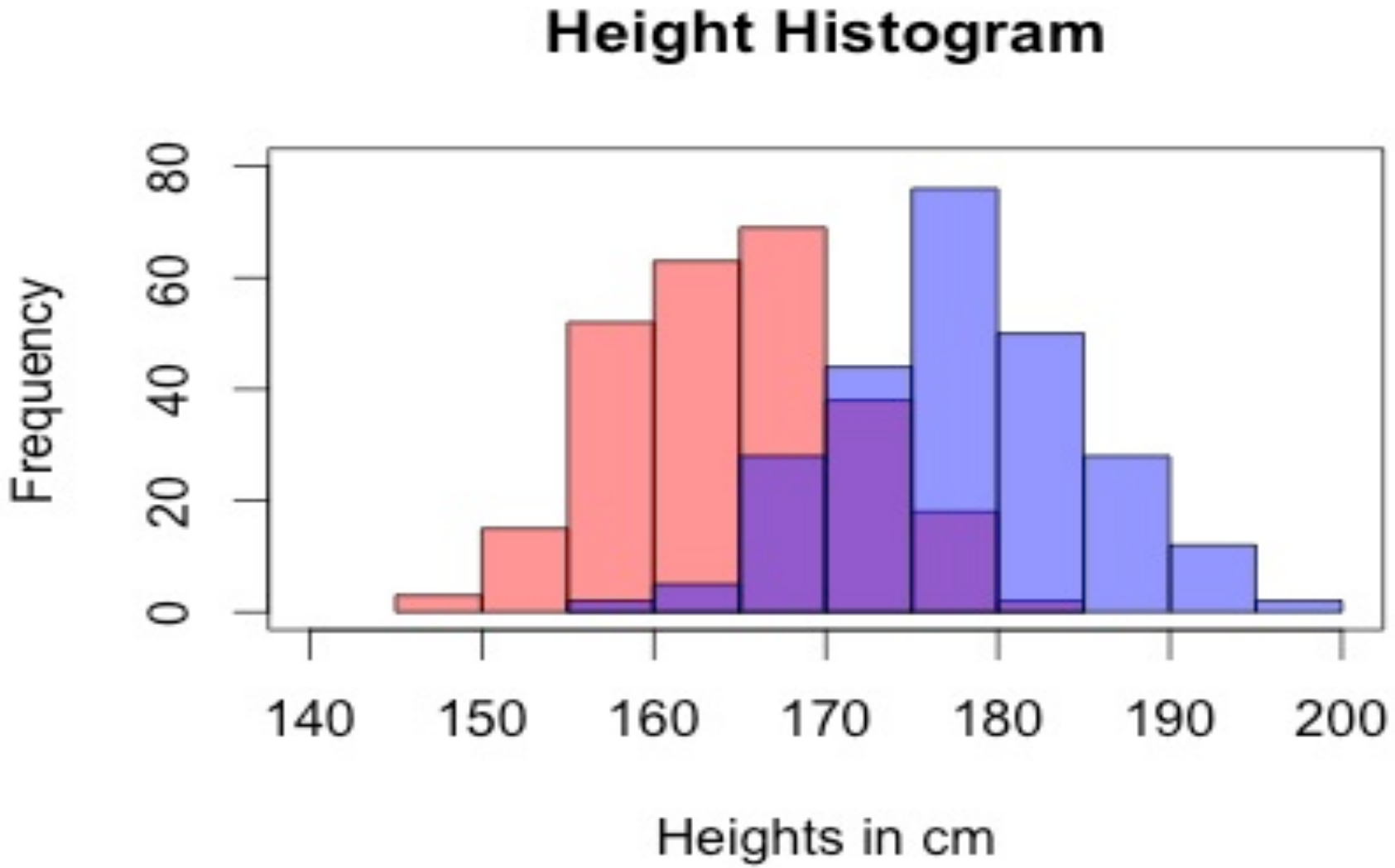
?plot

?par

Example:

```
setwd("~/Desktop/Mylectures/")

load("bdims.RData")

mdims <- subset(bdims, sex == 1)

fdims <- subset(bdims, sex == 0)

hist(fdims$hgt, main="Height Histogram", xlab="Heights in cm",col=rgb(1,0,0,0.5),xlim=c(140,200),ylim=c(0,80))

hist(mdims$hgt, main="Female Height Histogram", col=rgb(0,0,1,0.5),add=T)

box()
```

Output


Height Histogram
Frequency
0
20
40
60
80
140
150
160
170
180
190
200
Heights in cm

Building better search tools

✓ The example here is building better search tools

- Search for “mining data”, “data mining” and “data about mining”
- Imagine an e-store selling products: during search in catalog we may received 20% irrelevant product items
- Now we have problem with us and we need to describe the same to our “Big Data Team” for workable solution after easily fix such problem

- ✓ Let us have a close look the internal process

i. The user query is cleaned: typos are removed.

ii. It is then stemmed: plural is replaced by singular, verbs (ing form) are replaced by nouns (thus mining becomes mine), and so on.

iii. Stop words are removed: the, or, and, of, from, about, it, with, and so on.

For instance, IT jobs becomes jobs, and data about mining becomes data mining and (because of step 2) data mine.

iv. The query is normalized: the tokens are rearranged in alphabetical order (mine data becomes data mine).

v. Algorithms are used to determine if the query contains the name of a city, book, song, product, or movie, using lookup tables.

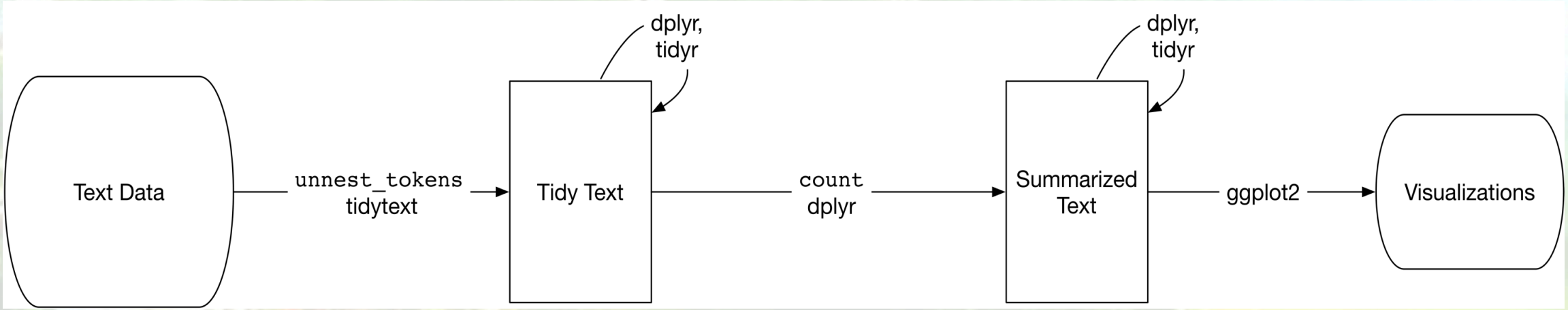
vi. Other algorithms are used to detect whether two tokens must be attached.
For example, New Delhi is one token, not two, even though it looks like two.

vii. Synonyms are detected. For example, perhaps automobile can be replaced by car.

viii. An algorithm is used to detect user intention (sometimes based on a user profile) for queries spanning multiple categories.

- ✓ Quantitative data, also known as continuous data, consists of numeric data that support arithmetic operations.
- ✓ This is in contrast with qualitative data, whose values belong to pre-defined classes with no arithmetic operation allowed.
- ✓ Here we are interested in text mining.
- ✓ Mining text is generally offered both qualitative and quantitative data analysis opportunities but today we will discuss about quantitative data analysis feature.
- ✓ For handling text data, it is easier to follow tidy data principles for effective handling and processing of the datasets.

✓ The flowchart for text mining is as under:



- ✓ Tidy data has a specific structure with following rules:
  - Each variable is a column
  - Each observation is a row
  - Each type of observational unit is a table
- ✓ It is popularly described as "A table with one-token-per-row"
- ✓ A token is a meaningful unit of text, such as a word, that we are interested in using for analysis, and tokenization is the process of splitting text into tokens.
- ✓ String: Text can, of course, be stored as strings, i.e., character vectors, within R, and often text data is first read into memory in this form.
- ✓ Corpus: These types of objects typically contain raw strings annotated with additional metadata and details.
- ✓ Document-term matrix: This is a sparse matrix describing a collection (i.e., a corpus) of documents with one row for each document and one column for each term. The value in the matrix is typically used for word count or similar purposes

- ✓ An n-gram is a contiguous sequence of n items from a given sequence of text or speech. The items can be phonemes, syllables, letters, words or base pairs according to the application. The n-grams typically are collected from a text or speech corpus.

- ✓ Tidy data sets allow manipulation with a standard set of "tidy" tools, including popular packages such as dplyr (Wickham and Francois 2016), tidyr (Wickham 2016), ggplot2 (Wickham 2009), and broom (Robinson 2017)

- ✓ R packages such as tm (Ingo Feinerer and Meyer 2008) and quanteda (Benoit and Nulty 2016) allow, for example, a workflow where importing, filtering, and processing is done using dplyr and other tidy tools, after which the data is converted into a document-term matrix for machine learning applications. The models can then be re-converted into a tidy form for interpretation and visualization with ggplot2.

Any help for you to do further research?

- ✓ After adopted any coding, we proceed to ask whether

(a) code can be usefully modeled by statistical language models and
(b) such models can be leveraged to support software engineers.

Using the widely adopted n-gram model or tidy() object of tm package in R , we provide empirical evidence supportive of a positive answer to both these questions.

Our main aim is to scale the same for Map Reduce architecture.

Not the end… Journey Begins

Perfection is not attainable. But if we chase perfection, we can catch excellence.

Vince Lombardi

Next Session

Practical

gemkousk@gmail.com