STATS 266 Handout - Data Structures & Manipulation

Qi Wang

2025-04-07

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
```

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

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

```r
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?

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

```
##   [1] -0.87 -0.11 -0.67 -0.84 -0.11 -0.37  0.38 -0.14  0.38  0.55  0.52 -0.53
##  [13]  0.47 -0.48  0.03  0.48 -1.00  0.01 -0.45 -0.29 -0.32 -0.58  0.16 -0.06
##  [25] -0.57 -0.01  0.16  0.25 -0.32  0.45 -0.45 -0.32 -0.01  0.32  0.84  0.34
##  [37]  1.02  0.75 -0.37  0.07 -0.59  0.10  0.81 -0.47 -0.52  0.12  0.04 -0.35
##  [49]  0.00  0.37  0.08 -0.35 -0.30  0.12 -0.13  0.28 -0.02 -0.33  0.07 -0.41
##  [61]  0.06  0.32  0.05 -0.39  0.93 -0.28  1.03 -0.30  0.30 -0.11 -0.56 -0.38
##  [73]  0.71  0.91  0.89  0.13 -0.67  0.21 -0.36 -1.23  0.76  0.18 -0.65  0.20
##  [85]  0.19  0.31  0.17  0.61 -0.32  1.37 -0.22 -0.13 -0.23 -0.51  0.10 -0.66
##  [97] -0.39 -0.68 -0.80  0.24
```

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

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

```
##  [1]   2   5   6   7   8   9  13  14  15  16  18  19  20  21  23  24  26  27  28
## [20]  29  30  31  32  33  34  36  39  40  42  44  46  47  48  49  50  51  52  53
## [39]  54  55  56  57  58  59  60  61  62  63  64  66  68  69  70  72  76  78  79
## [58]  82  84  85  86  87  89  91  92  93  95  97 100
```

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

```r
v1[idx]
```

```
##  [1] -0.11 -0.11 -0.37  0.38 -0.14  0.38  0.47 -0.48  0.03  0.48  0.01 -0.45
## [13] -0.29 -0.32  0.16 -0.06 -0.01  0.16  0.25 -0.32  0.45 -0.45 -0.32 -0.01
## [25]  0.32  0.34 -0.37  0.07  0.10 -0.47  0.12  0.04 -0.35  0.00  0.37  0.08
## [37] -0.35 -0.30  0.12 -0.13  0.28 -0.02 -0.33  0.07 -0.41  0.06  0.32  0.05
## [49] -0.39 -0.28 -0.30  0.30 -0.11 -0.38  0.13  0.21 -0.36  0.18  0.20  0.19
## [61]  0.31  0.17 -0.32 -0.22 -0.13 -0.23  0.10 -0.39  0.24
```

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.

Sometimes we need to sort the elements in the vector, we can use function `sort` or `order`:

```
x <- c(10,50,30,20,40)
sort(x)
```

```
## [1] 10 20 30 40 50
```

```
order(x)
```

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

The function `sort()` returns the sorted data. By adding an argument `decreasing = TRUE`, it returns the sorted data in decreasing order. The `order()` function returns to the order of the original data after arranging them in increasing or decreasing order.

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. If we want to return the matrix index instead of regarding the matrix to be a vector:

```
which(m1%%2==0, arr.ind = TRUE)
```

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

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
```

```
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:

```
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.

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

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

```
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
```

```r
typeof(df1)
```

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

```r
str(df1)
```

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

```r
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:

```r
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
```

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

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

```r
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:

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

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

```r
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:

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

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

4 Subsetting Dataframe with dplyr

Although `which` function is powerful enough to filter the data, there are some more convenient ways. For example, if you have used SQL before, I believe you will be not so used to using `which` function. The **`dplyr` package** provides powerful functions for **subsetting and filtering** data efficiently. The following cases can be easily done via `dplyr`.

- Selecting specific columns
- Filtering rows based on conditions
- Using logical operators for subsetting
- Extracting rows by position
- Combining multiple conditions

We will use the **`mtcars` dataset** as an example.

4.1 Load Data and `dplyr` Package

```r
# Load necessary libraries
library(dplyr)

# Load dataset
data(mtcars)

# View first few rows
head(mtcars)
```

```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

4.2 Selecting Specific Columns

Usually, a dataset will have column names for each column, like SQL grammer, we can "select" some columns from the dataframe.

4.2.1 Selecting One or More Columns

```
# Select specific columns
mtcars_selected <- mtcars %>%
  select(mpg, hp, wt)

# View first few rows
head(mtcars_selected)
```

```
##                    mpg  hp    wt
## Mazda RX4         21.0 110 2.620
## Mazda RX4 Wag     21.0 110 2.875
## Datsun 710        22.8  93 2.320
## Hornet 4 Drive    21.4 110 3.215
## Hornet Sportabout 18.7 175 3.440
## Valiant           18.1 105 3.460
```

4.2.2 Excluding Specific Columns

```
mtcars_excluded <- mtcars %>%
  select(-hp, -wt)

# View first few rows
head(mtcars_excluded)
```

```
##                    mpg cyl disp drat  qsec vs am gear carb
## Mazda RX4         21.0   6  160 3.90 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 3.90 17.02  0  1    4    4
## Datsun 710        22.8   4  108 3.85 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 3.08 19.44  1  0    3    1
```

```
## Hornet Sportabout 18.7   8  360 3.15 17.02  0  0    3    2
## Valiant           18.1   6  225 2.76 20.22  1  0    3    1
```

Note the pipe function "%>%" in R, provided by the magrittr package (included in dplyr), is used to make code more readable and structured by passing the result of one function directly into the next function. The basic syntax is:

```
#data %>% function1() %>% function2() %>% function3()
```

and this is equivalent to

```
#function3(function2(function1(data)))
```

The pipe function pass the output from the previous function to the next function as the first argument. In a general case:

```
double <- function(x) {
  x * 2
}

# Apply the function using a pipe
10 %>% double()  # Output: 20
```

```
## [1] 20
```

4.3 Filtering Rows with filter()

The `filter()` function is used to select rows based on conditions. ### Filtering for a Single Condition

```
# Cars with mpg greater than 20
mtcars_filtered <- mtcars %>%
  filter(mpg > 20)

# View first few rows
head(mtcars_filtered)
```

```
##                 mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4      21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag  21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710     22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Merc 240D      24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230       22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
```

4.3.1 Filtering with Multiple Conditions

```
# Cars with mpg > 20 and hp < 100
mtcars_filtered2 <- mtcars %>%
  filter(mpg > 20, hp < 100)

# View first few rows
head(mtcars_filtered2)
```

```
##                 mpg cyl  disp hp drat    wt  qsec vs am gear carb
## Datsun 710     22.8   4 108.0 93 3.85 2.320 18.61  1  1    4    1
## Merc 240D      24.4   4 146.7 62 3.69 3.190 20.00  1  0    4    2
## Merc 230       22.8   4 140.8 95 3.92 3.150 22.90  1  0    4    2
## Fiat 128       32.4   4  78.7 66 4.08 2.200 19.47  1  1    4    1
## Honda Civic    30.4   4  75.7 52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla 33.9   4  71.1 65 4.22 1.835 19.90  1  1    4    1
```

4.3.2 Using OR (|) Condition

```
# Cars with mpg > 25 OR hp > 150
mtcars_filtered3 <- mtcars %>%
  filter(mpg > 25 | hp > 150)

# View first few rows
head(mtcars_filtered3)
```

```
##                   mpg cyl  disp  hp drat   wt  qsec vs am gear carb
## Hornet Sportabout  18.7   8 360.0 175 3.15 3.44 17.02  0  0    3    2
## Duster 360         14.3   8 360.0 245 3.21 3.57 15.84  0  0    3    4
## Merc 450SE         16.4   8 275.8 180 3.07 4.07 17.40  0  0    3    3
## Merc 450SL         17.3   8 275.8 180 3.07 3.73 17.60  0  0    3    3
## Merc 450SLC        15.2   8 275.8 180 3.07 3.78 18.00  0  0    3    3
## Cadillac Fleetwood 10.4   8 472.0 205 2.93 5.25 17.98  0  0    3    4
```

4.4 Filtering Rows by Position with slice()

The slice() function extracts rows by position. Similar to the bracket, if we want to extract the first 5 rows:

```
mtcars_first5 <- mtcars %>%
  slice(1:5)

# View result
mtcars_first5
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
```

If we want to slice the last five rows:

```
mtcars_last5 <- mtcars %>%
  slice_tail(n = 5)

# View result
mtcars_last5
```

```
##                 mpg cyl  disp  hp drat    wt qsec vs am gear carb
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.9  1  1    5    2
## Ford Pantera L 15.8   8 351.0 264 4.22 3.170 14.5  0  1    5    4
## Ferrari Dino   19.7   6 145.0 175 3.62 2.770 15.5  0  1    5    6
## Maserati Bora  15.0   8 301.0 335 3.54 3.570 14.6  0  1    5    8
## Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.6  1  1    4    2
```

We can also slice random five rows:

```
mtcars_random <- mtcars %>%
  slice_sample(n = 5)  # Randomly selects 5 rows

# View result
mtcars_random
```

```
##                   mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Honda Civic       30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Pontiac Firebird  19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Volvo 142E        21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
## Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
```

The filter() function is used to select rows based on conditions.

4.5 Using Logical Operators in filter()

We can use logical operators for complex filtering. ### Using %in% for Matching Multiple Values

```r
# Cars with 4 or 6 cylinders
mtcars_subset <- mtcars %>%
  filter(cyl %in% c(4, 6))

# View result
head(mtcars_subset)
```

```
##                 mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4      21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag  21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710     22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Valiant        18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Merc 240D      24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
```

This code extracts rows where cyl is 4 or 6.

4.5.1 Filtering with NOT (!=)

```r
# Cars that are NOT 8 cylinders
mtcars_not8 <- mtcars %>%
  filter(cyl != 8)

# View result
head(mtcars_not8)
```

```
##                 mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4      21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag  21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710     22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Valiant        18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Merc 240D      24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
```

4.6 Combining Multiple Subsetting Methods

We can combine select(), filter(), and slice() for complex subsetting. If we extract cars with mpg $>$ 20, Select Only mpg and hp, and get first 5 rows:

```r
mtcars_combined <- mtcars %>%
  filter(mpg > 20) %>%
  select(mpg, hp) %>%
  slice(1:5)
```

```r
# View result
mtcars_combined
```

```
##                  mpg  hp
## Mazda RX4       21.0 110
## Mazda RX4 Wag   21.0 110
## Datsun 710      22.8  93
## Hornet 4 Drive 21.4 110
## Merc 240D       24.4  62
```

It's kind of similar to the SQL grammer, where you select the columns and filter the colums by some criterias.

5 Ackowledgement

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