Statistical Computing with R
Masters in Data Science 503 (S3)
Fourth Batch, SMS, TU, 2025

Shital Bhandary

Associate Professor

Statistics/Bio-statistics, Demography and Medical Education

Patan Academy of Health Sciences, Lalitpur, Nepal

Faculty, Masters in Medical Research, NHRC/Kathmandu University

Faculty, FAIMER Fellowship in Health Professions Education, India/USA

Review Preview

- Basics of R
- Chapter from "R for Everyone" book
- **We will discuss today based on this chapter!**

- Basics of coding in R
- Chapter from "Hands-on Programming with R" book
- **You must read this for the next class!**

Basics of R

- > 4 * 6 + 5
- (4 * 6) + 5
- 4 * (6 + 5)
- (4 + 6)^2 * 5 / 10 + 9 - 1
- ?

- R can do Math!
- It follows PEMDAS rule
- **P**arenthesis, **E**xponents, **M**ultiplication, **D**ivision, **A**ddition and **S**ubtraction
- **BODMAS rule?**

Variables in R: assigning and removing

- **x <- 2** (preferred)
- x = 2
- 2 -> x
- assign(“x”, 2)
- rm(x)

- Variable names can contain any combination of alphanumeric characters along with period(.) and underscore (_) e.g. **age.group** or **age_group**
- However, they cannot start with a number or an underscore e.g. **_age** or **5age**
- Best practice is to use actual names, usually nouns for variables instead of single letter e.g. **age, sex**

R is case sensitive

- theVariable <- 17
- will give error if we type:
- TheVariable
- THE VARIABLE

- Age <- 50
- will be different for:
- age
- AGE

Data Types

- Numeric

- x <- c(1,2,3,4,5,6,7,8,9)

- **Type of data can be checked using class() function**

- For numeric “class” and “is.numeric” both works:

- class(x)
- is.numeric(x)

Data Types

- Integer
- x <- c(1:9) or c(1L:9L)
- Or
- X <- c(1L,2L,3L,4L,5L,6L,7L,8L,9L)

- For integer "class" and "is.numeric" both works:
- class(x)

  "integer"

- is.numeric(x)

  TRUE **(why?)**

R promotes “integers” to “numeric” when needed

#Multiply integer by numeric in decimal values

- 4L * 2.8

#Divide integer by integer giving decimal value

- 5L / 2L

#Will not promote to numeric here

- 4L * 5L

#Will also not promote here

- **2L + 4L + 5L**

- class(4L)
- class(2.8)
- class(4L * 2.8)

- class(5L)
- class(2L)
- class(5L/2L)

- class(4L * 5L) **(?)**
- **class(2L + 4L + 5L)**

Data Types

- Character
- x <- “data”
- Factor
- y <- factor(“data”)

- x
- class(x)
- nchar(x)
- y
- class(y)
- nchar(y)

Factors and attributes in R:

- Factor is used to create and store categorical variable in R like Sex (Male/Female), Blood group (A, B, AB, O) and Blood Rh factor (Positive/Negative) etc. **We will need this for supervised learning!**

- > gender <- factor(c("male", "female", "female", "male")
- > typeof(gender)   #datatype
- > attributes(gender)   #Levels and class
- > unclass(gender)   #Check how it is stored in R

```r
gender <- factor(c("male", "female", "female", "male"))

typeof(gender)
## "integer"

attributes(gender)
## $levels
## [1] "female" "male"
##
## $class
## [1] "factor"
```

You can see exactly how R is storing your factor with `unclass`:

```r
unclass(gender)
## [1] 2 1 1 2
## attr(,"levels")
## [1] "female" "male"
```

https://rstudio-education.github.io/hopr/r-objects.html#attributes

Data Types

- Date
  - To store date
- POSIXct
  - To store date and time
- Easier manipulation of date and time objects can be accomplished using “lubridate” and “chron” packages

- date1 <- as.Date(“2023-03-29”)
- date1
- class(date1)
- as.numeric(date1)
- date2 <- as.POSIXct(“2023-03-29 06:30”)
- date2
- class(date2)
- as.numeric(date2)

Data Types

- Logical
  - TRUE (=1)
  - FALSE (=0)
- TRUE * 5
- FALSE * 5

#Class and check:

- k <- TRUE
- class(k)
- is.logical(k)

Logical Data Types

- 2 == 3 (FALSE)
- 2 != 3 (TRUE)
- 2 < 3 (TRUE)
- 2 <= 3 (TRUE)

- 2 > 3 (FALSE)
- 2 >= 3 (FALSE)
- “data” == “stats” (FALSE, why?)
- “data” < “stats” (TRUE, why?)

Vectors

- A vector is collection of elements, all of the same type.
- Vector in R is like a **set** with different types of data
- R is a vectorized language
- Vectors do not have dimension
- Vectors in R are not like the mathematical vector
- Column and row vectors can be represented as one-dimensional matrices, however!

Vectors and its operation in R

- x <- c(1,2,3,4,5,6,7,8,9,10)
- x is a vector containing 10 elements
- c stands for "combine"
- It combines multiple elements into a vector
- Shortcut is: 1:10 or 10:1 or -2:3 or 5:-7

- x * 3 #Multiplication by a scalar
- x + 2 #Addition with a scalar
- x – 3 #Subtraction with a scalar
- x / 4 # Division by a scalar
- x^2 #Exponentiation by a scalar
- sqrt(x) #Square root

Extending vector operations in R

#Two vector of equal length

- x <- 1:10
- y <- -5:4

- x+y
- x-y
- x*y
- x/y
- x^y

- #Check length of the vector
- length(x)
- length(y)

- length(x+y)

Extending vector operations in R

#Two vectors of unequal length

- x <- 1:10
- z <-c(1,2)

- x+z

- Shorter vector get recycled i.e. its elements are repeated, in order, until they have been matched up with every element of the longer vector

#Two vectors of unequal length

- x <- 1:10
- w <-c(1,2,3)

- x+w

- If the longer one is not a multiple of shorter one, warning is given

Extending vector operations in R

#Comparing vectors

x <= 5

x > y

x < y

- # Using “any” and “all”
- x <- 10:1
- y <- -4:5
- any(x<y)
- all(x<y)

- #Using “nchar”
- nchar(y)

Extending vector operations in R

#Assessing individual elements of a vector

- x[1] retrieves first element of x
- x[1,2] retrieves first and second elements of x
- x[c(1,4)] retrieves?

#Giving names to a vector

#Name value pair

c(One="a", Two="y", Last="r")

#Create vector then name it

w <- 1:3

names(w) <- c("a", "b", "c")

w

Calling in-built functions in R

We have already used

- nchar
- length
- as.Date
- as.POSIXct

#We can also use

- mean, var, sd
- round
- factorial

#Getting details of a sensed function

aproos(“mea”)

Missing data in R

- R has two types of missing data
- NA
- NULL

- Statistical programs use various techniques to represent missing data such as dash, a period or even the number 99
- R uses NA
- NA is represented as just another elements of a vector

NA type missing data in R

- zchar <- c(“Hockey”, NA, “Cricket”)
- nchar(z)

- z <- c(1,2,NA,8,3,NA,3)
- mean(z)

- Missing data can be handled using multiple imputation with mi, mice and Amelia packages

#The “is.na” function tests each element of vector for missingness

- is.na(z)

#The na.rm function with =TRUE argument will remove NA so that we can get values for:

- mean(z, na.rm=TRUE)
- var(z, na.rm=TRUE)
- sd(z, na.rm=TRUE)

NULL type missing data in R

- NULL is the absence of anything
- It is “nothingness”
- Functions can sometimes return NULL and their arguments can be NULL
- NULL is atomical and cannot exist with a vector

- z <- c(1, NULL, 3)
- z
- [1] 1 3
- is.null(z)

- d <- NULL
- is.null(d)

- is.null(7)

Pipes in R

- A new convention for calling functions in R is the pipe
- The pipe comes from the "magrittr" package BUT starting R 4.0.0 it has in-built pipe now
- x <- 1:10
- mean(x)

- Mean of x with pipe:
- library(magrittr)
- x %>% mean
- It works by taking the value or object on the left hand side of the pipe and inserting it into the first argument of the function that is on the right-hand side of the pipe

Chained pipes in R

- Pipes are most useful when used in a pipeline to chain together a series of function calls
- Given a vector z that contains numbers and NAs, we want to find out how may NAs are present
- Pipes is negligible slower than nesting; but not a bottleneck

#Traditionally we do it by nesting

- z <-c(1,2,NA,8,3,NA,3)
- sum(is.na(z))

#Pipes, without nesting

- z %>% is.na %>% sum

#Additional argument

z %>% mean(na.rm=TRUE)

Advanced data structures in R

- Data Frame (data.frame)
- In R data.frame, each column is a vector, each of which is has the same length and same type
- It lets each column holds a different type of data

- x <- 10:1
- y <- -4:5
- q <-c("Hockey", "Football", "Baseball", Kabaddi", "Rugby", "Pingpong", "Basketball", "Tennis", "Cricket", "Volleyball")
- theDF <-data.frame(x, y, q)
- theDF

Advanced data structures in R

- theDF <-data.frame(First=x, Second=y, Sport=q)
- names(theDF)
- names(theDF)[3]
- rownames(theDF)
- rownames(theDF) <- c(“One”, “Two”, “Three”, “Four”, “Five”, “Six”, “Seven”, “Eight”, “Nice”, “Ten”)
- Setting them back to generic index
- rownames(theDF) <- NULL
- rownames(theDF)

Advanced data structures in R

#Printing first few rows

- head(theDF)

#Printing first seven rows

- head(theDF, n=7)

Printing last few rows

- tail(theDF)

- class(theDF)

#Structure of data frame by variables

- str(theDF)

- theDF[3,2]; theDF[3, 2:3]
- theDF[, 3]; theDF[3,]
- theDF[, c("First", "Sport")]
- theDF[, "Sport", drop=FALSE]

Lists in R

- Often a container is needed to hold arbitrary objects of either the same type or varying types
- R accomplishes this through lists
- They store any number of items of any type; it will be helpful hold large texts with numbers and symbols and mining it
- A list can contain all numerics or characters or a mix of two or data.frame or, recursively, other lists
- Lists are created with list function where each argument to the function becomes an element of the list

Lists in R

#Three element list
- list1 <- list(1,2,3)

#Single element list
- list2 <- list(c(1,2,3)

#Two vector list
- list3 <- list(c(1,2,3), 3:7))

#List with data.frame and vector
- list4 <- list(theDF, 1:10)

#Three element list
- list5 <- list(theDF, 1:10, list3)

#Names of the list
- names(list5)
- names(list5) <-c("data.frame", "vector", "list")
- names(list5)
- list5
- list6 <- list(TheDataFrame=theDF, TheVector=1:10, TheList=list3)
- names(list6)

Access elements of list

- Use double square brackets

- Specify either the **element number or name**
- list5[[1]]
- list5[[“data.frame”]]

- This allows access to only one element at a time

#Accessed element manipulation

- lists5[[1]]$Sport #Sport variable
- lists5[[1]][, “Second”]
- lists5[[1]][, “Second”, drop=F]
- length(list5)

#Adding new element

- list5[[4]] <- 2
- list5[[“NewElement”]] <-3:6
- names(list5) & list5

Matrices in R

- This is a similar to a data.frame

- It is rectangular with rows and columns except that every single element must be the same type, most commonly all numerics

- They also act similarly to vectors with elements to element addition, multiplication etc.

- A <- matrix(1:10, nrow=5)
- B <- matrix(21:30, nrow=5)
- C <- matrix(21:40, nrow=2)

- nrow(A)
- ncol(B)
- dim(C)

- A + B; A * B; A – B; A = B

Matrix multiplication and names in R

- **Matrix multiplication of A and B matrices?**
- Number of columns of the left hand matrix to be same as number of rows of right hand matrix
  - A %*% C will work
  - A %*% B will not work
- Both A and B are 5 x 2 matrices so we will transpose B
  - A %*% t(B)

#Column/row names of matrix:

- colnames(A)
- colnames(A) <- c("Left", "Right")
- rownames(A) <- c("1st", "2nd", "3rd", "4th", "5th")
- t(A)
- colnames(B) <- c("First", "Second")
- rownames(B) <- c("One", "Two", "Three", "Four", "Five")

Arrays in R

- An array is essentially a multidimensional vector
- It must be of the same type, and individual elements are accessed in a similar fashion using square brackets
- **Very useful for creating and/or replicating multi-way tables in R**

- Array: first element is the row index, the second is the column index and remaining elements are for outer dimensions
- theArray <- array(1:12, dim=c(2,3,2))
  - 2 dimensional matrices both with 2 rows and 3 columns
- theArray [1, ,] 1st row of both
- theArray[1, ,1] 1st row of first
- theArray[,1,] 1st column of both

Questions/queries?

- Next session:

- **R Studio and use for coding, data manipulation and analysis**

Thank you!

@shitalbhandary