L2: R vector

Nan He, School of Basic Medical Sciences, Southern Medical University

2026-01-18

目录

R vector introduction

1. 引言

在生信分析中，我们 90% 的时间都在和数据打交道。理解 R 的数据结构，是写出高效、可复现代码的基石。R 的常见数据结构包括：向量（Vector）、矩阵（Matrix）、列表（List）和数据框（Data Frame）。而向量是一切数据结构的基础，比如数据分析最常见的数据框 (表格) 文件，每一列就可以看作是一个特定类型的向量，所以掌握对向量的操作是很重要的。本节重点讲解向量：如何创建、查看、索引、运算、处理缺失值，以及常见的向量相关函数。

2. 向量

向量是 R 中最基本的原子结构，必须包含同一类型的数据（要么全是数字，要么全是字符）。

2.1 向量的类型与长度

向量的类型可以用 `typeof()` 来求解，长度则是 `length()`：

```r
x_num <- c(1, 2, 3)
x_chr <- c("a", "b", "c")
x_lgl <- c(TRUE, FALSE, TRUE)

typeof(x_num)
```

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

```r
typeof(x_chr)
```

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

```r
typeof(x_lgl)
```

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

```r
length(x_num)
```

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

3. 创建向量

向量的创建是最常用的操作之一。下面按“用途”归纳常见函数。

3.1 直接拼接

最简单就是用 `c()` 函数来进行向量的生成:

```r
v0 <- c(3, 1, 2, 4)
v0
```

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

```r
## 也可以设置混合类型
v00 <- c(3, "c", "FALSE")
v00
```

```
## [1] "3"     "c"     "FALSE"
```

```r
typeof(v00) # 统一成字符类型
```

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

3.2 序列生成

这里介绍一些简单的序列生成函数，可以动手自己运行来体会每个函数的用法。

```r
v1 <- 1:10 # 生成 1 到 10
v2 <- seq(1, 10, by = 2) # by 为步长，即间隔长度
v3 <- seq(1, 10, length.out = 5) # 5 等分

v4 <- seq_len(10)                  # 用于循环计数
v5 <- seq_along(c(3, 1, 2, 4))     # 用于索引

v1; v2; v3; v4; v5
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
## [1] 1 3 5 7 9
## [1]  1.00  3.25  5.50  7.75 10.00
##  [1]  1  2  3  4  5  6  7  8  9 10
## [1] 1 2 3 4
```

3.3 重复生成

```r
rep(1, 10)
```

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

```r
rep(1:3, times = 2)  # 1 2 3 1 2 3
```

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

```r
rep(1:3, each = 2)   # 1 1 2 2 3 3
```

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

3.4 字符向量拼接

如果需要进行字符串的拼接，比如生成 gene1，gene2 等，一个一个打会很麻烦，可以考虑使用：

```r
v_char1 <- paste("char", 1:10, sep = "")
v_char2 <- paste("gene", 1:5, sep = "_")
v_char3 <- paste0("cell", 1:5)
v_char4 <- sprintf("char%02d", 1:10)  # 01, 02, ... 10

v_char1; v_char2; v_char3; v_char4
```

```
##  [1] "char1"  "char2"  "char3"  "char4"  "char5"  "char6"  "char7"  "char8"
##  [9] "char9"  "char10"
```

```
## [1] "gene_1" "gene_2" "gene_3" "gene_4" "gene_5"
```

```
## [1] "cell1" "cell2" "cell3" "cell4" "cell5"
```

```
##  [1] "char01" "char02" "char03" "char04" "char05" "char06" "char07" "char08"
##  [9] "char09" "char10"
```

3.5 分割字符串

注意：strsplit() 返回的是列表（因为每个字符串分割后可能长度不同）。

```
v_split <- strsplit("a,b,c", ",")
v_split
```

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

```
# 取出第一个元素
v_split[[1]]
```

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

3.6 随机向量

因为向量是随机生成的，为了保证可复现性，引入随机种子来保证每次运行的结果是一致的：

```
set.seed(2026) # 生成随机数种子

runif(5)               # 均匀分布
```

```
## [1] 0.6986735 0.5565305 0.1401400 0.2857233 0.5553690
```

```
rnorm(5, 0, 1)         # 正态分布
```

```
## [1] -1.9577250  1.0848713  0.2039567  0.5011464  1.0301596
```

```
rbinom(10, 1, 0.3)      # 二项分布（0/1）
```

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

```
## 放回抽样
sample(1:10, 8, replace = TRUE)
```

```
## [1]  2  5  8  5  3  3  3 10
```

```
## 不放回抽样
sample(1:10, 8, replace = FALSE)
```

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

4. 查看与检索向量

有时候一个向量很长，不可能全部看完。这时候可以通过观察这个向量的一些基本特征，从而来了解向量元素的分布：

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

v
```

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

```
head(v, 3) # 前 3 个
```

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

```
tail(v, 2) # 后 2 个
```

```
## [1] 40 50
```

```
length(v) # 长度
```

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

```
typeof(v) # 精确类型
```

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

```
class(v) # 粗糙类型
```

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

```
str(v) # 结构
```

```
##  num [1:5] 10 20 30 40 50
```

5. 索引与子集

R 中向量索引主要有三种：位置索引、逻辑索引、名称索引。

5.1 位置索引

即通过位置来取出向量的数据。

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

v[1]
```

```
## [1] 10
```

```r
v[2:4]
```

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

```r
v[c(1, 3, 5)]
```

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

```r
v[-1]      # 去掉第一个
```

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

```r
v[-c(2, 4)]
```

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

5.2 逻辑索引

通常我们需要取出满足一定条件的向量中的元素，这个时候就需要使用逻辑索引：

```r
v <- c(10, 20, 30, 40, 50)

v[v > 25] # 取出大于 25 的元素
```

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

```r
v[v %% 20 == 0]  # 取出能被 20 整除的元素
```

```
## [1] 20 40
```

5.3 名称索引

给向量加上 names 后，可以用名字索引，非常直观。

```r
v_named <- c(A = 10, B = 20, C = 30)
v_named
```

```
##  A  B  C
## 10 20 30
```

```r
names(v_named)
```

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

```r
v_named["B"]
```

```
##  B
## 20
```

```r
v_named[c("A", "C")]
```

```
##  A  C
## 10 30
```

6. 向量运算

R 中关于向量的很多运算默认是逐元素（element-wise），无需写循环就能高效计算。

```r
a <- 1:5
b <- 6:10

a + b
```

```
## [1]  7  9 11 13 15
```

```r
a * 2
```

```
## [1]  2  4  6  8 10
```

```r
a^2
```

```
## [1]  1  4  9 16 25
```

```r
sqrt(a)
```

```
## [1] 1.000000 1.414214 1.732051 2.000000 2.236068
```

```r
mean(a)
```

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

```r
sum(a)
```

```
## [1] 15
```

```r
sd(a)
```

```
## [1] 1.581139
```

6.1 广播

当两个向量长度不同，短向量会被“回收”重复使用：

```r
1:6 + c(10, 100)
```

```
## [1]  11 102  13 104  15 106
```

这里后面的向量只有两个元素，而前面的向量有 6 个元素，所以短向量会被循环使用，即先自我复制，从而使两个向量的长度相等，再相加。

7. 缺失值处理

生信数据中 NA 很常见（测序深度不足、检测不到表达等）。很多函数默认遇到 NA 会返回 NA，需要显式处理。

```
x <- c(1, 2, NA, 4, NA, 6)

is.na(x) # 返回每个元素是否为 NA 的逻辑值
```

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

```
sum(is.na(x)) # 返回该向量中为 NA 的元素个数
```

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

```
mean(x)                 # 会得到 NA
```

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

```
mean(x, na.rm = TRUE)   # 忽略 NA
```

```
## [1] 3.25
```

```
x[!is.na(x)]            # 去掉 NA
```

```
## [1] 1 2 4 6
```

```
na.omit(x)
```

```
## [1] 1 2 4 6
## attr(,"na.action")
## [1] 3 5
## attr(,"class")
## [1] "omit"
```

8. 常用向量的处理函数

8.1 排序与去重

这一部分中，我最常用的函数就是 `unique()`，使用这个函数可以看到该向量中有哪些唯一值，，在 data.frame 中（后面会提到），这样能够看到某个变量的取值情况，因为 df 的一个变量其实就是一个向量。

```
x <- c(3, 1, 2, 2, 5, 1)

sort(x)
```

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

```r
order(x)      # 返回排序后的索引
```

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

```r
x[order(x)]
```

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

```r
unique(x) # 看 x 中有哪些唯一值
```

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

当然，还可以用 summary() 和 table() 函数来看向量的分布情况，以及向量中某些元素出现的次数。如果在一个 df 中，这样就能最简单地来看某个变量的取值情况。比如，我想看一个数据库中人群年龄的分布情况：

```r
## 生成 100 个从 40 岁到 80 岁的年龄
age <- runif(n = 100, min = 40, max = 80) |> round(0)

summary(age)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   40.00   48.00   58.50   59.28   69.00   79.00
```

```r
table(age)
```

```
## age
## 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65
##  1  1  4  3  2  4  3  4  4  2  1  2  2  2  2  4  4  2  3  1  2  2  3  3  2  4
## 66 67 68 69 70 71 72 73 74 75 76 77 78 79
##  3  2  1  4  2  3  1  1  1  4  3  2  4  2
```

8.2 匹配与定位

比如我们要定位一个基因是否在某个基因向量里：

```r
genes <- c("TP53", "EGFR", "BRCA1", "MYC")
target <- c("EGFR", "MYC")

target %in% genes
```

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

```r
which(genes %in% target)
```

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

```r
match(target, genes)
```

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

8.3 集合运算

```r
a <- c("A", "B", "C")
b <- c("B", "C", "D")

intersect(a, b)  # 交集
```

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

```r
union(a, b)      # 并集
```

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

```r
setdiff(a, b)    # a 中有但 b 中没有
```

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

9. 综合思考题

题目：基因表达数据分析

假设你从一个 RNA-seq 实验中获得了一组基因的表达量数据（TPM 值），请完成以下任务：

背景数据：

```r
set.seed(2026)

# 20 个基因的表达量数据（含缺失值）
gene_names <- paste0("Gene", sprintf("%02d", 1:20))
expression <- c(runif(15, 0, 100), NA, NA, runif(3, 0, 100))
names(expression) <- gene_names

# 感兴趣的目标基因列表
target_genes <- c("Gene05", "Gene10", "Gene15", "Gene25")

# 已知的高表达基因阈值
high_expr_threshold <- 50
```

请回答以下问题：

1. **数据概览**：这个表达向量中有多少个缺失值？非缺失值的均值和标准差分别是多少？
2. **索引操作**：

- 提取第 3 到第 8 个基因的表达量
- 提取所有表达量大于 `high_expr_threshold`（高表达阈值）的基因名称
- 提取”Gene05”、“Gene12” 和”Gene18” 这三个基因的表达量

3. **缺失值处理**：创建一个新的向量 `expression_clean`，去除所有缺失值，并将表达量从高到低排序
4. **匹配与定位**：
   - 判断 `target_genes` 中哪些基因存在于我们的数据中
   - 找出目标基因在原向量中的位置索引
5. **集合运算**：假设另一个实验的高表达基因为 `c("Gene03", "Gene07", "Gene10", "Gene15", "Gene22")`，请找出：
   - 两个实验共有的高表达基因
   - 只在本实验中高表达的基因
6. **综合应用**：将所有非缺失表达量标准化到 0-1 范围（最小值变为 0，最大值变为 1），公式为：$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}}$

参考答案：

```
set.seed(2026)

# 数据准备
gene_names <- paste0("Gene", sprintf("%02d", 1:20))
expression <- c(runif(15, 0, 100), NA, NA, runif(3, 0, 100))
names(expression) <- gene_names
target_genes <- c("Gene05", "Gene10", "Gene15", "Gene25")
high_expr_threshold <- 50

# 1. 数据概览
sum(is.na(expression))
```

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

```
mean(expression, na.rm = TRUE)
```

```
## [1] 40.31455
```

```
sd(expression, na.rm = TRUE)
```

```
## [1] 28.10601
```

```
# 2. 索引操作
expression[3:8]
```

```
##     Gene03     Gene04     Gene05     Gene06     Gene07     Gene08
```

```
## 14.013996 28.572331 55.536901  2.513115 46.623055 86.101069
```

```
names(expression)[which(expression > high_expr_threshold)]
```

```
## [1] "Gene01" "Gene02" "Gene05" "Gene08" "Gene10" "Gene12" "Gene14" "Gene19"
```

```
expression[c("Gene05", "Gene12", "Gene18")]
```

```
##   Gene05   Gene12   Gene18
## 55.53690 69.18659 35.68093
```

```
# 3. 缺失值处理与排序
expression_clean <- sort(na.omit(expression), decreasing = TRUE)
expression_clean
```

```
##     Gene08     Gene14     Gene01     Gene12     Gene10     Gene02     Gene05
## 86.1010685 84.8532462 69.8673471 69.1865940 58.0806337 55.6530506 55.5369010
##     Gene19     Gene07     Gene18     Gene04     Gene09     Gene13     Gene15
## 54.5204504 46.6230555 35.6809315 28.5723306 25.2501179 23.1125047 15.3835241
##     Gene03     Gene06     Gene11     Gene20
## 14.0139958  2.5131150  0.5933257  0.1197614
```

```
# 4. 匹配与定位
target_genes %in% names(expression)
```

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

```
match(target_genes[target_genes %in% names(expression)], names(expression))
```

```
## [1]  5 10 15
```

```
# 5. 集合运算
my_high_genes <- names(expression)[which(expression > high_expr_threshold)]
other_high_genes <- c("Gene03", "Gene07", "Gene10", "Gene15", "Gene22")

intersect(my_high_genes, other_high_genes)
```

```
## [1] "Gene10"
```

```
setdiff(my_high_genes, other_high_genes)
```

```
## [1] "Gene01" "Gene02" "Gene05" "Gene08" "Gene12" "Gene14" "Gene19"
```

```
# 6. 标准化
expr_no_na <- expression[!is.na(expression)]
expr_normalized <- (expr_no_na - min(expr_no_na)) / (max(expr_no_na) - min(expr_no_na))
round(expr_normalized, 3)
```

```
## Gene01 Gene02 Gene03 Gene04 Gene05 Gene06 Gene07 Gene08 Gene09 Gene10 Gene11
##  0.811  0.646  0.162  0.331  0.645  0.028  0.541  1.000  0.292  0.674  0.006
## Gene12 Gene13 Gene14 Gene15 Gene18 Gene19 Gene20
##  0.803  0.267  0.985  0.178  0.414  0.633  0.000
```