Introduction to R

Abdirisak Mohamed

Spring 2018

Contents

Basics of R

RStudio is an integrated development environment for R. One of the advantages of R is that you can start off as a user. It has an interactive console where you can enter commands. It is similar to a calculator.

Arithmetic

```
> 4 + 5
[1] 9
```

```
>  1 + 2*3 - 4^2
[1] -9
```

```
>  1 + (2*3) - (4^2)
[1] -9
```

R Objects

R has five atomic (basic) classes of objects:

- Character
- Numeric (real numbers)
- Integer
- Complex
- Logic (TRUE/FALSE)

Examples (the <- is the assignment operator)

```
> name<-"Rockville"
> x<-sqrt(2)
> n<-8L
> z<-3 + 4i
> statement<-FALSE
```

The # character signals the start of a comment. # and everything after it are not evaluated

```
> y<-9 # not evaluated
```

We used the function sqrt() above. It is one of many built-in functions in R.

Built-In Functions

Print. You can either auto-print or use the function print()

```
> y
[1] 9
> print(y)
[1] 9
```

Integer sequence

```
> z<-1:10
> z
 [1]  1  2  3  4  5  6  7  8  9 10
> x <- seq(from=0, to=10, by=2)
> x
[1]  0  2  4  6  8 10
> length(x)
[1] 6
```

Vectors: The most basic object

A vector can only contain elements of the same class

```
> w<-c("name", "age", "gender")
> x<-c("TRUE", "FALSE")
> y<-c(1.5, 2.7)
> z<-1:10
```

The function vector() creates a vector

```
> x<-vector("numeric", length = 4)
> x
[1] 0 0 0 0
```

Index of a vector starts at 1 not 0

```
> x<-c(7,8,9)
> x[1]
[1] 7
> x[0] # numeric vector of length zero (i.e., empty)
numeric(0)
```

Implicit Coersion:

If you put elements of different classes in a vector, R coerces them into one class

```
> x <- c(2.9, "b")   ## character
> class(x)
[1] "character"
> y <- c(5, FALSE)    ## numeric
> class(y)
[1] "numeric"
> z <- c("b", TRUE)  ## character
> class(z)
[1] "character"
```

Explicit Coersion

You can be explicitly coerce a vector from one class to another using exisitng as.* functions

```
> x <- 1:3
> x
[1] 1 2 3
> class(x)
[1] "integer"
```

```
> as.numeric(x)
[1] 1 2 3
> as.logical(x)  # 0 = FALSE, other integers = TRUE
[1] TRUE TRUE TRUE
> as.character(x)
[1] "1" "2" "3"
>
> # You get NAs as the conversion is not supported
> y <- c("a", "b", "c")
> y
[1] "a" "b" "c"
> as.numeric(y)
Warning: NAs introduced by coercion
[1] NA NA NA
> as.logical(y)
[1] NA NA NA
>
> # Logical to numeric
> z<-c(TRUE, FALSE, TRUE)
> z
[1]  TRUE FALSE  TRUE
> as.numeric(z)
[1] 1 0 1
> as.character(z)
[1] "TRUE"  "FALSE" "TRUE"
```

Matrices

Matrices are vectors with dimensions. The dimension attribute is a vector v = c(nrow, ncol) of integers. Like vecors, matrices contain elements of the same class.

```
> my_mat <- matrix(nrow = 3, ncol = 4)
> my_mat
     [,1] [,2] [,3] [,4]
[1,]   NA   NA   NA   NA
[2,]   NA   NA   NA   NA
[3,]   NA   NA   NA   NA
> dim(my_mat)
[1] 3 4
> attributes(my_mat)
$dim
[1] 3 4
```

Matrices are filled column-wise. You can change it by setting byrow = TRUE.

```
> mat_2<- matrix(1:6, nrow = 2, ncol = 3)
> mat_2
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
>
> # Fill by rows
> mat_3<- matrix(1:6, nrow = 2, ncol = 3, byrow = TRUE)
> mat_3
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

Adding a dimension attribute to a vector makes it a matrix

```
> x<-1:6
> x
[1] 1 2 3 4 5 6
> dim(x)<-c(2,3)
> x
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

Column-binding and Row-binding

```
> x<-1:6
> x
[1] 1 2 3 4 5 6
> y<-11:16
> y
[1] 11 12 13 14 15 16
> c_mat<-cbind(x,y)
> c_mat
     x  y
[1,] 1 11
[2,] 2 12
[3,] 3 13
[4,] 4 14
[5,] 5 15
[6,] 6 16
> r_mat<-rbind(x,y)
> r_mat
  [,1] [,2] [,3] [,4] [,5] [,6]
```

```
x    1    2    3    4    5    6
y   11   12   13   14   15   16
```

Lists

Lists are like vectors but they can contain elements of different classes

```
> x <- list(1:6, c("name","gender", "age"), FALSE)
> x
[[1]]
[1] 1 2 3 4 5 6

[[2]]
[1] "name"   "gender" "age"

[[3]]
[1] FALSE
```

Accessing elements of a list

```
> x[[3]]
[1] FALSE
> x[[2]][3]
[1] "age"
> x[[1]][5]
[1] 5
```

Factors

Factors represent categorical variables

```
> x <- factor(c("January", "April", "January", "June", "April", "April"))
> x
[1] January April   January June    April   April
Levels: April January June
> table(x)
x
  April January    June
      3       2       1
> unclass(x)
[1] 2 1 2 3 1 1
attr(,"levels")
[1] "April"   "January" "June"
> # You can set the order of the levels
> y <- factor(c("January", "April", "January", "June", "April", "April"),
```

```
+             levels = c("January", "April", "June"))
> y
[1] January April   January June    April   April
Levels: January April June
```

Missing Values

NA stands for Not Available and NaN stands for Not a Number.

The function is.na() is used to test if an element or object is NA.

Similarly, is.nan() is for testing NaN.

NA values have a class,e.g., integer NA, character NA.

Every NaN value is also NA value but the converse does not hold

```
>  x<-c(3, 5, NA, "a", TRUE)
> is.na(x)
[1] FALSE FALSE  TRUE FALSE FALSE
> is.nan(x)
[1] FALSE FALSE FALSE FALSE FALSE
> y<-c(3, 5, NA, "a", NaN)
> is.na(y)
[1] FALSE FALSE  TRUE FALSE FALSE
> is.nan(y)
[1] FALSE FALSE FALSE FALSE FALSE
```

Data Frames

Data Frames contain data in tabular form.

Data Frames are a special type of list; every member of the list has the same length.

The columns are the members of the list.

Data frames can (in contrast to matrices) contain different classes of objects

Data frames have the attribute row.names

If you want to read a file as a data frame, use read.table() or read.csv().

Data Frames can be converted to a matrix by the function data.matrix()

```
> df <- data.frame( age = c(20,30,25, 23),
+                   class = c("Freshman", "Senior" , "Junior", "Sophomore") )
> df
  age     class
```

```
1  20  Freshman
2  30    Senior
3  25    Junior
4  23 Sophomore
> nrow(df)
[1] 4
> ncol(df)
[1] 2
```

Names

Vectors can have names

```
> vec<-c(25, 5.5, 150)
> vec
[1]  25.0   5.5 150.0
> names(vec)
NULL
> names(vec)<-c("age", "height", "weight")
> vec
   age height weight
  25.0    5.5  150.0
> names(vec)
[1] "age"    "height" "weight"
```

Lists with names

```
> z<-list(age = c(20,30,25, 23), class = c("Freshman", "Senior" , "Junior", "Sophomore"))
> z
$age
[1] 20 30 25 23

$class
[1] "Freshman"  "Senior"    "Junior"    "Sophomore"
```

Accessing a list by names

```
> z$age
[1] 20 30 25 23
> z$class
[1] "Freshman"  "Senior"    "Junior"    "Sophomore"
```

Matrices with names

```
> my_mat <- matrix(1:6, nrow = 2, ncol = 3)
> dimnames(my_mat) <- list(c("a", "b"), c("c", "d", "e"))
> my_mat
```

```
  c d e
a 1 3 5
b 2 4 6
```

Accessing names

```
> names(df)
[1] "age"   "class"
> colnames(df)
[1] "age"   "class"
> rownames(df)
[1] "1" "2" "3" "4"
> row.names(df)
[1] "1" "2" "3" "4"
```