Programming in Data science

Asad Waqar Malik

Department of Information Systems
Faculty of Computer Science and Information Technology
University of Malaya
Malaysia

Sept 12, 2019

Lecture Outline

Course Outline

- Introduction to R/IDE
- R Language Basics
- Operators, Atomic data types and Vectors
- Data Structures (Matrices, Lists, Factors) Subsetting
- Data Structure (Data Frames) Libraries Packages (dplyr)
- Control Structures (lapply, sapply...)Functions
- Files (Read and Write)
- Data Science Process : Getting Cleaning Data
- Mising values, NA and NaN Values
- Plotting Data with R
- Data Science Process : Statistics and Regression Modeling

Continuous assessment!

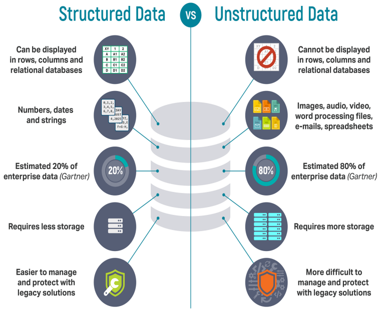
- Assignments 30%
- Lab test 20%
- Final Exam 50%

Course organization
- Eight Weeks (9/9/2019 – 3/11/2019)
- Mid semester break (4/11/2019 - 10/11/2019)
- Six Weeks (11/11/2019 - 22/12/2019)

Mid Semester Test 11/11/2019

Data science Overview

- Data science is the study of data. It involves developing methods of recording, storing, and analyzing data to effectively extract useful information.
- The goal is to gain insights and knowledge from any type of data — both structured and unstructured.
-

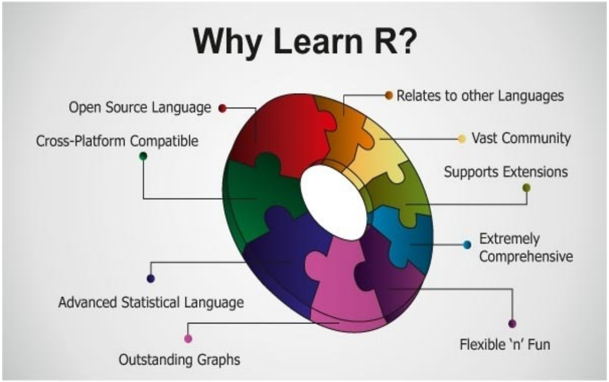

Structured Data
VS
Unstructured Data
Can be displayed in rows, columns and relational databases
Cannot be displayed in rows, columns and relational databases
Numbers, dates and strings
Images, audio, video, word processing files, e-mails, spreadsheets
Estimated 20% of enterprise data (Gartner)
20%
80%
Estimated 80% of enterprise data (Gartner)
Requires less storage
Requires more storage
Easier to manage and protect with legacy solutions
More difficult to manage and protect with legacy solutions

Data science turn raw data into understanding, insight, and knowledge.

Data science Usecase

- Knowing customer precise requirements from the existing data like the customer's past browsing history, purchase history, age and income. No doubt you had all this data earlier too, but now with the vast amount and variety of data, you can train models more effectively and recommend the product to your customers with more precision.
- The self-driving cars collect live data from sensors, including radars, cameras and lasers to create a map of its surroundings. Based on this data, it takes decisions like when to speed up, when to speed down, when to overtake, where to take a turn – making use of advanced machine learning algorithms.
- Data from ships, aircrafts, radars, satellites can be collected and analyzed to build weather forcast models. These models will not only forecast the weather but also help in predicting the occurrence of any natural calamities.

Introduction to R



Introduction to R

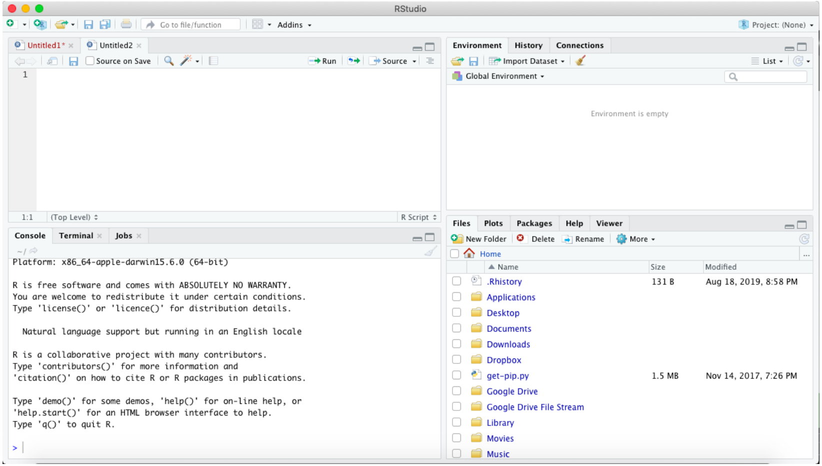
- R is free, open-source - result, many excellent programmers have contributed improvements and fixes to the R code.
- R runs anywhere - available for Windows, Unix systems, and the Mac.
- R supports extensions - Use existing packages to write their own code.
- R provides an engaged community – community mailing lists and websites such as Stack Overflow.
- R connects with other languages – easily connect to file systems, databases, and languages such as python, Java, and C++.

How to install R Studio

- Installing R: Go to R Project website[1] and download R for your operating system.
- Installing R Studio: Go to RStudio website[2] and click on "Download RStudio" and follow the directions for your operating system

[1] https://cran.r-project.org/
[2] https://www.rstudio.com/

R IDE

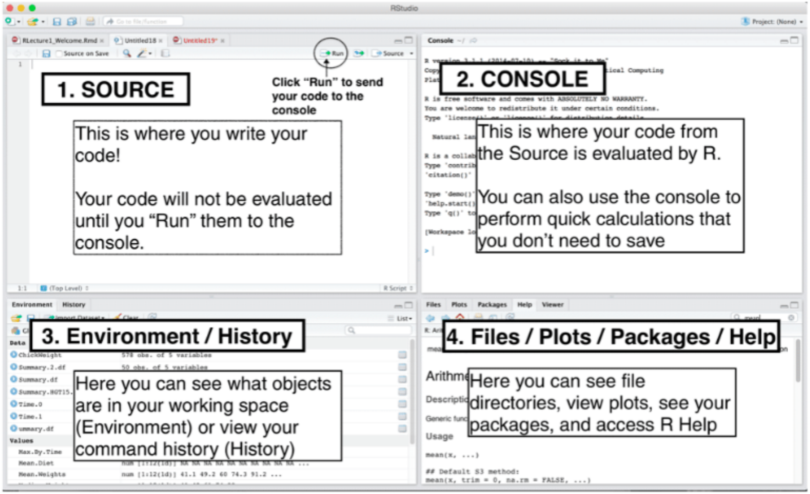


```
Platform: x86_64-apple-darwin15.6.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

Name	Size	Modified
.Rhistory	131 B	Aug 18, 2019, 8:58 PM
Applications		
Desktop		
Documents		
Downloads		
Dropbox		
get-pip.py	1.5 MB	Nov 14, 2017, 7:26 PM
Google Drive		
Google Drive File Stream		
Library		
Movies		
Music		

Figure 1: R Studio/IDE

R IDE



Click "Run" to send your code to the console

This is where you write your code!

Your code will not be evaluated until you "Run" them to the console.

This is where your code from the Source is evaluated by R.

You can also use the console to perform quick calculations that you don't need to save

Here you can see what objects are in your working space (Environment) or view your command history (History)

Here you can see file directories, view plots, see your packages, and access R Help

```
mean(x, ...)

## Default S3 method:
mean(x, trim = 0, na.rm = FALSE, ...)
```

Figure 2: R Studio/IDE[3]

R IDE[3]

- Source – Where you create and edit R Scripts – your collections of code. R scripts are just text files with the ".R" extension.
- Console – Present at the bottom left of the window. It is also called a command window. You can type code directly into the console after the prompt and get an immediate response.
- Environment/History – shows the names of all the data objects defined in current R session. The history window shows all commands that were executed in the console.
- Files/Plots/Packages/Help
  Files : gives you access to the file directory on your hard drive.
  Packages: Shows a list of all the R packages installed on your hard drive and currently loaded.
  Help Help: Get help, type the name of a function in the search window.

[3]https://datascienceplus.com/introduction-to-rstudio/

R Basics

- At the R prompt, the we type expressions.
- The ← symbol (gets arrow) is the assignment operator.

Code

```
> x <- 1        # assignment expression
> print(x)      # explicit printing
[1] 1           # output
> x
[1] 1
> msg <- "hello"
x <-        # Incomplete expression
```

- The grammar of the language determines whether an expression is complete or not.
- The # character indicates a comment. Anything to the right of the (including the itself) is ignored.
- R does not support multi-line comments or comment blocks.

R Basics

- When printed you will notice that an index for the vector is printed in square brackets [] on the side. For example, see this integer sequence of length 20.

Sample Code

```
> x <- 11:30
> x
[1] 11 12 13 14 15 16 17 18 19 20 21 22
[13] 23 24 25 26 27 28 29 30
```

Objects

- R has five basic classes of objects:

- Character – "a", "um"
- numeric (real numbers) – 2, 15.5
- integer – 2L (the L tells R to store this as an integer)
- logical – TRUE, FALSE
- complex – 1+4i (complex numbers with real and imaginary parts)

Data Structures in R

- In contrast to different programming languages like C and Java, R doesnot have variables declared as some data type.
- The variables are appointed with R-objects and the knowledge form of the R-object becomes the datatype of the variable.
- There are many types of R-objects. The popularly used ones are:

1. Vector
2. Matrix
3. Array
4. Lists
5. Data Frames

Vector

- Most basic type of R object is a vector.
- Empty vectors can be created with the vector() function.
- Only one rule about vectors in R, i.e. A vector can only contain objects of the same class.

Sample Code

```
> 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
```

- The c() function can be used to create vectors of objects by concatenating things together.

Vector

Sample Code

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

Help!

```
> x <- vector()
> x
```

What should be the output?

Vector

- Mixing Objects – Different classes of R objects get mixed together, happens by accident or purposefully.

Try it out!

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

Vector

- Mixing Objects – Different classes of R objects get mixed together, happens by accident or purposefully.

Output

```
1. [1] "1.7", "a"
2. [2]   1     2
3. ???? Try out
```

Coercion

- When different objects are mixed in a vector, coercion occurs so that every element in the vector is of the same class.
- Implicit coercion – When R tries to find a way to represent all of the objects in the vector in a reasonable fashion.
- Explicit coercion – Objects can be explicitly coerced from one class to another using the as.* functions.

Sample Code

```
> x <- 0:6
> class(x)
[1] "integer"
> as.numeric(x)
[1] 0 1 2 3 4 5 6
> as.logical(x)
[1] FALSE TRUE TRUE TRUE TRUE TRUE TRUE
>as.character(x)
[1] "0" "1" "2" "3" "4" "5" "6"
```

Coercion

- Sometimes, R can't figure out how to coerce an object and this can result in NA (not available) being produced.

Sample code

```
> x <- c("a", "b", "c")
> as.numeric(x)
Warning: NAs introduced by coercion
[1] NA NA NA

> as.logical(x)
[1] NA NA NA

> as.complex(x)
Warning: NAs introduced by coercion
[1] NA NA NA
```

Vectors from a Sequence of Numbers

- You can create vectors as a sequence of numbers.

Sample code

```
> series <- 1:10
> series <- seq(10)
> series <- seq(from =1, to= 10, by=0.1)
```

Using integer vector as index

- Vector index in R starts from 1, unlike most programming languages where index start from 0.
- ⇒ We can use a vector of integers as index to access specific elements.

Sample Code

```
> x <- seq(0,10,2)
> x
[1] 0 2 4 6 8 10

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

Using integer vector as index

⇒ We can use a vector of integers as index to access specific elements.

Sample Code

```
> x
[1] 0 2 4 6 8 10
> x[c(2,4)]      # access 2nd and 4th element
[1] 2 6
```

⇒ We can also use negative integers to return all elements except that those specified.

```
> x[-1]     # access all but not 1st element
[1] 2 4 6 8 10
```

Using integer vector as index

⇒ But we cannot mix positive and negative integers while indexing and real numbers, if used, are truncated to integers.

Sample Code

```
> x[c(2, -4)]     # cannot mix positive and negative↩
      integers
> x[c(2.4, 3.5)]  # real numbers are truncated to ↩
     integers
```

Using logical vector as index

- When we use a logical vector for indexing, the position where the logical vector is TRUE is returned.
- This useful feature helps us in filtering of vector as shown here.

Sample Code

```
> x[c(TRUE, FALSE,FALSE,TRUE, FALSE, FALSE)]
[1] 0  6

> x[x < 3 ]      # filtering vectors based on ↩
      conditions
[1] 0   2

> x[x > 3]
[1] 4 6 8 10
```

- The expression x>0 yield a logical vector (TRUE, FALSE, FALSE, TRUE) which is used for indexing.

How to modify a vector in R?

- We can modify a vector using the assignment operator.
- If we want to truncate the elements, use reassignments.

Sample Code

```
> x
[1] 0 2 4 6 8 10
> x[2] <- 4      # modify 2nd element
> x
[1] 0 4 4 6 8 10

> x[x > 5] <- 5 # modify elements greater than 5
> x
[1] 0 2 4 5 5 5

> x <- x[1:4]  # truncate to first 4 elements
> x
[1] 0 2 4 5
```

How to delete a Vector?

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

Sample Code

```
> x
[1] 0 2 4 6 8 10
> x <- NULL
> x
NULL
> X[4]
NULL
```

Arithmetic Operators

Operator	Description
+	addition
-	subtraction
*	multiplication
/	division
**	exponentiation
x %% y	modulus (x mod y) 5%%2 is 1
x % / % y	integer division 5% / %2 is 2

Logical Operators

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 OR y
x & y	x AND y
isTRUE(x)	test if X is TRUE

Operators

Sample Code

```
> x <- c(1:10)
> x[(x > 8) | (x < 5)]
[1] 1 2 3 4 9 10
```

R rep() Function

- `rep(..)` is used for replicating the values in x.

```
To repeat the vector c(0, 0, 7) three times, use ↩
    this code:
> rep(c(0, 0, 7), times = 4)
[1] 0 0 7 0 0 7 0 0 7 0 0 7
```

```
We can also repeat every value by specifying each ↩
    argument, like this:
> rep(c(2, 4, 2), each = 2)
[1] 2 2 4 4 2 2
```

R rep() Function Conti..

- For each value, we can tell R how often it has to repeat:

```
> rep(c(0, 7), times = c(4,3))
[1] 0 0 0 0 7 7 7
```

- The argument, length.out repeat the vector until it reaches the specified length, even if the last repetition is incomplete:

```
> rep(1:3,length.out=9)
[1] 1 2 3 1 2 3 1 2 3
```

R seq(..) Function

- `seq(..)` generates regular sequences

```
> seq(from = 4.5, to = 3.0, by = -0.5)
[1] 4.5 4.0 3.5 3.0
```

- You can specify the length of the sequence by using the argument, length.out. Afterwards, R can calculate the step size by itself.

```
> seq(from = -2.7, to = 1.3, length.out = 9)
[1] -2.7 -2.2 -1.7 -1.2 -0.7 -0.2 0.3 0.8 1.3
```

any(..), all(..) function

- It takes the set of vectors and returns a set of logical vectors, in which at least one of the value is true.
- Check if any or all the elements of a vector are TRUE. Both functions also accept many objects.

```
> any(..., na.rm=FALSE)
... means one or more R objects that need to be checked.
na.rm means state whether NA values should be ignored or not.
```

```
> all(..., na.rm=FALSE)
...means one or more R objects that need to be checked.
na.rm means state whether NA values should be ignored or not.
```

Exercise

- Write a R program to add two vectors of integers type and of length 3.
- Write a R program to multiply two vectors of integers type and of length 3.
- Consider a vector of size 3 with value v(4,5,6). Append two more values in the same vector so its become of length 5 i.e. v(4,5,6,7,8).

Exercise Code

In each case, what is the value of x?

1. x ← 2 - 1 * 2
2. x ← 6/3 -2+1*0+3/3 -3
3. x ← 19 %% 17 %% 13 # compare to (19 %% 17) %% 13 and 19 %% (17 %% 13)
4. x ← 2^17 %% 17
5. x ← 3 -2 %% 5+3 *2 -4/2