INTERNATIONAL JOURNAL FOR ADVANCE RESEARCH IN ENGINEERING AND TECHNOLOGY
WINGS TO YOUR THOUGHTS.....

Deploying R programming language for Big Data Analytics

Rohan Manoj Thakkar[1]

[1]Student of Computer Engineering Department
[1]K. J. Somaiya College of Engineering
Mumbai, Pin no.400077
[1]rohan.m.thakkar@gmail.com

Abstract: *The new trend of deploying techniques which use abundant amount of data is leading to a growing interest in the concept of `Big Data.' Since the arrival of such Big Data quests start right away during data acquisition, it becomes essential to know how to store what is kept with the right metadata. Much of the data today is wrongfully in a disorganized format. With the increasing need of a systematic data framework, Data Science has become the need of the hour. Data Science is basically an intersection of hacking skills, substantive expertise, and math and statistics knowledge. R programming language will be used for effective data analysis. Tools required are git, R, GitHub, RStudio, version control and markdown.*

Keywords: *Data Science, R programming, Data analytics, Big Data, Statistical Data Science.*

1. INTRODUCTION

Over the last several years, data has become cheaper to collect and it is much easier to store. There're also many free computing tools available online which could be used to deal with the entire data deluge that is assaulting all different areas of science and business.

The term 'Big Data' isn't something unfamiliar to data scientists, software engineers and other professionals in the corporate sector. Big data is a new frontier in the sense that we have data in areas that we didn't used to have that data. For example, we didn't have access to information about GPS – coordinates from vehicles from everyone in the entire world. It wasn't possible to sequence everybody's genome. And now that's all possible since we have access to this data and it allows us to answer questions we never could before. So, it's an incredibly exciting time and one can get in there and use that data to answer those questions [10].

Statistics is the science of learning from data. It's very rare to get a data set where all of the answers are really clear, and there's no uncertainty. In any case where there is uncertainty, that's where statistics comes and plays a role. The current time is like the moment that Jeff Bezos discovered the internet. He got into building an internet company at the time when there was an explosive growth in internet usage and it opened the door for the opportunity to build something huge and wonderful [5]. That's what the time is right now for data.

Recently, there has been an explosive growth of data in every possible area. And so it's the opportunity right now to jump on a rocket and, and find out something interesting and carry it off into a major endeavor. Another plus point is that the tools, competitions and websites have all been developed around the idea of helping to learn about data but also getting involved in projects that have super high profile results. So, one example is the Heritage Health Prize [8].

Figure 1

As seen in figure 1, the Heritage Health Prize was a $3 million contest for people who could analyze data and come up with a better predictor of who would be admitted to a hospital in another year. That is a huge amount of money that's being invested in these ideas of algorithm development and data science of prediction.

This paper focuses almost exclusively on the use of the R programming language. That's because it's increasingly the most commonly used language for data science. There are other languages which are also good complements to the R programming language, like Python. This paper however focuses on R because it has a broad range of packages that allow one to go from the rawest of raw files, all the way to interactive reports and documents and web applications that one can share with collaborators. Also, R is free and it has a comprehensive set of

packages for all the processes which are involved in data science. It has one of the best development environments of any programming language. And so it's really possible to learn about the cutting edge of packages that are being developed. It is also very easy to install and play nicely together, which is a feature that doesn't always happen in a lot of the languages that are used for data science [9].

To begin with, it is necessary to understand the conceptual introduction to the concepts related to transforming data into knowledge. After getting the necessary tools from online sources, the further steps should be reviewed.

2. LITERATURE REVIEW AND IMPLEMENTATION SCHEMA

To use R for effective data analysis, necessary steps are installing and configuring the software necessary for statistical programming, discussing the basic concepts of generic programming language in reference to their implementation in a high-profile statistical language. Certain aspects in statistical computing are programming in R, reading the data into R, accessing the R packages and writing the R functions [2].

2.1 Data types and basic operations

Some basic types of data types are:

✓ Objects

R has five basic or 'atomic' classes of objects:

1. Logical (True/False)
2. Character
3. Complex
4. Integer
5. Numeric (Real numbers)

Vector is a very basic object. It can contain objects of only the same class. But one exception to this rule is of a list, which is presented as a vector but can have objects of different classes. The vector() function is used to create empty vectors.

✓ Numbers

Numbers here are considered numeric objects. To explicitly address an integer, it is vital to specify the L suffix. For example, Entering 2 gives a numeric object; whereas entering 2L explicitly gives an integer.

Special number 'Inf' represents infinity.

For example, 2/0; Inf can be used in calculations.

For example, 2/Inf is 0.

The value NaN stands for an undefined value.

For example, 0/0.

Attributes

R objects can have attributes like

1. Names, dimnames
2. Dimensions (For example, matrices and arrays)
3. Class
4. Length
5. Other user-defined attributes/metadata

The attributes () function should be used to access attributes of an object.

✓ Entering Input

Expressions ought to be typed at the R prompt. The <- symbol is the assignment operator.

```
➢ y <- 20
➢ print(y)
  [1] 20
➢ y
  [1] 20
➢ msg <- "bye"
```

The semantics of language are the deciding factors behind the completion of an expression.

```
➢ x <-                ## Incomplete expression
```

The '#' character implies that it is a comment. Any character that comes on the right hand side of the # (including #) is dumped.

✓ Evaluation

When an expression is entered at prompt, the expression is checked for completion and then, it is evaluated. The evaluated expression's result is returned. It may be auto-printed. Types of printing are:

```
➢ x <- 5              ## no
➢ x                   ## auto
  [1] 5
➢ print(x)            ## explicit
  [1] 5
```

'[1]' indicates that x is a vector whereas 5 is the first element.

✓ Printing

```
➢ y <- 1:17
➢ y
  [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
  [16] 16 17
```

As seen above, ':' operator is used to create sequences of entries in the integer format.

✓ Creating vectors

The function denoted by c() is used to create objects' vectors.

```
➢ x <- c(0.5, 0.6)          ## numeric
➢ x <- c(TRUE, FALSE)       ## logical
➢ x <- c(T, F) ## logical
➢ x <- c("a", "b", "c")     ## character
➢ x <- 9:29 ## integer
➢ x <- c(1+0i, 2+4i)        ## complex
```

INTERNATIONAL JOURNAL FOR ADVANCE RESEARCH IN ENGINEERING AND TECHNOLOGY

WINGS TO YOUR THOUGHTS.....

Using the vector() function

- x <- vector("numeric", length = 10)
- x
  [1] 0 0 0 0 0 0 0 0 0 0

Mixing Objects

- y <- c(1.7, "a") ## character
- y <- c(TRUE, 2) ## numeric
- y <- c("a", TRUE) ## character

When different objects are mixed in a vector, coercion does occur in a way such that each element in the vector is attributed to a single class.

2.2 Subsetting

There are many operators which can be used to retrieve subsets of objects. For example,

- The '[' operator gives an object of same class as original. It can also be used to tick two or more elements.
- Next, the '[[' operator can be used to retrieve elements from list or data frame. It can be used only to retrieve one element and class of returned object shall not be necessarily a list or data frame
- The '$' operator can be used to retrieve the elements from list or data frame according to name. Its language is similar to '[['.

An example is as follows:

- y <- c("a", "b", "c", "c", "b", "a")
- y[2]
  [1] "b"
- y[1]
  [1] "a"
- y[1:3]
  [1] "a" "b" "c"
- y[y > "a"]
  [1] "b" "c" "c" "d"
- p <- y > "b"
- p
  [1] TRUE FALSE TRUE TRUE FALSE TRUE
- y[u]
  [1] "a" "c" "c" "a"

✓ Subsetting a matrix

One can subset a matrix with (a,b) indices.

- x <- matrix(1:6, 2, 3)
- x[1, 2]
  [1] 3
- x[2, 1]
  [1] 2

Or one could also skip indices.

- x[1,]
  [1] 1 3 5
- x[, 2]
  [1] 3 4

✓ Subsetting Lists

Subsetting lists can be done as follows:

- y <- list(a = 1:4, b = 0.8)
- y[1]
  $a
  [1] 1 2 3 4
- y[[1]]
  [1] 1 2 3 4
- y$b
  [1] 0.8
- y[["b"]]
  [1] 0.8
- y["b"]
  $b
  [1] 0.8

✓ Subsetting the nested elements of Lists

The '[['operator can adopt an integer list.

- x <- list(a = list(10, 12, 14), b = c(3.14, 2.81))
- x[[c(1, 3)]]
  [1] 14
- x[[1]][[3]]
  [1] 14
- x[[c(2, 1)]]
  [1] 3.14

✓ Partial Matching

Partial names-matching can be achieved with '[[' and '$'.

- x <- list(aardvark = 1:5)
- x$a
  [1] 1 2 3 4 5
- x[["a"]]
  NULL
- x[["a", exact = FALSE]]
  [1] 1 2 3 4 5

✓ Removing NA Values

It is necessary to remove missing values.
NA stands for missing value.

- x <- c(1, 2, NA, 4, NA, 5)
- bad <- is.na(x)
- x[!bad]
  [1] 1 2 4 5

2.3 Control structures

Control structures in R provide a platform to control the execution flow of program. Some of them are:

✓ if, else

This is used to check the validity of a condition. Consider the following example:

```
if(<condition1>) {
## do a
} else {
## do b
}
if(<condition2>) {
## do c
} else if(<condition3>) {
## do d
} else {
## do e
}
```

✓ for

Can be used for executing a loop repeatedly. Three ways of doing it:
For example,

```
y <- c("d", "b", "a")
for(i in 1:3) {
print(y[i])
}
for(j in seq_along(y)) {
print(x[j])
}
for(letter in y) {
print(letter)
}
for(j in 1:3) print(y[j])
```

✓ while

It is deployed to execute instruction(s) while a condition is true. For example,

```
co <- 0
while(co < 21) {
print(co)
co <- co + 0.5
}
```

✓ repeat

Its purpose is to indefinitely execute an infinite loop. Consider the following example:

```
y1 <- 1
a <- 4e-6
repeat {
x1 <- computeEstimate()
if( (x1 – y1)  > 0) && (x1-y1) < a) {
break
} else if{
((x1-y1) < 0) && (y1-x1) < a
y1 <- x1
}
}
```

✓ next, return

The 'next' statement skips an iteration of a loop.
The 'return' control structure indicates that a function should exit and return a value. For example,

```
for(j in 1:111) {
if(i <= 45) {
## Skip
next
}
## Do z
}
```

2.4 Functions

They are considered as "first class objects", that is, similar to other R objects [7]. Certain properties of functions are:

- They can be passed as arguments to other functions
- They can be nested, so that you can define a function inside of another function
- Their return value is the final expression to be evaluated.

Their creation occurs by using the function() command and later, they are stored as R objects. Peculiarly, these are objects of class "function". For example,

```
x <- function(<arg>) {
## Do a1
}
```

✓ Function Arguments

Some properties of function arguments are:

- Named ones have default values.
- Formal ones are in function definition
- Formals ones also return a list of all the formal arguments
- Not every function call uses each formal argument
- Arguments maybe skipped if required. If not, they possess the values by default.

✓ Matching Arguments

INTERNATIONAL JOURNAL FOR ADVANCE RESEARCH IN ENGINEERING AND TECHNOLOGY

WINGS TO YOUR THOUGHTS.....

Function arguments are compared and matched by name or position [6]. So all the following calls to 'sd' are equivalent

- md <- rnorm(100)
- sd(md)
- sd(y = md)
- sd(y = md, na.rm = FALSE)
- sd(na.rm = FALSE, y = md)
- sd(na.rm = FALSE, md)

Function arguments can be matched partially if required. Order of operations when assigned an argument is to verify for -

1. Exact match
2. Partial match
3. Positional match

2.5 Lexical Scoping

The general scoping rules for R make R different from S

- Scoping rules determine the association of a value with a free variable.
- Related to the scoping rules is the way R uses the list of searched content with a motive of binding one value to corresponding symbol.
- In particular, lexical scoping is useful for making simple the computations.

Let us consider a function that has 2 formal arguments, that is, p and q.

Here, within the body of the function, let there be another symbol r.

In this case r is free variable. Scoping rules of a language make decisions about how values should be assigned to the free variables. They are neither formal arguments nor local variables. Consider the following example:

```
a <- function(p, q) {
p^2 + q / r
}
```

Consequences are:

- All objects should be present in memory
- All functions should carry a pointer to their respective environments that define them.

One application of lexical scoping is that plotting the likelihood of occurrence of an event is possible [1]. Consider the following example:

Refer to figure 2 for the following:

```
L1 <- make.NegLogLik(normals, c(1, FALSE))
a <- seq(2.7, 2.9, length = 150)
b <- sapply(x, L1)
plot(a, exp(-(b - min(b))), type = "l")
```

Refer to figure 3 for the following:

```
L2 <- make.NegLogLik(normals, c(FALSE, 2))
p <- seq(1.5, 2.5, length = 150)
q <- sapply(p, nLL)
    plot(p, exp(-(q - min(q))), type = "l")
```

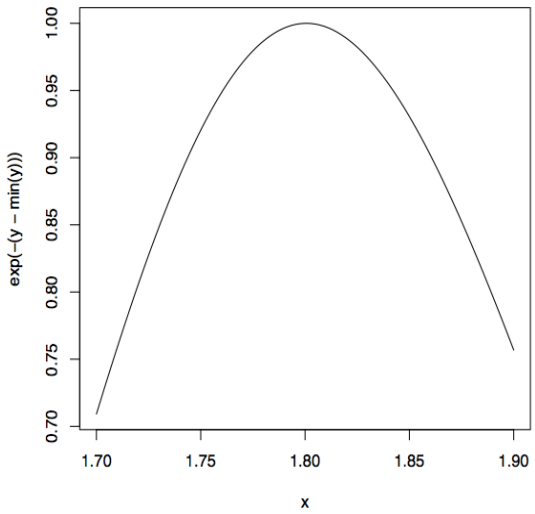

Figure 2

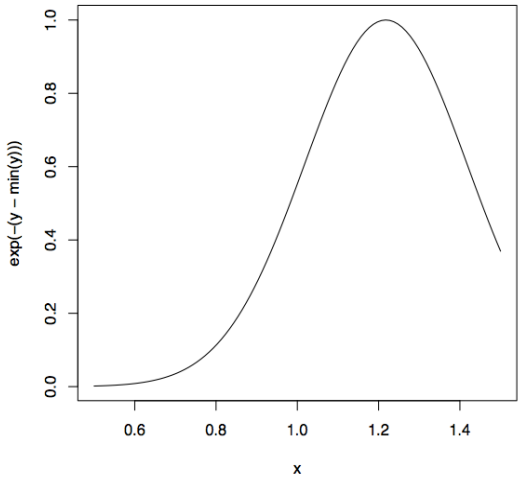

Figure 3

2.6 Date

Represented by the Date class, it can also be coerced from a character string by using as.Date() function as follows:

```
p <- as.Date("1970-01-01")
p
## [1] "1970-01-01"
unclass(p)
## [1] 0
unclass(as.Date("1970-01-02"))
```

```
## [1] 1
```

Date is stored in internal memory as number of days passed since '1970-01-01' [3].

2.7 Time

Represented using the POSIXct or the POSIXlt class, it can also be coerced from a character string directly by using.POSIXlt or as.POSIXct function [3].

```
p <- Sys.time()
p
## [1] "2014-08-24 12:05:30 EST"
q <- as.POSIXlt(p)
names(unclass(p))
## [1] "sec" "min" "hour" "monthday" "month"
## [5] "year" "weekday" "yearday" "isdst"
q$sec
## [1] 12.05
```

Time is saved in internal memory as the total number of seconds passed since '1970-01-01'

3. CHALLENGES IN BIG DATA

Few of the main challenges that pose a threat in big data analytics are as follows:

- Heterogeneity and Incompleteness
- Scale
- Timeliness
- Privacy
- Human Collaboration

There are some more challenging research problems. For instance, it is not yet known to share private data while putting a limit on the value of disclosure but also ensure sufficient utility of data in the shared information [4].

4. CONCLUSION

We live today in a trend of Big Data. From better analysis of large amounts of data which are accessible, there is a slight scope of making further advances in multifarious disciplines of scientific knowledge. Once achieved, this has the capacity to improve the usability of many firms. But the constraints are common across a host of domains. Thus, it is not profitable to address in the context of one domain alone. On the same lines, these problems will demand solutions like transformation. We need to support fundamental research in the topic of Big Data which would be addressing such technical issues if to attain the promised gifts of Big Data.

REFERENCES

[1] **Gartner Group. 2012. "Gartner Says Big Data Creates Big Jobs: 4.4 Million IT Jobs Globally to Support Big Data By 2015," IBM. 2013.**

[2] **"What Is Big Data?," Chen H., Chiang R. H. L., and Storey V. C. 2012.**

[3] **"Business Intelligence and Analytics: From Big Data to Big Impact," MIS Quarterly (36:4), pp. 1165-1188. Shmueli G., and Koppius, O. R. 2011.**

[4] **"Predictive Analytics in Information Systems Research," MIS Quarterly (35:3), pp. 553-572.**

[5] **Mattmann C, Crichton D, Hughes JS, Kelly S, Hardman S, Joyner R, Ramirez P: A classification and evaluation of data movement technologies for the delivery of highly voluminous scientific data products. In Proceedings of the NASA/IEEE Conference on Mass Storage Systems and Technologies (MSST2006). Maryland: IEEE Computer Society; 2006:131-135.**

[6] **Mattmann C, Crichton D, Hart A, Kelly S, Hughes JS: Experiments with storage and preservation of nasas planetary data via the cloud.IEEE IT Prof Spec Theme Cloud Comput 2010, 12(5):28-35.**

[7] **Kang Y, Kung SH, Jang H-J: Simulation process support for climate data analysis. InProceedings of the 2013 ACM Cloud and Autonomic Computing Conference. Vietri sul Mare, Italy: ACM; 2013:29-29. Mattmann C, Zitting J: Tika in action. NY, USA: Manning Publications Co.; 2011.**

[8] **Mattmann C, Medvidovic N, Ramirez PM, Jakobac V: Unlocking the grid. In CBSE. Lecture notes in computer science, vol. 3489. Edited by Heineman GT, Crnkovic I, Schmidt HW, Stafford JA, Szyperski CA, Wallnau KC. St. Louis, MO: Springer; 2005:322-336.**

[9] **Mattmann C, Garcia J, Krka I, Popescu D, Medvidovic N: The anatomy and physiology of the grid revisited. In WICSA/ECSA. London, UK: IEEE/IFIP; 2009. Yu J, Buyya R: taxonomy of workflow management systems for grid computing.**

[10] **J Grid Comput 2005, 3(3–4):171-200. .Massie ML, Chun BN, Culler DE: The ganglia distributed monitoring system: design, implementation, and experience, Parallel Comput 2004, 30(7):817-840**