STATS 266 Handout - Data Structures & Manipulation

Qi Wang

2025-02-28

Contents

1 Introduction

Welcome to **STATS 266: Introduction to R**. This handout provides an introduction about data structures and manipulation in R. By the end of this document, you should be able to:

- Identify the 5 main data types.
- Begin exploring data frames, and understand how they are related to vectors and lists.
- Be able to ask questions from R about the type, class, and structure of an object.
- Understand the information of the attributes "names", "class", and "dim".

For this part, valuable materials to refer to include http://adv-r.had.co.nz/Subsetting.html and https://swcarpentry.github.io/r-novice-gapminder/04-data-structures-part1.html.

2 Type of Variables

There are five major types of atomic vectors: logical, integer, double, complex, and character. Integer and double are also known as numeric vectors.

```
v1 <- c(TRUE,FALSE)
typeof(v1)
```

```
## [1] "logical"
```

```
v2 <- c(1L,2L,3L)
typeof(v2)
```

```
## [1] "integer"
```

```
v3 <- c(1,2,3)
typeof(v3)
```

```
## [1] "double"
```

```
v4 <- c("string 1", "string 2")
typeof(v4)
```

```
## [1] "character"
```

```r
v5 <- 1+1i
typeof(v5)
```

```
## [1] "complex"
```

3 Data Structures

Data in the real world will not exist as one single element. They always exists like a vector, matrix, or data frame and so on. So this section introduces the data structures in R and how we can subset them.

3.1 Atomic Vectors

3.1.1 What is an atomic vector?

An atomic vector in R is the simplest type of data structure that holds elements of the same data type. It is a one-dimensional structure where all elements belong to the same class (e.g., numeric, character, logical).

I believe most of you are now confused what does the function `c()` do here. It can be understood as a way to concatenate elements and creating a new vector.

To combine atomic vectors, the function `c()` is used. It flattens vectors, creating a new atomic vector containing all of the elements. When dealing with combining different types, we need to be careful:

```r
c(1,0,TRUE,FALSE)
```

```
## [1] 1 0 1 0
```

```r
c(1,0,'a','b')
```

```
## [1] "1" "0" "a" "b"
```

For missing values, R treats them as `NA` short for not applicable. Most calulations related to `NA` lead to `NA`, unless the results hold true for all possible values.

3.1.2 Subsetting an atomic vector

We use single bracket `[i]` to get the i-th element of the vector:

```r
v1 <- c(2,4,6,8,10)
v1[1]
```

```
## [1] 2
```

```
v1[c(1,3)]
```

```
## [1] 2 6
```

```
v1[c(T,F,T,F,T)]
```

```
## [1]  2  6 10
```

More advanced cases include filtering some elements that we want. For example, if we want elements between -0.5 and 0.5 in a vector, what should we do?

```
v1 <- round(rnorm(100,0,0.5),2)
v1
```

```
##   [1]  0.43  0.75  0.39 -0.76  0.62 -0.55 -0.84  0.41 -0.19  0.95 -0.09 -0.81
##  [13]  0.40 -0.06 -0.16  1.03 -0.64 -0.33 -0.43 -0.33  0.63 -0.44  0.95  0.11
##  [25] -0.15  0.44 -0.25 -0.43 -0.38  0.06 -0.15  0.18 -0.20 -0.38  0.06 -0.98
##  [37]  0.45 -0.36  0.53 -0.90  0.67  1.08  0.03  0.18  0.18 -0.65  0.17 -0.08
##  [49] -0.47  0.00  0.03 -0.45  0.79  0.16 -0.29  0.47  0.92  0.65 -0.21 -0.27
##  [61] -1.29  0.13 -0.39  0.25  0.07  0.20  0.36 -0.59 -0.05  0.27  0.56  0.34
##  [73] -0.01 -0.51  0.44 -0.38 -1.95  0.38 -0.84  0.06 -0.06  0.77  0.10  0.14
##  [85] -0.23  0.22 -0.61  0.89 -0.26  0.54 -0.33  0.83 -0.38  0.79 -0.71 -0.83
##  [97]  0.24 -0.27  0.05 -0.48
```

We want to find, which of the elements in the vector are those we want:

```
idx <- which(v1 > -0.5 & v1 < 0.5)
idx
```

```
##  [1]   1   3   8   9  11  13  14  15  18  19  20  22  24  25  26  27  28  29  30
## [20]  31  32  33  34  35  37  38  43  44  45  47  48  49  50  51  52  54  55  56
## [39]  59  60  62  63  64  65  66  67  69  70  72  73  75  76  78  80  81  83  84
## [58]  85  86  89  91  93  97  98  99 100
```

These are indices! It returns to the position of the vector that we want. So we just subset the vector:

```
v1[idx]
```

```
##  [1]  0.43  0.39  0.41 -0.19 -0.09  0.40 -0.06 -0.16 -0.33 -0.43 -0.33 -0.44
## [13]  0.11 -0.15  0.44 -0.25 -0.43 -0.38  0.06 -0.15  0.18 -0.20 -0.38  0.06
## [25]  0.45 -0.36  0.03  0.18  0.18  0.17 -0.08 -0.47  0.00  0.03 -0.45  0.16
## [37] -0.29  0.47 -0.21 -0.27  0.13 -0.39  0.25  0.07  0.20  0.36 -0.05  0.27
## [49]  0.34 -0.01  0.44 -0.38  0.38  0.06 -0.06  0.10  0.14 -0.23  0.22 -0.26
## [61] -0.33 -0.38  0.24 -0.27  0.05 -0.48
```

Basic ideas here are to first find which positions in the series are those we want. We set filtering conditions, combine with `which` function to get the positions, then just subset the full vector.

3.2 Matrix

Note that vectors are just one dimensional, we can also create a matrix or tensor(array) in R when we have more than one dimension. For example, we can create a 4 by 4 matrix:

```
matrix(1:16, nrow = 4, ncol = 4)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    5    9   13
## [2,]    2    6   10   14
## [3,]    3    7   11   15
## [4,]    4    8   12   16
```

Did you notice that the elements are assigned by column? we can add an argument `byrow = TRUE` in the matrix function, to assign values by row. We can also bind two matrices by row or column:

```
m1 <- matrix(1:16, nrow = 4, ncol = 4)
m2 <- matrix(17:32, nrow = 4, ncol = 4)
cbind(m1,m2)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## [1,]    1    5    9   13   17   21   25   29
## [2,]    2    6   10   14   18   22   26   30
## [3,]    3    7   11   15   19   23   27   31
## [4,]    4    8   12   16   20   24   28   32
```

```
rbind(m1,m2)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    5    9   13
## [2,]    2    6   10   14
## [3,]    3    7   11   15
## [4,]    4    8   12   16
## [5,]   17   21   25   29
## [6,]   18   22   26   30
## [7,]   19   23   27   31
## [8,]   20   24   28   32
```

In R, we have some functions to calculate the sum of the rows and columns too:

```
colSums(m1)
```

```
## [1] 10 26 42 58
```

```
rowSums(m1)
```

```
## [1] 28 32 36 40
```

More: In statistical analysis, we can do numeric matrix operations including multiplication, addition, inversion, transpose, eigen decomposition, determinant, and so on. Refer to: http://www.philender.com/courses/multivariate/notes/matr.html for more interesting operations. ### Subsetting a Matrix To subset a matrix, there are three possible ways using bracket:

```
m1
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    5    9   13
## [2,]    2    6   10   14
## [3,]    3    7   11   15
## [4,]    4    8   12   16
```

Subsetting a matrix by regarding it to be a vector: (Dangerous)

```
m1[1:5]
```

```
## [1] 1 2 3 4 5
```

Subsetting the specific row or column of the matrix:

```
m1[1,]
```

```
## [1]  1  5  9 13
```

```
m1[,1]
```

```
## [1] 1 2 3 4
```

Subsetting the specific some elements of the matrix:

```
m1[1,1]
```

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

```
m1[1,1:2]
```

```
## [1] 1 5
```

We can also use `which()` function to subset the matrix, this is left for an exercise for you.

3.3 Array

Similar to matrix, if we have more than two dimensions of the data, we need to use an array. It's also known as tensor in deep learning literature:

```
a1 <- array(1:8, dim = c(2,2,2))
a1
```

```
## , , 1
##
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
##
## , , 2
##
##      [,1] [,2]
## [1,]    5    7
## [2,]    6    8
```

It's like a 3D Lego now, right? We have each slices being a matrix, and put one slices above the other one. Subsetting here follows the similar rules, the only different thing from matrix is that now you need three dimensional coordinates to subset the ones we want.

3.4 Lists

A list is a collection of objects.

```
l1 <- list(1:3,
"a",
c(TRUE,FALSE,TRUE),
c(1,2)
)
l1
```

```
## [[1]]
## [1] 1 2 3
##
## [[2]]
## [1] "a"
##
## [[3]]
## [1]  TRUE FALSE  TRUE
##
## [[4]]
## [1] 1 2
```

```r
str(l1)
```

```
## List of 4
##  $ : int [1:3] 1 2 3
##  $ : chr "a"
##  $ : logi [1:3] TRUE FALSE TRUE
##  $ : num [1:2] 1 2
```

Lists work differently from c(), as they can contain objects of different types. We can also assign names to the lists for each vector in the list:

```r
names(l1) <- c("a", "b", "c", "d")
l1
```

```
## $a
## [1] 1 2 3
##
## $b
## [1] "a"
##
## $c
## [1]  TRUE FALSE  TRUE
##
## $d
## [1] 1 2
```

3.4.1 Subsetting a List

There are two ways to subset a list, by the order (just like atomic vectors), or by the name. But if we are going to subset by the order, double bracket (`[[]]`) is needed. To subset the list by name, we can use a dollar sign \texttt{\$}+name.

```r
l1[[1]]
```

```
## [1] 1 2 3
```

```r
l1$c
```

```
## [1]  TRUE FALSE  TRUE
```

3.5 Dataframe

In data analysis, the lists that we frequently use are data frames and tibbles. Data frames are the lists to store the data for analysis.

```
df1 <- data.frame(x = 1:3, y = c(TRUE,FALSE,FALSE))
df1
```

```
##   x     y
## 1 1  TRUE
## 2 2 FALSE
## 3 3 FALSE
```

```
typeof(df1)
```

```
## [1] "list"
```

```
str(df1)
```

```
## 'data.frame':    3 obs. of  2 variables:
##  $ x: int  1 2 3
##  $ y: logi  TRUE FALSE FALSE
```

```
attributes(df1)
```

```
## $names
## [1] "x" "y"
##
## $class
## [1] "data.frame"
##
## $row.names
## [1] 1 2 3
```

A dataframe can be converted to matrix, and it also works in the other direction:

```
a <- matrix(1:9, nrow = 3)
colnames(a) <- c("A", "B", "C")
a
```

```
##      A B C
## [1,] 1 4 7
## [2,] 2 5 8
## [3,] 3 6 9
```

```
is.data.frame(data.frame(a))
```

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

```
is.matrix(a)
```

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

3.5.1 Subsetting a dataframe

Either use the same way as subsetting the matrix, or using the dollar sign followed by the name of the colum that we want:

```
df_a <- data.frame(a)
df_a$A
```

```
## [1] 1 2 3
```

```
df_a[1,]
```

```
##   A B C
## 1 1 4 7
```

We can also use the bracket combined with the column name to subset a dataframe:

```
df_a["A"]
```

```
##   A
## 1 1
## 2 2
## 3 3
```

4 Ackowledgement

This teaching material is adapted from the previous material of this course made by Marcela Alfaro-Córdoba and Sheng Jiang.