第 3 章

数 据 结 构

在数据分析和处理过程中，选择合适的数据结构存储和操作数据至关重要。R 语言提供了多种数据结构，可以灵活、高效地管理各种类型的数据。

本章将深入探讨 R 语言中最基本的数据结构，包括向量、列表、矩阵、数据框、数组、因子以及字符串。这些数据结构是进行数据处理、分析和建模的基础。通过了解每种数据结构的创建方法、常见操作以及如何访问和修改其中的元素，可以更好地掌握数据的管理和操作。

3.1 向量

向量(Vector)是 R 语言中最基本的数据结构之一。它是一组有序的数值、字符或逻辑值的集合，且**只能包含一种数据类型**，如所有元素都为数值或字符。向量在数据处理、统计分析中广泛使用，是构建更复杂数据结构(如矩阵和数据框)的基础。

如图 3-1 所示是一个整数类型向量，图 3-2 所示是字符类型向量。

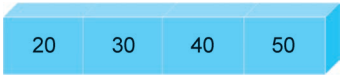

图 3-1 整数类型向量

图 3-2 字符类型向量

3.1.1 创建向量

创建向量的方法有很多，以下是一些常见的方法。

1. 使用 c()函数创建向量

c()函数是最常用的创建向量的方法，c 代表 combine(组合)，可以将多个元素组合成一个向量。示例代码如下。

```
# 1. 使用 c()函数创建向量
# 创建数值向量
numeric_vector <- c(1, 2, 3, 4, 5)
```

```
# 创建字符向量
char_vector <- c("a", "b", "c")

# 创建逻辑向量
log_vector <- c(TRUE, FALSE, TRUE, TRUE)
```

2. 使用“:”运算符创建向量

使用“:”运算符可以创建一个从某个值到另一个值的数值序列，步长为 1。示例代码如下。

```
# 2. 使用":"运算符创建向量
# 创建从 1 到 10 的数值向量
seq_vector <- 1:10
print(seq_vector)            # 输出:1 2 3 4 5 6 7 8 9 10
```

3. 使用 seq()函数创建等差数列

seq()函数用于创建具有特定步长的数值序列，可以指定起点、终点和步长。示例代码如下。

```
# 创建从 1 到 10,步长为 2 的数值向量
seq_vector <- seq(1, 10, by = 2)
print(seq_vector)            # 输出 1 3 5 7 9
```

4. 使用 rep()函数创建重复值向量

使用 rep() 函数可以将某些值重复多次，从而创建包含重复元素的向量。示例代码如下。

```
# 创建一个包含 5 个数值 3 的向量
rep_vector <- rep(3, times = 5)
print(rep_vector)            # 输出 3 3 3 3 3

# 创建一个重复 1、2、3 两次的向量
rep_vector <- rep(c(1, 2, 3), times = 2)
print(rep_vector)            # 输出 1 2 3 1 2 3
```

通过这些方法，可以轻松创建各种类型和内容的向量，满足不同的需求。

3.1.2 向量运算

向量支持常见的数学运算，如加法、减法、乘法和除法，这些运算会应用于向量的每个元素。示例代码如下。

```
# 定义向量
vector1 <- c(1, 2, 3)
vector2 <- c(4, 5, 6)

# 1. 向量相加
sum_vector <- vector1 + vector2        # 结果: c(5, 7, 9)
print(sum_vector)
```

```
# 2. 向量相减
diff_vector <- vector1 - vector2        # 结果: c(-3, -3, -3)
print(diff_vector)

# 3. 向量相乘
prod_vector <- vector1 * vector2        # 结果: c(4, 10, 18)
print(prod_vector)

# 4. 向量相除
div_vector <- vector1 / vector2         # 结果: c(0.25, 0.40, 0.50)
print(div_vector)
```

示例输出结果如下。

```
[1] 5 7 9
[1] -3 -3 -3
[1] 4 10 18
[1] 0.25 0.40 0.50
```

3.1.3　向量属性

在R语言中,向量是最基本的数据结构之一。向量的属性包括长度、类型和维度等。以下是一些关于向量属性的详细介绍和示例。

1. 长度

使用length()函数获取向量的长度(元素数量),示例代码如下。

```
# 创建一个向量
vec <- c(10, 20, 30, 40, 50)
vec_length <- length(vec)
print(vec_length)                       # 输出:5
```

2. 类型

使用typeof()或class()函数获取向量的类型,它们的区别如下。

- typeof()提供对象的基本(或存储)类型信息。
- class()提供对象的类属性信息,这对于理解对象的自定义行为和属性非常有用。

在大多数情况下,当想了解一个向量的基本类型时,会使用typeof()。而当想了解一个对象是否属于特定的类(特别是当这个对象是通过某种特定的函数或方法创建时)时,会使用class()。

示例代码如下。

```
vec_type1 <- class(vec)
print(vec_type1)                        # 输出:numeric

vec_type2 <- typeof(vec)
print(vec_type2)                        # 输出:double
```

3. 维度

虽然一维向量的维度通常不需要特别考虑,但可以使用dim()函数查看维度,示例代码如下。

```
vec_dim <- dim(vec)
print(vec_dim)                    # 输出：NULL,表示没有维度(是一维向量)
```

3.1.4 访问向量元素

要访问向量元素,可以使用向量的索引。在 R 语言中,所有向量的索引都从 1 开始,如图 3-3 所示。

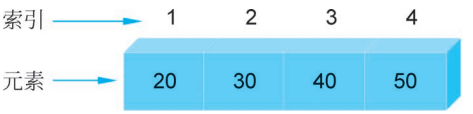

图 3-3 向量的索引

注意 在 R 语言中,向量的索引是从 1 开始的,这意味着访问向量元素时,第一个元素的索引为 1,而非 0。这种设计使得 R 语言在进行数据处理时更加直观,特别是在统计分析和数据科学领域。这种索引方式符合统计学的习惯,减少了因索引错误带来的混淆。

在 R 语言中,通过方括号[]运算符可以访问向量中的特定元素。下面是一些使用 R 语言中方括号[]运算符进行索引的示例。

1. 整数索引

可以使用正整数访问向量中的元素。正整数表示元素的位置,示例代码如下。

```
vec <- c(10, 20, 30, 40)
# 访问第二个元素
second_element <- vec[2]
print(second_element)             # 输出：20
```

2. 逻辑索引

可以使用逻辑向量(TRUE/FALSE)筛选元素。只有在逻辑向量中对应位置为 TRUE 的元素会被返回,示例代码如下。

```
vec <- c(10, 20, 30, 40)
# 选择所有大于 20 的元素
greater_than_20 <- vec[vec > 20]
print(greater_than_20)            # 输出：30 40
```

3. 字符索引

对于命名向量,可以使用字符名称访问元素。例如,如果向量的元素有名称,则可以用名称来索引,示例代码如下。

```
named_vec <- c(a = 1, b = 2, c = 3)
# 使用名称访问元素
value_b <- named_vec["b"]
print(value_b)                    # 输出：2
```

4. 负整数索引

负整数可以用来排除特定位置的元素。负值表示排除该位置的元素，示例代码如下。

```
vec <- c(10, 20, 30, 40)
# 排除第一个元素
excluding_first <- vec[ - 1]
print(excluding_first)          # 输出: 20 30 40
```

5. 范围索引

可以使用冒号:指定一个范围，返回该范围内的所有元素，示例代码如下。

```
vec <- c(10, 20, 30, 40)
# 访问前 3 个元素
first_three <- vec[1:3]
print(first_three)              # 输出: 10 20 30
```

3.2 列表

在 R 语言中，列表是一种重要的数据结构，能存储**不同类型**的数据元素。它是一种灵活且强大的容器，可以包含多种数据类型，如向量、矩阵、数据框，甚至其他列表，如图 3-4 所示是包含多种类型数据的列表。

图 3-4 列表

3.2.1 创建列表

使用 list()函数可以创建列表。list()函数的语法如下。

```
list(name1 = value1, name2 = value2, …, nameN = valueN)
```

其中，

- name1，name2，…，nameN：所有元素的名称，可选。若指定了名称，访问元素时可以用名称引用。
- value1，value2，…，valueN：列表中的各个元素，可以是数值、字符、向量、矩阵、数据框、其他列表等任何 R 语言中的对象。

下面是创建列表的示例代码。

```
# 创建一个包含不同类型元素的列表
my_list <- list(
  Name = "Jerry",                # 字符型元素
  Age = 24,                      # 数值型元素
  Scores = c(90, 85, 88),        # 向量
  Passed = TRUE,                 # 逻辑值
  Info = data.frame(             # 数据框
    Subject = c("Math", "Science"),
    Score = c(90, 88)
  )
)
```

```
# 查看列表内容
print(my_list)
```

示例输出结果如下。

```
$ Name
[1] "Jerry"

$ Age
[1] 24

$ Scores
[1] 90 85 88

$ Passed
[1] TRUE

$ Info
  Subject Score
1    Math    90
2 Science    88
```

3.2.2 访问列表中的元素

在R语言中可以使用$符号或[[ ]]符号访问列表元素，这与访问向量元素的方式非常相似。这两种方法都提供了方便的方式来获取列表中的特定元素，具体取决于元素是否具有名称。

1. 使用$运算符访问元素

如果列表的元素有名称，就可以使用$符号直接访问该元素，示例代码如下。

```
# # 创建一个包含不同类型元素的列表
# my_list <- list(
#   Name = "Jerry",                  # 字符型元素
#   Age = 24,                        # 数值型元素
#   Scores = c(90, 85, 88),          # 向量
#   Passed = TRUE,                   # 逻辑值
#   Info = data.frame(               # 数据框
#     Subject = c("Math", "Science"),
#     Score = c(90, 88)
#   )


source("./code/chapter3/3.2.1 创建列表.R")①

# 1. 使用 $ 运算符访问元素
print(my_list$Name)                  # 输出:"Jerry"
print(my_list$Age)                   # 输出:24
print(my_list$Scores)                # 输出:90 85 88
print(my_list$Passed)                # 输出:TRUE
```

代码解释：

上述代码中的 my_list 列表对象是在“3.2.1 创建列表.R”文件中定义的，因此若要在当前文件中使用 my_list 列表，则需要通过代码第①行的 source 语句将其加载到当前的 R 语言环境中。

2. 使用[[]]索引访问元素

[[]]可用来通过索引或名称访问列表中的单个元素，且适用于没有名称的列表元素，示例代码如下。

```
# 通过索引访问
my_list[[1]]                    # 输出 "Jerry"
# 通过名称访问
my_list[["age"]]                # 输出 24
```

3.3　矩阵

在 R 语言中，矩阵是一种重要的数据结构，用于存储二维数据。矩阵的每个元素**必须是相同的数据类型**，如图 3-5 所示是 3 行 3 列的数值矩阵。

3.3.1　创建矩阵

创建矩阵，使用 matrix( ) 函数，该函数的语法如下。

```
matrix(data, nrow, ncol, byrow = FALSE)
```

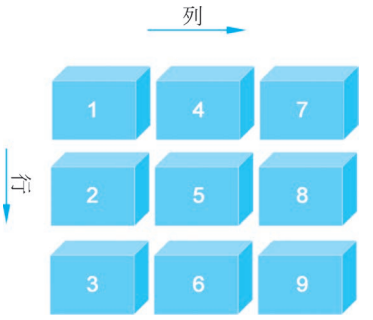

图 3-5　3 行 3 列的数值矩阵

参数说明如下。

- data：具有相同数据类型的数据项。
- nrow：行数。
- ncol：列数。
- byrow(可选)：逻辑值，指定是否按行填充(默认是按列填充)，用于控制在创建矩阵时，数据如何按行或列的顺序填充矩阵。这个参数的取值可以是 TRUE 或 FALSE，它影响矩阵的填充方式。

(1) 当 byrow=TRUE 时，数据将按行的顺序填充矩阵。这意味着从左到右填充一行，然后移到下一行，以此类推。这种方式在某些情况下更符合直觉，特别是当有一串数据，希望将其按行分组成矩阵时。

(2) 当 byrow=FALSE 时，数据将按列的顺序填充矩阵。这意味着从上到下填充一列，然后移到下一列，以此类推。这是默认的填充方式。

下面通过几个示例介绍一下如何创建矩阵。

1. 按列填充矩阵

创建一个按列填充的 3 行 3 列的矩阵，示例代码如下。

```
mat1 <- matrix(1:9, nrow = 3, ncol = 3)
print(mat1)
```

示例输出结果如下：

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

2. 按行填充矩阵

创建一个按行填充的 3 行 3 列的矩阵，示例代码如下。

```
# 2. 按行填充矩阵
mat2 <- matrix(1:9, nrow = 3, ncol = 3, byrow = TRUE)
print(mat2)
```

示例输出结果如下。

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

3. 使用向量创建矩阵

使用向量创建一个 2 行 3 列的矩阵，示例代码如下。

```
vec <- c(10, 20, 30, 40, 50, 60)
mat3 <- matrix(vec, nrow = 2)
print(mat3)
```

上述代码中使用 matrix()函数将向量 vec 转换成矩阵。参数 nrow=2 表示希望矩阵有 2 行。由于 vec 有 6 个元素，矩阵将自动分配列数。这里的矩阵将会是 2 行 3 列，示例输出结果如下。

```
     [,1]   [,2]  [,3]
[1,]   10    30    50
[2,]   20    40    60
```

3.3.2 矩阵属性

在 R 语言中，矩阵有一些属性，这些属性可用于描述和操作矩阵的特征和结构。以下是一些常见的矩阵属性。

(1) 维度：矩阵的维度是其最基本的属性，确定了矩阵的行数和列数。可使用 dim()函数获取矩阵的维度。

(2) 行数：使用 nrow()函数获取矩阵的行数。该函数返回矩阵中的行数作为结果。

(3) 列数：使用 ncol()函数获取矩阵的列数。该函数返回矩阵中的列数作为结果。

(4) 类型：矩阵类型就是矩阵中元素的数据类型，通常是 integer、numeric、character 等。

示例代码如下。

```
# 创建一个 3 行 4 列的矩阵
mat <- matrix(1:12, nrow = 3, ncol = 4)


# 获取矩阵的维度
dims <- dim(mat)
cat("矩阵的维度:", dims, "\n")
# 矩阵的维度: 3 4

# 获取矩阵的行数
rows <- nrow(mat)
cat("矩阵的行数:", rows, "\n")
# 矩阵的行数: 3

# 获取矩阵的列数
cols <- ncol(mat)
cat("矩阵的列数:", cols, "\n")
# 矩阵的列数: 4

# 获取矩阵中元素的数据类型
data_type <- typeof(mat)
cat("矩阵元素的数据类型:", data_type, "\n")
# 矩阵元素的数据类型: integer

# 获取矩阵中元素的总数量
total_elements <- length(mat)
cat("矩阵中元素的总数量:", total_elements, "\n")
# 矩阵中元素的总数量: 12
```

上述代码中使用 cat()函数将文本或变量的值输出到控制台,示例输出结果如下。

```
矩阵的维度: 3 4
矩阵的行数: 3
矩阵的列数: 4
矩阵元素的数据类型: integer
矩阵中元素的总数量: 12
```

提示 R 语言中,paste()和 cat()函数都用于将信息输出到控制台,但它们有一些区别。

(1) paste()函数用于将多个文本或对象连接成一个字符串,并返回一个新的字符串,而不是直接输出到控制台。可以通过 sep 参数指定连接文本之间的分隔符。

(2) cat()函数用于将文本输出到控制台,而不是返回一个新的字符串。它常用于在 R 语言中打印信息、结果或变量的值,并可使用 sep 参数指定输出文本之间的分隔符。其主要目的是输出文本,而非创建新的字符串。

3.3.3 设置行名和列名

在R语言中，为矩阵设置行名和列名是一个重要的步骤，它可以使矩阵的内容更易于理解和访问。

1. 设置行名

使用rownames()函数设置矩阵的行名。可以将一个字符向量赋值给矩阵的行名，示例代码如下。

```
rownames(mat) <- c("Row1", "Row2", "Row3")
```

2. 设置列名

使用colnames()函数设置矩阵的列名。同样，可以将一个字符向量赋值给矩阵的列名，示例代码如下。

```
colnames(mat) <- c("Col1", "Col2", "Col3", "Col4")
```

完整的示例代码如下。

```
# 创建一个3行4列的矩阵
mat <- matrix(1:12, nrow = 3, ncol = 4)
# 设置行名
rownames(mat) <- c("Row1", "Row2", "Row3")
# 设置列名
colnames(mat) <- c("Col1", "Col2", "Col3", "Col4")
# 查看矩阵
print(mat)
```

示例输出结果如下。

```
     Col1  Col2 Col3   Col4
Row1    1     4    7     10
Row2    2     5    8     11
Row3    3     6    9     12
```

3.3.4 访问矩阵中的元素

访问矩阵中的元素是在R语言中进行矩阵操作的基本操作之一，以下是一些示例，展示如何在R语言中访问矩阵中的元素。

1. 访问特定元素

使用行索引和列索引可访问矩阵中特定位置的元素，例如mat[1, 2]表示“第1行，第2列”的元素，示例代码如下。

```
# 创建一个3行4列的矩阵
mat <- matrix(1:12, nrow = 3, ncol = 4)
# 访问第一行第二列的元素
element <- mat[1, 2]                    # 返回4
```

2. 提取整行

通过指定行索引并留空列索引(用逗号隔开),可以提取矩阵中某一行的向量,例如mat[1,]表示提取矩阵的第1行,示例代码如下。

```
first_row <- mat[1, ]                    # 返回 c(1, 4, 7, 10)
```

3. 提取整列

类似于提取整行,开发者可以通过指定列索引并留空行索引,可以提取某一列,例如mat[,2]表示提取矩阵的第2列,示例代码如下。

```
second_col <- mat[, 2]                   # 返回值为 c(4, 5, 6)
```

4. 访问多个元素

开发者可以使用c()函数在行或列位置上选择多个元素。通过指定多个行或列索引,可以提取多行或多列。例如mat[c(1,2),]表示提取矩阵的第1行和第2行,示例代码如下。

```
rows <- mat[c(1, 2), ]                   # 返回第 1 行和第 2 行的数据
cols <- mat[, c(1, 3)]                   # 返回第 1 列和第 3 列的数据
```

5. 访问特定范围的元素

冒号":"操作符用于选择一系列连续的行或列。通过mat[1,1:2]可以提取第1行中第1到第2列的元素,示例代码如下。

```
first_row_part <- mat[1, 1:2]            # 返回值为 c(1, 4)
```

6. 使用名称访问元素

为矩阵的行和列命名后,可以直接使用行名和列名访问元素。例如,通过mat["Row1", "Col2"]可以访问第1行和第2列交叉的元素,示例代码如下。

```
# 设置行名和列名
rownames(mat) <- c("Row1", "Row2", "Row3")
colnames(mat) <- c("Col1", "Col2", "Col3", "Col4")
named_element <- mat["Row1", "Col2"]     # 返回值为 4
print(element)
```

示例输出结果如下。

```
     Col1  Col2  Col3  Col4
Row1    1     4     7    10
Row2    2     5     8    11
Row3    3     6     9    12
[1] 4
```

3.4 数据框

在R语言中,数据框(Data Frame)是一种用于存储表格数据的主要数据结构。数据框

可以看作一个**二维表格**，其中每一列是**同一种数据类型**，而每一行表示一个观测值或记录。数据框适用于统计分析和数据处理。图 3-6 所示为学生信息数据框。

Name	Age	Score
Jerry	24	90
Tom	22	85
Spike	23	88

图 3-6　学生信息数据框

数据框的关键特点如下。

(1) 唯一的列名：数据框中的每一列(如图 3-6 所示的 Name、Age、Score)都具有唯一的列名，用于标识列数据。

(2) 行数相同：每列的行数相同，如图 3-6 所示每列都有 3 行数据，对应 Jerry、Tom 和 Spike 3 位个体，确保了数据的完整性。

(3) 列内数据类型一致：同一列内的数据类型一致，如图 3-6 所示 Age 列的数据都是数值类型，Name 列的数据都是字符类型。

(4) 列间数据类型不同：不同列可以包含不同的数据类型，如图 3-6 所示 Name 列为字符类型，而 Age 和 Score 列为数值类型。

3.4.1　创建数据框

要创建数据框，可以使用 data.frame()函数。以下是创建数据框的基本语法。

```
df <- data.frame(
  列名 1 = 向量 1,
  列名 2 = 向量 2,
  ...
)
```

说明如下。

- df：这是为数据框指定的名称，读者可以根据自己的需要进行命名。
- 列名 1,列名 2,列名 3,…：这些是要为数据框的各列指定的列名。列名应为有效的标识符(不能以数字开头，且不能包含空格等)，并且每个列名在数据框中必须是唯一的。
- 向量 1,向量 2,向量 3,…：这些是包含数据的向量，它们将成为数据框的各列。向量中的元素数量必须相同，以确保数据框的结构完整且每一列都能对齐。

以下是一个示例，展示如何创建一个包含学生信息的数据框。

```
#将数据组织成向量
names <- c("Jerry", "Tom", "Spike")
ages <- c(24, 22, 23)
scores <- c(90, 85, 88)
```

```
# 使用 data.frame()函数创建数据框
df <- data.frame(Name = names, Age = ages, Score = scores)

# 查看数据框的内容
print(df)
```

上述代码中使用 data.frame()函数将这 3 个向量组合成一个数据框 df。每个向量成为数据框的一列，并且使用参数 Name、Age、Score 分别为这些列指定了列名。

示例输出结果如下。

```
   Name  Age Score
1 Jerry  24   90
2 Tom    22   85
3 Spike  23   88
```

3.4.2　数据框的基本属性

以下是数据框的一些基本属性及其详细说明。

(1) 列：数据框的每一列数据类型(数值、字符、因子等)相同。列名用于标识每列的名称，通常反映该列所代表的属性，例如 Age、Score 等。可以使用 names()函数返回数据框的所有列名，而 ncol()函数可以返回数据框的列数。

(2) 行：数据框的每一行表示一个观测值(或记录)，即一个实例的集合。而 nrow()函数返回数据框的行数。

(3) 维度：数据框具有行数和列数，可以使用 dim()函数查看。

下面是一个示例代码，展示了如何创建数据框并使用相关函数查看其基本属性，包括列、行和维度。

```
# 创建数据框
names <- c("Jerry", "Tom", "Spike")          # 姓名
ages <- c(24, 22, 23)                        # 年龄
scores <- c(90, 85, 88)                      # 成绩

df <- data.frame(Name = names, Age = ages, Score = scores)      # 创建数据框

# 查看数据框的内容
print(df)

# 1. 查看列名
column_names <- names(df)
print("列名:")
print(column_names)

# 2. 查看行数和列数
num_rows <- nrow(df)                         # 行数
num_cols <- ncol(df)                         # 列数
print(paste("行数:", num_rows))
print(paste("列数:", num_cols))
```

```
# 3. 查看维度
dimensions <- dim(df)
print("维度:")
print(dimensions)
```

示例输出结果如下。

```
   Name  Age Score
1  Jerry  24  90
2  Tom    22  85
3  Spike  23  88
[1] "列名: "
[1] "Name"  "Age"  "Score"
[1] "行数: 3"
[1] "列数: 3"
[1] "维度: "
[1] 3 3
```

3.4.3 访问数据框

在R语言中,可以通过多种方式访问数据框中的元素,包括按列、按行,或按行列组合访问数据框的内容。以下是常见的几种方法。

1. 通过列名称访问数据框列

可以通过$或[[ ]]访问数据框的特定列。

```
# 创建一个示例数据框
df <- data.frame(Name = c("Alice", "Bob", "Carol"),
                 Age = c(25, 30, 35),
                 Score = c(88, 92, 95))

# 使用 $ 符号访问列
print(df$Name)              # 输出 Name 列的所有值

# 使用 [[ ]] 访问列
print(df[["Age"]])          # 输出 Age 列的所有值
```

2. 通过行和列索引访问元素

可以使用[行,列]的形式通过索引访问数据框中的特定元素、行或列。

```
# 访问第 1 行第 2 列的元素
print(df[1, 2])             # 输出 25
# 访问第 2 行的所有元素
print(df[2, ])              # 输出整行的内容
# 访问第 3 列的所有元素
print(df[, 3])              # 输出整列的内容
```

3. 使用列名和行索引的组合访问

可以通过列名和行索引的组合访问数据框的特定元素或子集。

```
# 访问 Name 列的第 1 行
print(df[1, "Name"])          # 输出 "Alice"
# 访问 Age 列的前两行
print(df[1:2, "Age"])         # 输出 25 30
```

4. 使用 subset()函数筛选数据

subset()函数可用于根据条件筛选数据框中的行。

```
# 筛选 Age 大于 28 的行
result <- subset(df, Age > 28)
print(result)
```

3.5　数组

在 R 语言中，数组(Array)是一种可以存储多维数据的对象。数组的维度可以是任意数量，并且每一维的数据类型必须相同(如数值、字符、逻辑值等)。与向量和矩阵不同，数组能存储更高维度的数据。如图 3-7 所示是一个三维数组，该组可以视为包含 3 个层(Layer)的结构，每个层包含 2 行(Row)3 列(Column)。

3.5.1　创建数组

创建数组使用 array()函数，该函数的基本语法如下。

```
array(data, dim = c(dim1, dim2, …, dimN))
```

参数说明如下。

- data：填充数组的数据，可以是一个向量。
- dim：一个指定数组各个维度大小的向量，用于定义数组的形状(维度)。

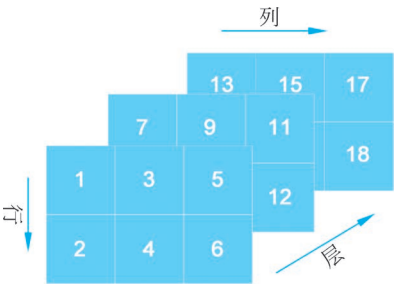

图 3-7　三维数组

创建数据的示例如下。

1. 创建一个二维数组

创建一个包含 6 个元素的二维数组，维度为 2×3，代码如下。

```
my_2d_array <- array(1:6, dim = c(2, 3))

# 输出数组内容
print(my_2d_array)
```

示例输出结果如下。

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

2. 创建一个三维数组

创建一个包含 18 个元素的三维数组，维度为 3×3×2，代码如下。

```
my_3d_array <- array(1:18, dim = c(3, 3, 2))

# 输出数组内容
print(my_3d_array)
```

示例输出结果如下。

```
, , 1

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

, , 2

     [,1]   [,2]  [,3]
[1,]   10     13    16
[2,]   11     14    17
[3,]   12     15    18
```

3. 创建一个四维数组

创建一个四维数组，维度为 2×3×2×2，代码如下。

```
my_4d_array <- array(1:24, dim = c(2, 3, 2, 2))

# 输出四维数组的内容
print(my_4d_array)
```

示例输出结果如下。

```
, , 1, 1

     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

, , 2, 1

     [,1]   [,2]  [,3]
[1,]    7      9    11
[2,]    8     10    12

, , 1, 2

     [,1]   [,2]  [,3]
[1,]   13     15    17
[2,]   14     16    18

, , 2, 2

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

```
[1,]    19    21    23
[2,]    20    22    24
```

3.5.2　访问数组中的元素

在R语言中，可以通过多种方式访问数组中的元素。以下是访问三维数组中元素的示例，涵盖不同的访问方法。

1. 通过索引访问单个元素

访问三维数组中的特定元素，示例代码如下。

```
# 创建一个三维数组，维度为 3×4×2
my_array <- array(1:24, dim = c(3, 4, 2))

# 查看数组内容
print(my_array)

# 1. 通过索引访问单个元素
# 访问第一层的第二行第三列的元素
element1 <- my_array[2, 3, 1]
print(element1)              # 输出:8

# 访问第二层的第一行第四列的元素
element2 <- my_array[1, 4, 2]
print(element2)              # 输出:22
```

2. 通过索引访问特定层

获取整个层的数据，示例代码如下。

```
# 获取第一层的所有数据
layer1 <- my_array[, , 1]
print(layer1)
```

示例输出结果如下。

```
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

3. 访问整行或整列

选择特定行或列的所有元素，示例代码如下。

```
# 获取第一层的第二行的所有元素
row_elements <- my_array[2, , 1]
print(row_elements)          # 输出:2 5 8 11

# 获取第二层的第三列的所有元素
column_elements <- my_array[, 3, 2]
print(column_elements)       # 输出:19 20 21
```

3.6 因子

在R语言中，因子(Factor)是一种数据类型，用于存储**分类数据**。因子在数据分析中特别有用，因为它们能帮助用户处理和分析分类变量，例如性别(男、女)、颜色(红、绿、蓝)、学历(高中、本科、硕士、博士)等。因子将这些类别数据存储为整数，并将每个整数映射到一个标签或水平。

3.6.1 创建因子

创建因子：使用factor()函数，其基本语法格式如下。

```
factor(x, levels, labels, ordered = FALSE)
```

参数说明如下。

- x：要转换为因子的向量。
- levels：因子的水平(即分类)，可以指定各类别的顺序。
- labels：因子各水平的标签，若不指定，则直接使用x中的值作为标签。
- ordered：逻辑值，是否创建有序因子，默认为FALSE。

使用factor()函数创建因子的示例代码如下。

```
# 创建一个字符向量
data <- c("apple", "banana", "apple", "orange", "banana")  ①
# 将字符向量转换为因子
factor_data <- factor(data)                                ②
# 查看因子
print(factor_data)
```

代码解释如下。

- 代码第①行使用c()函数创建了一个字符向量data。这个向量包含了5个元素："apple"、"banana"、"apple"、"orange"和"banana"，这里data是一个普通的字符向量，可以包含重复的值。
- 代码第②行使用factor()函数将字符向量data转换为因子factor_data，在转换过程中，R语言会自动识别并提取data中的唯一值(即水平)，并为这些唯一值赋予一个整数编码。这意味着："apple"和"banana"被识别为两个不同的水平。

示例输出结果如下。

```
[1] apple banana apple orange banana
Levels: apple banana orange
```

3.6.2 创建有序因子

有时，因子的水平具有特定的顺序。可以在创建因子时使用levels参数设置水平的顺序，同时将ordered参数设置为TRUE，以明确水平的排序关系，代码如下。

```
# 创建一个字符向量
data <- c("apple", "banana", "apple", "orange", "banana")

# 创建一个有序因子
ordered_factor <- factor(
  data,
  levels = c("apple", "orange","banana"),
  ordered = TRUE
)

# 查看有序因子
print(ordered_factor)
```

上述代码中使用 factor()函数将字符向量 data 转换为有序因子 ordered_factor，其中 levels=c("apple","orange","banana")指定了因子的水平，并且定义了它们的顺序；ordered=TRUE 表示创建一个有序因子，这样 R 语言会将水平视为有序关系，"apple"<"orange"<"banana"。

示例输出结果如下。

```
[1] apple banana apple orange banana
Levels: apple < orange < banana
```

3.6.3　自定义因子标签

当创建因子时，可以使用 labels 参数为每个水平指定标签，示例代码如下。

```
factor_data <- factor(data,
                      levels = c("apple", "banana", "orange"),
                      labels = c("苹果", "香蕉", "橘子"))

# 查看因子
print(factor_data)
```

在上述代码中，使用 factor()函数创建因子时，通过 labels=c("苹果","香蕉","橘子")参数为每个因子水平设置了标签。这样，因子水平"apple"被表示为"苹果"，"banana"被表示为"香蕉"，"orange"被表示为"橘子"。标签通常是更具描述性的名称，可以使用中文、英文或其他语言，以使数据更加直观、易懂。

示例输出结果如下。

```
[1] apple banana apple orange banana
Levels: apple banana orange

[1] 苹果 香蕉 苹果 橘子 香蕉
Levels: 苹果 香蕉 橘子
```

3.6.4　使用 table()函数进行数据汇总

R 语言中的 table()函数是用于计算类别数据的频数分布的非常有用的工具。table()

函数的基本用法如下。

```
table(x)
```

其中，参数 x 可以是向量、因子、数据框或矩阵。

table()函数返回的对象是一个频率表，它的每一列代表一个类别，数字表示该类别出现的次数。

示例代码如下。

```
# 示例:使用 table() 计算因子列的频率
data <- data.frame(
  gender = factor(c("Male", "Female", "Male", "Female", "Male", "Male"))
)

# 计算 gender 列的频率分布
table(data$gender)
```

示例输出结果如下。

```
Female   Male
     2      4
```

代码解释如下。

table(data$gender)会对 gender 列进行频率统计，并返回每个类别的计数。在这个例子中，gender 列包含两个不同的值：Male 和 Female。

- Female：该类别在数据中出现了 2 次。
- Male：该类别在数据中出现了 4 次。

这表示在这个数据集中，Female 出现了 2 次，Male 出现了 4 次。

3.7 字符串

在 R 语言中，字符串被视为字符向量，是存储文本信息的常见数据结构。每个字符串都是一个字符向量，开发者可以像操作向量一样对字符串进行操作，例如通过索引访问单个字符或子字符串，使用循环遍历字符，以及利用向量化操作处理整个字符串向量。

3.7.1 创建字符串

在 R 语言中，字符串可以用单引号(')或双引号(")表示，所以字符串与字符表示方式是一样的。

示例代码如下。

```
# 使用单引号表示字符串
string1 <- '这是一个字符串'

# 使用双引号表示字符串
string2 <- "这是另一个字符串"
```

```
# 包含单引号的字符串
string3 <- "他是一个很棒的朋友,叫作 'Alice'."    # 定义字符串变量 string3,其中包含单引号

# 包含双引号的字符串
string4 <- '她说:"你好,世界!"'                    # 定义字符串变量 string4,其中包含双引号
```

3.7.2　字符串操作

在 R 语言中,字符串操作非常常见且灵活,提供了多种函数来处理和修改字符串。以下是一些基本的字符串操作,包括拼接、切割、查找、替换和转换等。

下面介绍这些字符串操作。

1. 字符串拼接

paste()函数用于连接多个字符串,默认使用一个空格作为分隔符。示例代码如下。

```
string1 <- "Hello"
string2 <- "World"
combined_string <- paste(string1, string2)    # 输出: "Hello World"
```

2. 字符串切割

strsplit()函数用于将字符串根据指定的分隔符拆分成多个子字符串,并返回一个列表。strsplit()函数的语法如下。

```
strsplit(x, split)
```

其中,

- x:要拆分的字符串;
- splits:用于拆分字符串的分隔符。

示例代码如下。

```
text <- "苹果,香蕉,橙子"
fruits <- strsplit(text, ",")                # 输出: 列表 [['1']] "苹果" "香蕉" "橙子"
```

3. 字符串长度

使用 nchar()函数计算字符串的长度(字符数),示例代码如下。

```
length1 <- nchar("Hello")        # 输出: 5
length2 <- nchar("你好!")        # 输出: 3
```

4. 查找字符串位置

查找字符串位置可以使用 regexpr()函数,它用于查找模式在字符串中第一次出现的位置,返回的值包括匹配的起始位置,如果未找到,则返回 -1。regexpr()函数的语法如下。

```
regexpr(pattern, text)
```

其中,

- pattern:要查找的模式。
- text:要搜索的字符串。

示例代码如下。

```
# 示例字符串
text <- "Hello, world! Welcome to R programming."
pattern <- "R"

# 查找第一个匹配位置
first_position <- regexpr(pattern, text)
# 打印匹配位置的整数值
print(first_position[1])          # 输出: [1] 26
```

上述代码使用 regexpr()函数查找第一个匹配子字符串的位置时,返回值是一个带有附加属性的对象。通过 first_position[1]提取出匹配的位置(整数值),使得输出更简洁且易于理解。

5. 替换

字符串替换可以使用 gsub()函数,在字符串中替换所有符合条件的内容。gsub()函数的语法如下。

```
gsub(pattern, replacement, x)
```

其中,

- pattern:需要查找的字符串或正则表达式;
- replacement:替换为的字符串;
- x:要处理的字符串。

示例代码如下。

```
text <- "Hello, World!"
new_text <- gsub("World", "R", text)        # 输出: "Hello, R!"
```

6. 转换大小写

将字符串转换为大写的函数是 toupper(),将字符串转换为小写的函数是 tolower(),示例代码如下。

```
uppercase <- toupper("Hello")               # 输出: "HELLO"
lowercase <- tolower("Hello")               # 输出: "hello"
```

7. 提取子字符串

substr()函数用于提取字符串中的子字符串,其语法如下。

```
substr(x, start, stop)
```

其中,

- x:要提取的字符串。
- start:子字符串的起始位置(从 1 开始)。
- stop:子字符串的结束位置。

示例代码如下:

```
text <- "Hello, World!"
substring <- substr(text, start = 1, stop = 5)        # 输出: "Hello"
```

8. 字符串格式化

sprintf()函数可用于格式化字符串，可以将变量值嵌入字符串中，其语法如下。

```
sprintf(format, …)
```

其中，

- format：格式字符串，包含格式占位符。
- …：要插入的值。

示例代码如下。

```
name <- "Alice"
age <- 30
formatted_string <- sprintf("我的名字是 %s，我 %d 岁。", name, age)
# 输出："我的名字是 Alice，我 30 岁。"
```

3.8 本章练习

1. 问答题

（1）请简要描述 R 语言中的向量，并举例说明如何创建一个向量。向量和其他数据结构（如列表、矩阵）有何不同？

（2）请简述列表和数据框的定义和应用场景，指出它们之间的主要区别，并举例说明如何创建和访问这两种数据结构。

（3）请解释因子的定义及其作用，并说明如何创建一个因子。为什么在处理分类数据时，因子比字符向量更有优势？

（4）请描述如何在 R 语言中创建一个矩阵，并举例说明如何设置矩阵的行名和列名。可以执行哪些常见操作，如访问矩阵元素、修改矩阵的值等？

2. 选择题

（1）在 R 语言中，以下哪个数据结构能存储不同类型的数据元素？（　　）

A. 向量　　B. 列表　　C. 矩阵　　D. 数组

（2）在 R 语言中，创建因子的函数是（　　）。

A. factor()　　B. array()　　C. list()　　D. data.frame()

（3）在 R 中，以下哪个数据结构是二维的，并且允许元素具有相同的数据类型？（　　）

A. 向量　　B. 列表　　C. 矩阵　　D. 因子

（4）以下哪个函数可用于将一个数据框的列转换为因子？（　　）

A. as.factor()　　B. as.character()　　C. as.list()　　D. as.matrix()

3. 编程题

编写一个 R 语言脚本，创建一个包含 5 个数字的向量，并执行以下操作：

- 计算向量的总和。
- 计算向量的均值。
- 计算向量的标准差。