R language basics, part 1

HUST Bioinformatics course series

Wei-Hua Chen (CC BY-NC 4.0)

02 September, 2024

section 1: outline

TOC

注：不一定按照上述顺序

section 2: R basics - 基本数据类型

基本数据类型

最基本的数据类型包括**数字**和**字符串**，是其它数据类型的基本组成部分。

数字

```
## 整数
287
```

```
## [1] 287
```

```
## 小数
99.99
```

```
## [1] 99.99
```

```
## 科学计数法
1e-3
```

```
## [1] 0.001
```

```
1e2
```

```
## [1] 100
```

逻辑符号

真

```
TRUE
T
```

假

```
FALSE
F
```

其本质是数字

```
1 + TRUE
```

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

```
2 * FALSE
```

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

字符串

字符串则是可以是任何字符的组合，由单引号或双引号包括。比如：

```
'a sentence' ## 单括号
" 一个字符串" ## 双括号
'1.123'      ## 像是数字的字符串
'*%%*()!@##&@(9'   ## 乱码
```

简单数据类型

简单数据类型包括 vector 和矩阵，它们都可以包含某一种基本数据类型的多个数值，比如由多个数字组成的矩阵，多个字符串组成的 vector 等。但它们**只能包含单一数据类型**；这一点稍后会有解释。

```
c(100, 20, 30) ## 整数 vector
c(" 字符串", " 数组"," 是我") ## 字符串 vector
c(TRUE, FALSE, TRUE, T, F) ## 一个逻辑 vector
```

如上所示，数组通常用函数 c() 来定义。除此之外，还可以用 ':' 操作符号来定义包含连续整数的 vector：

```
2:8
```

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

vector 的数据类型转换规则

vector 只能包含一种基本数据类型。因此，在定义数组时，如果输入的数值是混合的，那么某些基本数据类型会自动转换为其它类型，以保证数值类型的一致性；这在英文里称为 coerce，有强制转换的意思。这种转换的优先级为：

- 逻辑类型 -> 数字类型
- 逻辑类型 -> 字符串
- 数字类型 -> 字符串

我们可以用 `class()` 或 `str()` 函数来判断 vector 包含的数据类型。以后会介绍两者的不同。

vector 的数据类型转换规则

```
class( c(45, TRUE, 20, FALSE, -100) ); ## 逻辑和数字类型
```

```
## [1] "numeric"
```

```
str( c("string a", FALSE, "string b", TRUE) ); ## 逻辑和字符
```

```
##  chr [1:4] "string a" "FALSE" "string b" "TRUE"
```

```
str( c("a string", 1.2, "another string", 1e-3) ); ## 数字和字符
```

```
##  chr [1:4] "a string" "1.2" "another string" "0.001"
```

矩阵 (matrix)

矩阵也可看做是一种带有限制的二维数组，其限制是，矩阵内的数值必须是同一种类型。当输入混合有多种基本数据类型时，矩阵会按上面提到的规则进行强制转换。

矩阵由函数 matrix() 定义，比如：

```
matrix( c(20, 30.1, 2, 45.8, 23, 14), nrow = 2, byrow = T );
```

```
##      [,1] [,2] [,3]
## [1,] 20.0 30.1    2
## [2,] 45.8 23.0   14
```

Matrix 函数的参数

```
?matrix
```

matrix() 参数测试

```
matrix( c(20, 30.1, 2, 45.8, 23, 14), nrow = 2, byrow = T );
```

```
##      [,1] [,2] [,3]
## [1,] 20.0 30.1    2
## [2,] 45.8 23.0   14
```

```
matrix( c(20, 30.1, 2, 45.8, 23, 14), nrow = 2, byrow = F );
```

```
##      [,1] [,2] [,3]
## [1,] 20.0  2.0   23
## [2,] 30.1 45.8   14
```

```
matrix( c(20, 30.1, 2, 45.8, 23, 14), nrow = 2,
       dimnames = list( c("row_A", "row_B"), c("A", "B", "C") ) );
```

```
##           A    B  C
## row_A 20.0  2.0 23
## row_B 30.1 45.8 14
```

matrix() 同时指定 ncol 和 nrow

矩阵的指定长度，即 nrow × ncol，可以不同于输入数据的长度。矩阵长度较小时，输入数据会被截短；而矩阵长度较大时，输入数据则会被重复使用。举例如下：

```
## 生成一个 2x5 长度为 10 的矩阵，但输入数据的长度为 20
matrix( 1:20, nrow = 2, ncol = 5, byrow = T);
```

```
## Warning in matrix(1:20, nrow = 2, ncol = 5, byrow = T): data length differs
## from size of matrix: [20 != 2 x 5]
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3    4    5
## [2,]    6    7    8    9   10
```

```
## 生成一个 2x3 长度为 6 的矩阵，但输入数据长度只有 3
matrix( 1:3, nrow = 2, ncol = 3, byrow = T );
```

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

matrix() 更多测试

注意：上面两种情况下，系统并不会给出任何提示。但下面两种情况，系统会报警告信息。第一种情况，矩阵长度大于输入数据长度，且前者不是后者的整数倍。

```
matrix( 1:3, nrow = 2, ncol = 4, byrow = T );
```

```
## Warning in matrix(1:3, nrow = 2, ncol = 4, byrow = T): data length [3] is not a
## sub-multiple or multiple of the number of rows [2]
```

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

matrix() 更多测试，cont.

第二种情况，矩阵长度小于输入数据的长度，且后者不是前者的整数倍。

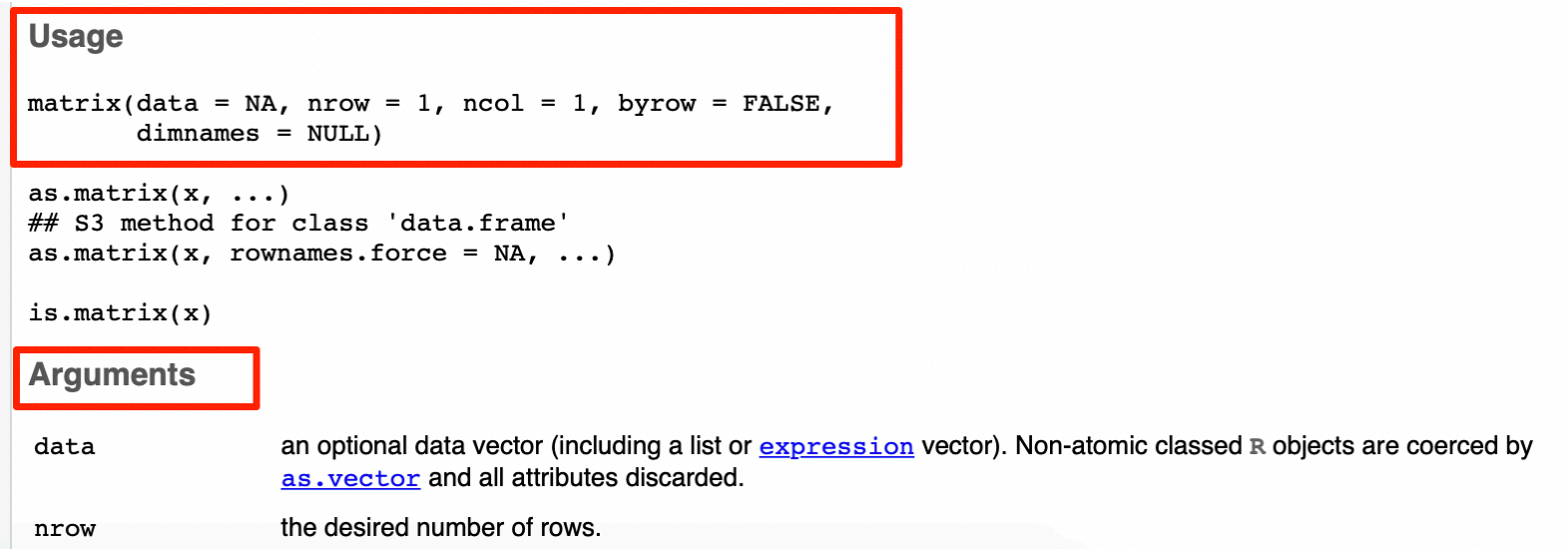
```
matrix( letters[1:20], nrow = 3, ncol = 5, byrow = T );
```

```
## Warning in matrix(letters[1:20], nrow = 3, ncol = 5, byrow = T): data length
## [20] is not a sub-multiple or multiple of the number of rows [3]
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,] "a"  "b"  "c"  "d"  "e"
## [2,] "f"  "g"  "h"  "i"  "j"
## [3,] "k"  "l"  "m"  "n"  "o"
```

用? matrix 命令获得帮助

在 Console 窗口中输入 ? matrix 命令，可在窗口 4 获得有关 matrix() 函数的帮助信息：



Usage

```
matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE,
       dimnames = NULL)

as.matrix(x, ...)
## S3 method for class 'data.frame'
as.matrix(x, rownames.force = NA, ...)

is.matrix(x)
```

Arguments

data	an optional data vector (including a list or expression vector). Non-atomic classed R objects are coerced by as.vector and all attributes discarded.
nrow	the desired number of rows.

Figure 1: matrix() 函数使用方法

class() 和 str() 的区别

```
class(matrix( c(20, 30.1, 2, 45.8, 23, 14),
              nrow = 2, byrow = T ));
```

```
## [1] "matrix" "array"
```

```
str(matrix( c(20, 30.1, 2, 45.8, 23, 14),
            nrow = 2, byrow = T ));
```

```
##  num [1:2, 1:3] 20 45.8 30.1 23 2 14
```

str() 更多功用以后会详细介绍

数组（arrray）

vector 和矩阵都是数组。vector 是一维数组，矩阵是二维数组。这就意味着：a) 还可以有更多维的数组，b) 高维数组与 vector 和矩阵一样，只能包含一种基本数据类型。高维数组可以由函数 array() 定义：

```
array( data = LETTERS[1:16],
       dim  = c(2,4,2),
       dimnames = list( c("A","B"),
                        c("one","two","three","four"),
                        c(" 一", " 二") ));
```

```
## , , 一
##
##   one two three four
## A "A" "C" "E"   "G"
## B "B" "D" "F"   "H"
##
## , , 二
##
##   one two three four
## A "I" "K" "M"   "O"
## B "J" "L" "N"   "P"
```

section 3: R basics - 简单算术

运算符

加减乘除老一套

```
1 + 2 - 3 * 4 / 5; ## 加减乘除
1 + (2 - 3) * 4 / 5;  ##  改变优先级
```

进阶操作

```
2 ^ 6; ## 阶乘
5 %% 2; ## 取余
```

逻辑运算符

```
T | F; ## or
T & F; ## and
5 | 0; ## == 0 FALSE,  > 0 TRUE
```

? 5 & -1 结果应该是 TRUE 还是 FALSE ???

数学函数

- log, log2, log10
- exp
- sin, cos, tan
- sqrt

变量与赋值

在进行下一步之前，先介绍变量与赋值的基本知识

在 R 中，可以用任何下面的方式**赋值**给**变量**

```
## 常用
x <- c(10,100,1000, 10000);
## -- 也可以;
x = c(10,100,1000, 10000);
## -- 少见
c(10,100,1000, 10000) -> x;
## -- ???
assign( "x", c(10, 100, 1000, 10000) );
```

其中 x 就叫做变量 **注**：赋值后，变量内容默认不再打印到 console **注 2**: c() 又叫做 **concatenation function** （以后会再次介绍）

变量的管理

现在我们运行多个赋值操作:

```
x <- c(10,100,1000, 10000);
m <- matrix( c(20, 30.1, 2, 45.8, 23, 14), nrow = 2,
       dimnames = list( c("row_A", "row_B"), c("A", "B", "C") ) );
```

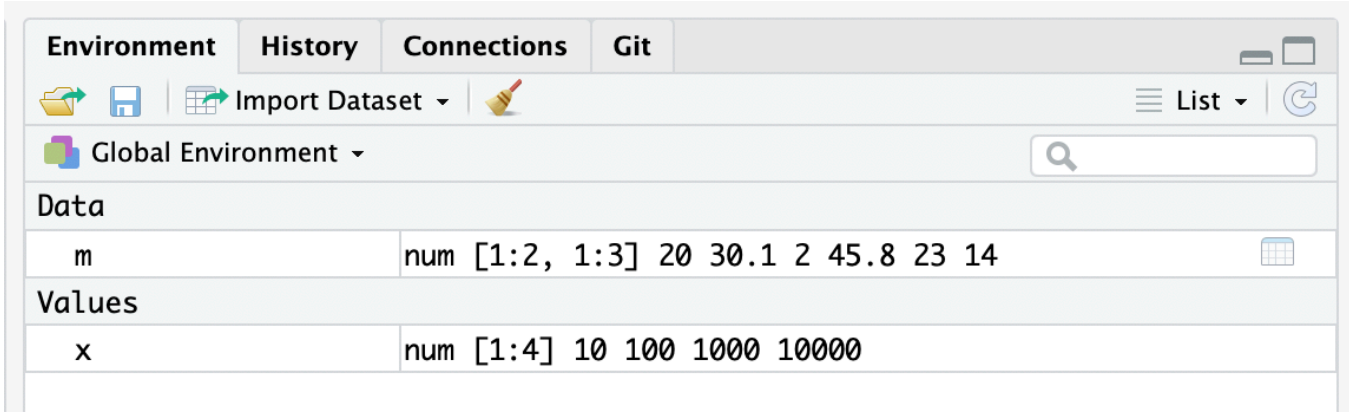


Figure 2: RStudio enviroment window

通过 Console window 管理变量

```
ls();  ## 显示当前环境下所有变量
```

```
## [1] "color_block" "m"           "x"
```

```
rm( x ); ## 删除一个变量
ls();
```

```
## [1] "color_block" "m"
```

```
##rm(list=ls()); ## 删除当前环境下所有变量!!!
```

vector 算术, cont.

vectorisation：R 最重要的一个概念

```
x <- c(10,100,1000, 10000);
( y <- sqrt( x  ) * 4 + 10  ); ## 赋值的之后打印变量内容
```

```
## [1]  22.64911  50.00000 136.49111 410.00000
```

vectorisation 的核心在于数据自动循环使用

```
x / c(10,100);
```

```
## [1]   1   1 100 100
```

```
x / c(10,100,1000); ## 会报错但仍会循环计算
```

```
## Warning in x/c(10, 100, 1000): longer object length is not a multiple of
## shorter object length
```

```
## [1]    1    1    1 1000
```

matrix 的算术

先查看 m 的内容：

```
m;
```

```
##          A    B  C
## row_A 20.0  2.0 23
## row_B 30.1 45.8 14
```

```
m / 10;
```

```
##          A    B   C
## row_A 2.00 0.20 2.3
## row_B 3.01 4.58 1.4
```

```
m /  c(1,10,100);
```

```
##           A     B    C
## row_A 20.00  0.02 2.30
## row_B  3.01 45.80 0.14
```

matrix 的算术, cont.

```
m / c(1,10);
```

```
##           A    B    C
## row_A 20.00 2.00 23.0
## row_B  3.01 4.58  1.4
```

```
m / c(1,10,100,1000); ## 多于列数
```

```
## Warning in m/c(1, 10, 100, 1000): longer object length is not a multiple of
## shorter object length
```

```
##           A      B    C
## row_A 20.00 0.0200 23.0
## row_B  3.01 0.0458  1.4
```

更多 matrix 相关函数

注：其中许多也要用于其它数据类型（以后会讲到）

- dim(m);
- nrow(m);
- ncol(m);
- range(m); ## 内容是数字时
- summary(m); ## 也可用于 vector

vector manipulation

合并

```
a <- 1:3;
b <- LETTERS[1:3];

( ab <- c(a,b) );
```

```
## [1] "1" "2" "3" "A" "B" "C"
```

```
mode( ab ); ## 一个新的函数 ~ ...
```

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

vector manipulation, cont.

取部分

```
ab[1];
```

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

```
ab[ c(1, 4, 5) ];
```

```
## [1] "1" "A" "B"
```

```
ab[ 2:5 ];
```

```
## [1] "2" "3" "A" "B"
```

```
ab[ length(ab) ]; ## length 函数
```

```
## [1] "C"
```

vector manipultaion, cont.

替换单个值

```
( ab[1] <- " —" );
```

```
## [1] "—"
```

vector manipulation, cont.

替换多个值

```
ab[c(2,3)] <- c(" 二", " 三");
ab
```

```
## [1] "一" " 二" " 三" "A"  "B"  "C"
```

vector manipulation, cont.

naming elements & then replace a value

```
( names( ab ) <- as.character( ab ) ); ## 注意 names() 和 as.character() 函数的用法
```

```
## [1] "一" "二" "三" "A"  "B"  "C"
```

```
ab[ c("A", "B") ] <- c("ah", "bo");

ab;
```

```
##   一   二   三   A   B   C
## "一" "二" "三" "ah" "bo" "C"
```

vector manipulation, cont.

other useful functions

```
## reverse
rev(1:10);
```

```
##  [1] 10  9  8  7  6  5  4  3  2  1
```

```
## sort  & order
lts <- sample( LETTERS[1:20] );
sort( lts );
```

```
##  [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S"
## [20] "T"
```

```
order( lts  ); ## 注意与 sort 的不同
```

```
##  [1] 10  3  4 12  2 17 11  1 14 20 18 13 16 15 19  7  9  6  8  5
```

matrix manipulation

取一行、多行

```
m;
```

```
##          A    B  C
## row_A 20.0  2.0 23
## row_B 30.1 45.8 14
```

```
m[ 1 , ];
```

```
##  A  B  C
## 20  2 23
```

```
m[ 1:2,  ];
```

```
##          A    B  C
## row_A 20.0  2.0 23
## row_B 30.1 45.8 14
```

matrix manipulation, cont.

取一行、多行, cont.

```
m[ "row_A", ]
```

```
##  A  B  C
## 20  2 23
```

```
m[ c( "row_B", "row_A" ), ] ## 注意取的顺序
```

```
##          A    B  C
## row_B 30.1 45.8 14
## row_A 20.0  2.0 23
```

matrix manipulation, cont.

取一列、多列

```
m[ , 1 ];
```

```
## row_A row_B
##  20.0  30.1
```

```
m[, c(1,2)];
```

```
##           A    B
## row_A 20.0  2.0
## row_B 30.1 45.8
```

```
m[, c("B","A")];## 注意取的顺序
```

```
##           B    A
## row_A  2.0 20.0
## row_B 45.8 30.1
```

matrix manipulation, cont.

取部分

```
m;
```

```
##           A    B  C
## row_A 20.0  2.0 23
## row_B 30.1 45.8 14
```

```
m[1:2, 2:3]; ## 取其中一部分
```

```
##           B  C
## row_A  2.0 23
## row_B 45.8 14
```

matrix manipulation, cont.

替换一行

```
m;
```

```
##           A    B  C
## row_A 20.0  2.0 23
## row_B 30.1 45.8 14
```

```
m[1, ] <- c(10);
```

matrix manipulation, cont.

替换一列

```
m[, "C"] <- c(230, 140);
m;
```

```
##           A    B   C
## row_A  10.0 10.0 230
## row_B  30.1 45.8 140
```

matrix manipulation, cont.

替换多行、多列

```
## -- 替换前两行
m[ 1:2 , ] <- matrix(  1:6, nrow = 2 );

## -- 替换其中两列
m[  , c("C", "B") ] <- matrix(  100:103, nrow = 2 );
```

替换 subset

```
m[ 1  , c("C", "B") ] <- matrix(  110:111, nrow = 1 );
```

转置

```
t(m);
```

```
##   row_A row_B
## A    10  30.1
## B    10  45.8
## C   230 140.0
```

section 3: 特別値

数据类型汇总

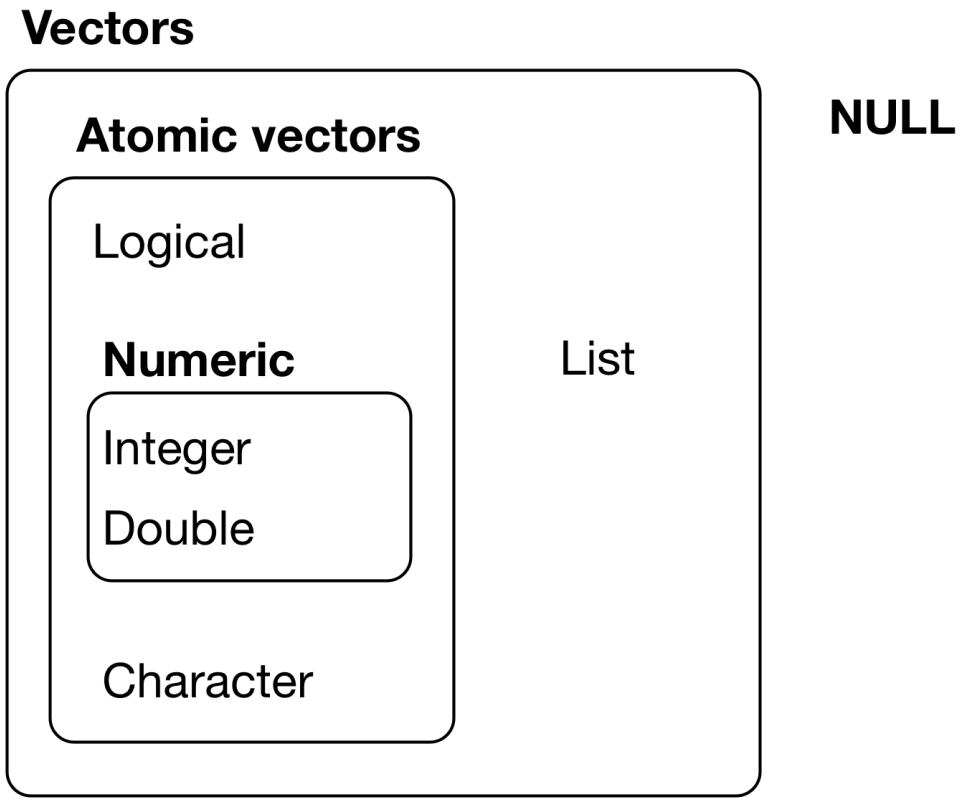


Figure 3: R 的数据类型及相互关系

图片出自：`https://r4ds.had.co.nz/vectors.html`

用 typeof() 函数确定以上类型

```
typeof(letters);
```

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

```
typeof(1:10);
```

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

```
typeof( list("a", "b", 1:10) );
```

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

```
typeof( 1:10 %% 3 == 0 );
```

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

```
typeof(1);
```

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

```
typeof(1L);
```

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

NULL

NULL: expressions and functions whose value is undefined

```
typeof( NULL );
```

```
## [1] "NULL"
```

特别值以及它们是如何产生的

一次认识所有：

```
c(-1, 0, 1) / 0;
```

```
## [1] -Inf  NaN  Inf
```

```
NA
```

注意 NULL 既是特别值也是特别数据类型

判定特別值

以下判定为 TRUE:

```
nul <- readr::read_csv(file = "data/talk02/null.csv");
nul;
```

```
## # A tibble: 4 x 5
##   `function`     `0`   `-Inf` `NA`  `NaN`
##   <chr>          <chr> <chr>  <chr> <chr>
## 1 is.finite()    x     <NA>   <NA>  <NA>
## 2 is.infinite()  <NA>  x      <NA>  <NA>
## 3 is.na()        <NA>  x      x     <NA>
## 4 is.nan()       <NA>  <NA>   <NA>  x
```

其它 is. 函数

```
is.null( NULL );
is.numeric( NA );
is.numeric( Inf );

## 用于替代 typeof 的函数
is.list();
is.logical();
is.character();
is.vector();

## 更多 ...
```

section 4: 简单回顾

今日内容回顾

R 基本数据类型

- 数字和字符串
- 整数、小数、逻辑
- 数据类型之间转换；自动规则
- `matrix`

R 数据类型的查看方式

- `mode`
- `class`
- `typeof`
- `is.xxx` 函数们

基本数据类型的操作

- 定义
- 取部分

section 5: 练习 & 作业

练习 & 作业

- Exercises and homework 目录下 talk02-homework.Rmd 文件
- 完成时间：见钉群的要求