1.3 数据结构 ii：列表、数据框、因子

1.3.1 列表（list）

列表，可以包含不同类型的对象，甚至可以包括其他列表。列表的灵活性使得它非常有用。

例如，用 R 拟合一个线性回归模型，其返回结果就是一个列表，其中包含了线性回归的详细结果，如线性回归系数（数值向量）、残差（数值向量）、QR 分解（包含一个矩阵和其他对象的列表）等。因为这些结果全都被打包到一个列表中，就可以很方便地提取所需信息，而无需每次调用不同的函数。

列表最大的好处就是，它能够将多个不同类型的对象打包到一起，使得可以根据位置和名字访问它们。

创建列表

可以用函数 `list()` 创建列表。不同类型的对象可以放入同一个列表中。

例如，创建了一个列表，包含 3 个成分：一个单元素的数值向量、一个两元素的逻辑向量和一个长度为 3 的字符向量：

```
l0 = list(1, c(TRUE, FALSE), c("a", "b", "c"))
l0
```

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

可以在创建列表时，为列表的每个成分指定名字：

```
l1 = list(A = 1, B = c(TRUE, FALSE), C = c("a", "b", "c"))
l1
```

```
## $A
## [1] 1
##
## $B
## [1]  TRUE FALSE
```

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

也可以创建列表后再对列表成分命名或修改名字：

```
names(l1) = NULL       # 移除列表成分的名字
names(l1) = c("x","y","z")
```

从列表中提取成分的内容

提取列表中成分下的内容，最常用的方法是用 `$`，通过成分名字来提取该成分下的内容：

```
l1$y
l1$m                   # 访问不存在的成分 m，将会返回 NULL
```

也可以用 `[[n]]` 来提取列表第 `n` 个成分的内容，`n` 也可以换成成分的名字：

```
l1[[2]]                # 同 l1[["y"]]
```

用 `[[]]` 提取列表中某个成分的内容更加灵活，可用在函数调用中，通过参数来传递成分名字：

```
p = "y"                # 想要提取其内容的成分名字
l1[[p]]
```

提取列表子集

经常也需要从列表中提取多个成分及其内容，由这些成分组成的列表构成了原列表的一个子集。

就像提取向量和矩阵的子集一样，提取一个列表子集是用 `[]`，可以取出列表中的一些成分，作为一个新的列表。

`[]` 中可以用字符向量表示成分名字，用数值向量表示成分位置，或用逻辑向量指定是否选择，来取出列表成分。

```
l1["x"]                    # 同 l1[1]
l1[c("x", "z")]            # 同 l1[c(1, 3)], l1[c(TRUE, FALSE, TRUE)]
```

> 用 `[]` 提取若干成分时，返回列表的子集，还是一个列表；用 `[[ ]]` 提取单个成分的元素，返回的是对应成分的元素。
> 总之，`[]` 提取对象的子集，类型仍是该对象；`[[]]` 提取对象的内容（下一级元素）。

对列表的成分赋值

即先访问（提取）到列表的成分，再赋以相应的值。注意，若给一个不存在的成分赋值，列表会自动地在对应名称或位置下增加一个新成分。

```
l1$x = 0   # 将列表的成分 x 赋值为 0
```

也可以同时给多个列表成分赋值：

```
l1[c("x", "y")] = list(x = "new value for y", y = c(3, 1))
```

若要移除列表中的某些成分，只需赋值为 NULL：

```
l1[c("z", "m")] <- NULL
```

列表函数

用函数 as.list() 可将向量转换成列表：

```
l2 = as.list(c(a = 1, b = 2))
l2
```

```
## $a
## [1] 1
##
## $b
## [1] 2
```

用去列表化函数 unlist()，可将一个列表打破成分界限，强制转换成一个向量[5]：

```
unlist(l2)
```

```
## a b
## 1 2
```

tidyverse 系列中的 purrr 包为方便操作列表，提供了一系列列表相关的函数，建议读者查阅使用：

- pluck()：同 [[提取列表中的元素
- keep()：保留满足条件的元素
- dicard()：删除满足条件的元素
- compact()：删除列表中的空元素
- append()：在列表末尾增加元素
- flatten()：摊平列表（只摊平一层）

[5]若列表的成分具有不同类型，则自动向下兼容到统一类型。

1.3.2 数据框（数据表）

R 语言中做统计分析的样本数据，都是按数据框类型操作的。

数据框是指有若干行和列的数据集，它与矩阵类似，但并不要求所有列都是相同的类型；本质上讲，数据框就是一个列表，它的每个成分都是一个向量，并且长度相同，以表格的形式展现。总之，**数据框是由列向量组成、有着矩阵形式的列表**。

数据框与最常见的数据表是一致的：每一列代表一个变量属性，每一行代表一条样本数据：

姓名	性别	年龄	专业
Ken	Male	24	Finance
Ashley	Female	25	Statistics
Jennifer	Female	23	Computer Science

图 1.7: 数据表样式

R 中自带的数据框是 `data.frame`，建议改用更现代的数据框：`tibble`[6]。

Hadley 在 `tibble` 包中引入一种 `tibble` 数据框，以代替 `data.frame`；而且 `tidyverse` 包都是基于 `tibble` 数据框。

`tibble` 对比 `data.frame` 的优势：

- `tibble()` 比 `data.frame()` 做的更少：不改变输入变量的类型（**R 4.0.0 之前默认将字符串转化为因子！**），不会改变变量名，不会创建行名；
- `tibble` 对象的列名可以是 R 中的“非法名”：非字母开头、包含空格，但定义和使用变量时都需要用倒引号 ‘ 括起来；
- `tibble` 在输出时不自动显示所有行，避免大数据框时显示很多内容；
- 用 `[]` 选取列子集时，即使只选取一列，返回结果仍是 `tibble`，而不自动简化为向量。

创建数据框

用 `tibble()` 根据若干列向量创建 `tibble`：

```
library(tibble)  # 或 tidyverse

persons <- tibble(
  Name = c("Ken", "Ashley", "Jennifer"),
  Gender = c("Male", "Female", "Female"),
  Age = c(24, 25, 23),
```

[6]读者若习惯用 R 自带的 `data.frame`, 只需要换个名字，将 `tibble` 改为 `data.frame` 即可。

```
  Major = c("Finance", "Statistics", "Computer Science")
)
persons
```

```
## # A tibble: 3 x 4
##   Name     Gender   Age Major
##   <chr>    <chr>  <dbl> <chr>
## 1 Ken      Male      24 Finance
## 2 Ashley   Female    25 Statistics
## 3 Jennifer Female    23 Computer Science
```

用 `tribble()` 按行录入数据式创建 `tibble`：

```
tribble(
  ~Name, ~Gender, ~Age, ~Major,
  "Ken", "Male", 24, "Finance",
  "Ashley", "Female", 25, "Statistics",
  "Jennifer", "Female", 23, "Computer Science"
)
```

用 `as_tibble()` 将 `data.frame`, `matrix`, 各成分等长度的 `list`, 转换为 `tibble`。

数据框既是列表的特例，也是矩阵的推广，因此访问这两类对象的方式都适用于数据框。例如与矩阵类似，对数据框的行列重新命名：

```
df = tibble(id = 1:4,
            level = c(0, 2, 1, -1),
            score = c(0.5, 0.2, 0.1, 0.5))
names(df) = c("id", "x", "y")
df
```

```
## # A tibble: 4 x 3
##      id     x     y
##   <int> <dbl> <dbl>
## 1     1     0   0.5
## 2     2     2   0.2
## 3     3     1   0.1
## 4     4    -1   0.5
```

提取数据框的元素、子集

数据框是由列向量组成、有着矩阵形式的列表，所以可以用两种操作方式来访问数据框的元素和子集。

1. 以列表方式提取数据框的元素、子集

若把数据框看作是由向量组成的列表，则可以沿用列表的操作方式来提取元素或构建子集。例如，可以用 `$` 按列名来提取某一列的值，或者用 `[[]]` 按照位置或列名提取。

例如，提取列名为 `x` 列的值，得到向量：

```
df$x                  # 同 df[["x"]], df[[2]]
```

```
## [1]  0  2  1 -1
```

以列表形式构建子集完全适用于数据框，同时也会生成一个新的数据框。提取子集的操作符 `[]` 允许用数值向量表示列的位置，用字符向量表示列名，或用逻辑向量指定是否选择。

例如，提取数据框的一列或多列，得到子数据框：

```
df[1]                 # 提取第 1 列，同 df["id"]
```

```
## # A tibble: 4 x 1
##      id
##   <int>
## 1     1
## 2     2
## 3     3
## 4     4
```

```
df[1:2]               # 同 df[c("id", "x")], df[c(TRUE, TRUE, FALSE)]
```

```
## # A tibble: 4 x 2
##      id     x
##   <int> <dbl>
## 1     1     0
## 2     2     2
## 3     3     1
## 4     4    -1
```

2. 以矩阵方式提取数据框的元素、子集

以列表形式操作并不支持行选择。以矩阵形式操作更加灵活，若将数据框看作矩阵，其二维形式的存取器可以很容易地获取一个子集的元素，同时支持列选择和行选择。

换句话说，可以使用 `[i, j]` 指定行或列来提取数据框子集，`[ , ]` 内可以是数值向量、字符向量或者逻辑向量。

若行选择器为空，则只选择列（所有行）：

```
df[, "x"]
```

```
## # A tibble: 4 x 1
##       x
##   <dbl>
## 1     0
## 2     2
## 3     1
## 4    -1
```

```
df[, c("x", "y")]   # 同 df[, 2:3]
```

```
## # A tibble: 4 x 2
##       x     y
##   <dbl> <dbl>
## 1     0   0.5
## 2     2   0.2
## 3     1   0.1
## 4    -1   0.5
```

若列选择器为空，则只选择行（所有列）：

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

```
## # A tibble: 2 x 3
##      id     x     y
##   <int> <dbl> <dbl>
## 1     1     0   0.5
## 2     3     1   0.1
```

同时选择行和列：

```
df[1:3, c("id", "y")]
```

```
## # A tibble: 3 x 2
##      id     y
##   <int> <dbl>
## 1     1   0.5
## 2     2   0.2
## 3     3   0.1
```

根据条件筛选数据。例如用 `y >= 0.5` 筛选 `df` 的行，并选择 `id` 和 `y` 两列：

```
df[df$y >= 0.5, c("id","y")]
```

```
## # A tibble: 2 x 2
##      id     y
##   <int> <dbl>
## 1     1   0.5
## 2     4   0.5
```

按行名属于集合 {x, y, w} 来筛选 df 的行，并选择 x 和 y 两列：

```
ind = names(df) %in% c("x", "y", "w")
df[1:2, ind]
```

```
## # A tibble: 2 x 2
##       x     y
##   <dbl> <dbl>
## 1     0   0.5
## 2     2   0.2
```

给数据框赋值

给数据框赋值，就是选择要赋值的位置，再准备好同样大小且格式匹配的数据，赋值给那些位置即可，所以同样有列表方式和矩阵方式。

1. 以列表方式给数据框赋值

用 $ 或 [[ ]] 对数据框的 1 列赋值

```
df$y = c(0.6,0.3,0.2,0.4)    # 同 d[["y"]] = c(0.4,0.5,0.2,0.8)
```

利用现有列，创建 (计算) 新列：

```
df$z = df$x + df$y
df
```

```
## # A tibble: 4 x 4
##      id     x     y     z
##   <int> <dbl> <dbl> <dbl>
## 1     1     0   0.5   0.5
## 2     2     2   0.2   2.2
## 3     3     1   0.1   1.1
## 4     4    -1   0.5  -0.5
```

```
df$z = as.character(df$z)    # 转换列的类型
df
```

```
## # A tibble: 4 x 4
##      id     x     y z
##   <int> <dbl> <dbl> <chr>
## 1     1     0   0.5 0.5
## 2     2     2   0.2 2.2
## 3     3     1   0.1 1.1
## 4     4    -1   0.5 -0.5
```

用 [] 可以对数据框的 1 列或多列进行赋值：

```
df["y"] = c(0.8,0.5,0.2,0.4)
df[c("x", "y")] = list(level = c(1,2,1,0),
                       score = c(0.1,0.2,0.3,0.4))
```

2. 以矩阵方式给数据框赋值

以列表方式对数据框进行赋值时，也是只能访问列。若需要更加灵活地进行赋值操作，可以以矩阵方式进行。

```
df[1:3,"y"] = c(-1,0,1)
df[1:2,c("x","y")] = list(level = c(0,0),
                          score = c(0.9,1.0))
```

一些有用函数

函数 str() 或 glimpse() 作用在 R 对象上，显示该对象的结构：

```
str(persons)
```

```
## tibble [3 x 4] (S3: tbl_df/tbl/data.frame)
##  $ Name  : chr [1:3] "Ken" "Ashley" "Jennifer"
##  $ Gender: chr [1:3] "Male" "Female" "Female"
##  $ Age   : num [1:3] 24 25 23
##  $ Major : chr [1:3] "Finance" "Statistics" "Computer Science"
```

summary() 作用在数据框/列表上，将生成各列/成分的汇总信息：

```
summary(persons)
```

```
##      Name              Gender               Age
##  Length:3           Length:3           Min.   :23.0
##  Class :character   Class :character   1st Qu.:23.5
##  Mode  :character   Mode  :character   Median :24.0
##                                        Mean   :24.0
##                                        3rd Qu.:24.5
```

```
##                                       Max.   :25.0
##     Major
##  Length:3
##  Class :character
##  Mode  :character
##
##
##
```

经常需要将多个数据框（或矩阵）按行或按列进行合并。用函数 rbind()，增加行（样本数据），要求宽度（列数）相同；用函数 cbind()，增加列（属性变量），要求高度（行数）相同。

例如，向数据框 persons 数据框中添加一个人的新记录：

```
rbind(persons,
      tibble(Name = "John", Gender = "Male",
             Age = 25, Major = "Statistics"))
```

```
## # A tibble: 4 x 4
##   Name     Gender   Age Major
##   <chr>    <chr>  <dbl> <chr>
## 1 Ken      Male      24 Finance
## 2 Ashley   Female    25 Statistics
## 3 Jennifer Female    23 Computer Science
## 4 John     Male      25 Statistics
```

向 persons 数据框中添加两个新列表示每个人是否已注册和其手头的项目数量：

```
cbind(persons, Registered = c(TRUE, TRUE, FALSE),
               Projects = c(3, 2, 3))
```

```
##       Name Gender Age            Major Registered Projects
## 1      Ken   Male  24          Finance       TRUE        3
## 2   Ashley Female  25       Statistics       TRUE        2
## 3 Jennifer Female  23 Computer Science      FALSE        3
```

> bind() 和 cbind() 不会修改原始数据，而是生成一个添加了行或列的新数据框。更建议用 dplyr 包中的 bind_rows() 和 bind_cols()。

函数 expand.grid() 可生成多个属性水平值所有组合（笛卡尔积）的数据框：

```
expand.grid(type=c("A", "B"), class=c("M", "L", "XL"))
```

```
##   type class
## 1    A     M
## 2    B     M
## 3    A     L
## 4    B     L
## 5    A    XL
## 6    B    XL
```

1.3.3 因子（factor）

变量分为离散/分类型、连续/数值型（通常的数值变量，可带小数位）；分类型变量又分为：名义型（无顺序好坏之分的分类变量，如性别）、有序型（有顺序好坏之分的分类变量，如疗效）。

名义型和有序型的分类变量，在 R 语言中称为因子，因子本质上是一个带有水平（level）属性的整数向量，其中“水平”是指事前确定可能取值的有限集合。

因子提供了一个简单且紧凑的形式来处理分类数据，因子用水平来表示所有可能的取值，例如，性别有两个水平：男、女。

创建因子

函数 `factor()` 用来创建因子，基本格式为：

```
factor(x, levels, labels, ordered, ...)
```

其中，

`x`：为创建因子的数据向量；

`levels`：指定因子的各水平值，默认为 x 中不重复的所有值；

`labels`：设置各水平名称（前缀），与水平一一对应；

`ordered`：设置是否对因子水平排序，默认 `FALSE` 为无序因子，`TRUE` 为有序因子；

该函数还包含参数 `exclude`：指定有哪些水平是不需要的（设为 `NA`）；`nmax` 设定水平数的上限。

```
x = c(" 男", " 女", " 男", " 男", " 女")
sex = factor(x)
sex
```

```
## [1] 男 女 男 男 女
## Levels: 男 女
```

```
levels(sex)                    # 访问因子水平
```

```
## [1] "男" "女"
```

```
levels(sex) = c("M", "F")      # 修改因子水平
sex
```

```
## [1] M F M M F
## Levels: M F
```

函数 gl() 用来生成有规律的水平值组合因子。对于多因素试验设计，用该函数可以生成多个因素完全组合，基本格式为：

```
gl(n, k, length, labels, ordered, ...)
```

其中，

n：为因子水平个数；

k：为同一因子水平连续重复次数；

length：为总的元素个数，默认为 n * k, 若不够则自动重复；

labels：设置因子水平值；

ordered：设置是否为有序，默认为 FALSE。

```
tibble(
  Sex = gl(2, 3, length=12, labels=c(" 男", " 女")),
  Class = gl(3, 2, length=12, labels=c(" 甲", " 乙", " 丙")),
  Score = gl(4, 3, length=12, labels=c(" 优", " 良", " 中", " 及格"))
)
```

```
## # A tibble: 12 x 3
##    Sex   Class Score
##    <fct> <fct> <fct>
##  1 男    甲    优
##  2 男    甲    优
##  3 男    乙    优
##  4 女    乙    良
##  5 女    丙    良
##  6 女    丙    良
##  7 男    甲    中
##  8 男    甲    中
##  9 男    乙    中
## 10 女    乙    及格
## 11 女    丙    及格
```

```
## 12 女     丙     及格
```

使用因子

R 中因子是以整数型向量存储的，每个因子水平对应一个整数型的数。对字符型向量创建的因子，可以指定因子水平顺序，否则默认会按照字母顺序，再对应到整数型向量。

> 不能直接将因子数据当字符型操作，需要用 `as.character()` 转化。

考虑用一个变量存储字符串型的月份：

```
x1 = c("Dec", "Apr", "Jan", "Mar")
```

这有两个问题：

- 实际只需要 12 个月份值，对拼写错误也无能为力；
- 不会按照想要的方式排序：

```
sort(x1)
```

```
## [1] "Apr" "Dec" "Jan" "Mar"
```

若改用因子型就能避免上述问题。创建因子型，首先要创建一个有效的“水平值”列表：

```
month_levels = c("Jan", "Feb", "Mar", "Apr", "May", "Jun",
                 "Jul", "Aug", "Sep", "Oct", "Nov", "Dec")
```

再来创建因子：

```
y1 = factor(x1, levels = month_levels)
y1
```

```
## [1] Dec Apr Jan Mar
## Levels: Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
```

```
sort(y1)
```

```
## [1] Jan Mar Apr Dec
## Levels: Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
```

并且任何不在水平集中的值都将转化为 NA，相当于“识错”：

```
x2 = c("Dec", "Apr", "Jam", "Mar")
y2 = factor(x2, levels = month_levels)
y2
```

```
# [1] Dec  Apr  <NA> Mar
# Levels: Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
```

有时候你可能更希望让水平的顺序与其在数据集中首次出现的次序相匹配，设置参数 `levels = unique(x)`：

```
f1 = factor(x1, levels = unique(x1))
f1
```

```
## [1] Dec Apr Jan Mar
## Levels: Dec Apr Jan Mar
```

有用函数

函数 `table()`，可以统计因子各水平的出现次数（频数），也可以统计向量中每个不同元素的出现次数，返回结果为命名向量。

```
table(sex)
```

```
## sex
## M F
## 3 2
```

函数 `cut()`，用来做连续变量离散化：将数值向量切分为若干区间段，返回因子。基本格式为：

```
cut(x, breaks, labels, ...)
```

其中，
`x`：为要切分的数值向量；
`breaks`：切分的界限值构成的向量，或表示切分段数的整数。
该函数还包含参数 `right` 设置区间段是否左开右闭，`include.lowest` 设置是否包含下界，`ordered_result` 设置是否对结果因子排序。

```
Age = c(23,15,36,47,65,53)
cut(Age, breaks = c(0,18,45,100),
    labels = c("Young","Middle","Old"))
```

```
## [1] Middle Young  Middle Old    Old    Old
## Levels: Young Middle Old
```

`tidyverse` 系列中的 `facats` 包是专门为处理因子型数据而设计的，提供了一系列方便的函数，建议读者查阅使用：

- `fct_count()`：计算因子各水平频数、占比，可按频数排序
- `fct_c()`: 合并多个因子的水平
- 改变因子水平的顺序：
  - `fct_relevel()`: 手动对水平值重新排序
  - `fct_infreq()`: 按高频优先排序
  - `fct_inorder()`: 按水平值出现的顺序
  - `fct_rev()`: 将顺序反转
  - `fct_reorder()`: 根据其它变量或函数结果排序（绘图时有用）
- 修改水平：
  - `fct_recode()`: 对水平值逐个重编码
  - `fct_collapse()`: 推倒手动合并部分水平
  - `fct_lump_*()`: 将多个频数小的水平合并为其它
  - `fct_other()`: 将保留之外或丢弃的水平合并为其它
- 增加或删除水平：
  - `fct_drop()`: 删除若干水平
  - `fct_expand`: 增加若干水平
  - `fct_explicit_na()`: 为 `NA` 设置水平