第3章　R语言数据处理

R语言数据处理

- 数据的输入与输出
  - 终端输出、读取数据、保存数据
- 数据选择
  - 常见数据操作函数
  - 向量、数组、矩阵、数据框、列表等取子集
  - 常见数据选择函数：
    - subset函数、which函数、with函数
- 数据转换和整理
  - 修改、增加、删除
  - 排序、合并、缺失处理

3.1 数据的输入与输出

3.1.1 终端输出

- 在R交互运行时要显示某一个对象的值只要键入其名字即可。
- 这实际上是调用了**print()函数**，即print(x)。在非交互运行（程序）中应使用print()来输出。
- print()函数是一个通用函数，即它对不同的自变量有不同的反应。对各种特殊对象如数组、模型结果等都可以规定print的输出格式。

```
num <- 12.345678
print(num, digits = 6)  # 使用digits时，会进行四舍五入
## 1] 12.3457
print(num, digits = 4)
## [1] 12.35
```

- cat()函数也用来输出，但它可以把多个参数连接起来再输出（具有paste()的功能）。例如：

  > cat("i = ", i, "\n")

- **注意使用cat()时要自己加上换行符“\n”**。它把各项转换成字符串，中间隔以空格连接起来，然后显示。
- 如果要使用自定义的分隔符，可以用sep=参数，例如：> cat(c("AB", "C"), c("E", "F"), "\n", sep="") ABCEF
- cat()还可以指定一个参数file=给一个文件名，可以把结果写到指定的文件中，如：

  cat("i = ", 1, "\n", file="c:/work/result.txt")

  非常适用于中间或最后结果的存储。

- cat()函数和print()都不具有很强的自定义格式功能，为此可以使用cat()与format() 函数配合实现。
- format()函数可以把数字和字符串转为统一格式的字符串，例如：

```
format(c(1, 100, 1000))

format(c("a", "dddd", "bba"))

## [1] "   1" " 100" "1000"

## [1] "a    " "dddd" "bba "
```

例：print()函数和format()函数配合

```
print(format(64.922, digits = 4))
print(format(123.456789, digits = 4))
## [1] "64.92"
## [1] "123.5"
```

R中目前format() 函数功能仍较弱，但R有一个formatC函数可以提供类似C语言的printf格式功能。

- formatC对输入向量的每一个元素单独进行格式转换而不生成统一格式，例如：

```
> formatC(c(1, 10000))
 [1] "1" "1e+004"
```

```
formatC(c((-1:1)/0, c(1, 100) * pi), width = 6, digits = 1)
## [1] "  -Inf" "   NaN" "   Inf" "     3" " 3e+02"
```

- **R的输出缺省显示在交互窗口。**
- 可以用sink()函数指定一个文件以把后续的输出转向到这个文件，并可用append参数指定是否要在文件末尾附加：

```
sink("E:/work/result.txt", append=TRUE)
ls()
d
sink()
```

调用无参数的sink()把输出恢复到交互窗口。

3.1.2 读取数据

- 在数据分析中，数据量一般是比较大的，变量也很多，如果用上述方法来建立数据集，是不可取的，上述方法适用于少量数据、少量变量的分析，对于大量数据和变量，一般应在其他软件中输入（或数据来源是其他软件的输出结果），再读到R中处理． R软件有多种读数据文件的方法。
- 另外，所有的计算结果也不应只在屏幕上输出，应当保存在文件中，以备使用。

读取R内置数据

- 自带的datasets数据包和其他R包内部数据的读取，可使用data()函数。
  - ✓data()函数不带任何参数，则列出R内部datasets数据包
  - ✓data()函数的package参数可以列出指定R包的数据集

```
# 可以先调用rpart包，再加载指定的stagec数据集：
library(rpart)
data(stagec)
# 也可以不加载包，直接加载数据：
data(stagec, package = "rpart")
```

读取纯文本文件

- 读纯文本文件有两个函数，一个是read.table()函数，另一个是scan()函数.
- read.table()函数是读表格形式的文件，若“住宅”数据已经输入一个纯文本文件”houses.data”中，其格式如下：

其中第一行为变量名，第一列为记录序号

	Price	Floor	Area	Rooms	Age	Cent.heat
01	52.00	111.0	830	5	6.2	no
02	54.75	128.0	710	5	7.5	no
03	57.50	101.0	1000	5	4.2	no
04	57.50	131.0	690	6	8.8	no
05	59.75	93.0	900	5	1.9	yes

读取纯文本文件

- 利用read.table()函数可读入数据，如
- rt<-read.table("houses.data")
- 此时变量rt是一个数据框,其形式与纯文本文件“houses.data”格式相同.

```
# 从网站地址url下载后读取
rt <- read.table("https://github.com/whcsu/rpstat/blob/main/JSdata.txt",
  header = TRUE)
# 读取本地文件
rt <- read.table("JSdata.txt", header = TRUE)
```

此时变量**rt**是一个数据框

读取纯文本文件

read.table函数的格式：

```
read.table(file, header = FALSE, sep = "", quote = "\"",
           dec = ".", row.names, col.names,
           as.is = !stringsAsFactors,
           na.strings = "NA", colClasses = NA, nrows = -1,
           skip = 0, check.names = TRUE, fill =
!blank.lines.skip,
           strip.white = FALSE, blank.lines.skip = TRUE,
           comment.char = "#",
           allowEscapes = FALSE, flush = FALSE,
           stringsAsFactors = default.stringsAsFactors(),
           fileEncoding = "", encoding = "unknown")
```

读取纯文本文件

- read.csv(),read.csv2()可以看做read.table()的变体
- read.csv()函数使用“.”作为小数点、逗号（“，”）作为分隔符，与使用参数sep=“,”的read.table()是等效的
- read.csv2()函数则使用逗号（“,”）作为小数点、分号（“;”）作为分隔符。
- 使用read.table、read.csv()读取中文数据时，若出现乱码，可尝试更改打开文件的encoding编码：

```
rt <- read.table("JSdata.txt", header = TRUE,
encoding = "UTF-8")
```

读其它格式的数据文件

- R软件除了可以读纯文本文件外，还可以读其他统计软件格式的数据，如'Epi Info', 'Minitab', 'S', 'SAS', 'SPSS', 'Stata', 'Systat', 'Weka'等，要读入其他格式数据库，必须先调入foreign模块，它不属于R的内在模块，需要在使用前调入.
- 调入的方法很简便，只需键入命令：
- library(foreign)

- 已知数据分别存成SPSS数据文件(" educ_salarys．sav")、SAS数据文件(" educ_salarys.xpt")、S-PLUS数据文件(" educ_salarys')和Stata数据文件 ("educ_salarys.dta").
- **读SPSS文件的格式是：**
- rs<-read.spss("educ_salarys.sav")
- 其变量rs是一个列表，如果打算形成数据框，则命令格式为
- rs<-read.spss("educ_salarys.sav",to.data.frame=TRUE)

读其它格式的数据文件

读SAS文件的格式是：

- rx<-read.xport("educ_salarys.xpt")
- 其变量rx

读S-PLUS文件的格式是：

- rs<-read.S("educ_salarys")
- 其变量rs是一个数据框。

读Stata文件的格式是：

- rd<-read.dta("educ_salarys.dta")
- 其变量rd是一个数据框。

读其它格式的数据文件

- 读取Excel数据文件
- 利用xlsx 、 XLConnect等R包读取
- library(xlsx)
- read.xlsx2("myfile.xlsx", sheetName = "Sheet1")

- library(XLConnect)
- wb <- loadWorkbook("myfile.xlsx")
- myDf <- readWorksheet(wb, sheet = "Sheet1", header = TRUE)

读取数据库数据

- R中有多种面向关系型数据库管理系统的接口，包括SQL Server、Access、MySql、Oracle、DB2、Sybase、SQLite、Teradata、PostgreSQL。一些包通过原生的数据库驱动来提供访问功能，另一些则是通过ODBC或JDBC来实现。
- 在R中通过RODBC包访问数据库比较流行，该方式允许R连接到任意一种ODBC驱动的数据库
- RODBC包允许R和一个通过ODBC连接的SQL数据库之间进行双向通信。即R不仅可以读取数据库的数据，也可以修改数据库中的数据。

读取数据库数据

R中RODBC包中包含的函数有

函　数	描　述
odbcConnect(*dsn*,uid="",pwd="")	建立一个到ODBC数据库的连接
sqlFetch(*channel*,*sqltable*)	读取ODBC数据库中的某个表到一个数据框中
sqlQuery(*channel*,*query*)	向ODBC数据库提交一个查询并返回结果
sqlSave(*channel*,*mydf*,tablename=*sqtable*,append=FALSE)	将数据框写入或更新（append=TRUE）到ODBC数据库的某个表中
sqlDrop(*channel*,*sqtable*)	删除ODBC数据库中的某个表
close(*channel*)	关闭连接

假设要读入一个数据库中的表A和B，程序如下

```
library(RODBC)
ch <- odbcConnect("mydsn",uid="Rob0,pwd="asb123")
#mydsn为数据源名称，uid为数据库用户名，pwd为密码
TA <- sqlFetch(ch ,A)
TB <- sqlQuery(ch ,"select * from B")
close(ch)
```

3.1.3 保存数据

- 创建了一个R数据框

```
dfdat<-data.frame(
Name=c("Alice","Becka","James","jeffrey","John"),
Age=c(13,13,12,13,12),
Height=c(56.5,65.3,57.3,62.5,59.0),
Weight=c(84.0,98.0,83.0,84.0,99.5))
```

R中保存数据的最简单方法之一是使用 **save()** 函数将其保存为 **RData**格式的文件。

```
save(dfdat, file = "dfdat.RData")
```

RData是二进制格式，存储方式非常高效，只能在R中才能打开。

- 与save()函数类似的命令还有save.image()和saveR
  ()等函数。
- 区别在于saveRDS()一般用于保存单个数据对象，s
  ()函数可以保存一个或多个数据对象，save.image
  ()可以保存整个工作空间的数据对象。
- saveRDS()保存的单个文件（“rds”格式）需要readRDS()读取，save()和save.image()函数保存的.RData可以用load()读取。

```
load(file = "dfdat.RData")
```

保存为文本文件

write.table()函数

- write.table(x, file = “”, append = FALSE, quote = TRUE, sep = “ ”, eol = “\n”, na = “NA”, dec = “.”, row.names = TRUE, col.names = TRUE, qmethod = c(“escape”, “double”), fileEncoding = “”)
- 其中x是数据，通常是矩阵，也可以是向量。file是文件名(缺省时文件名为”data”).
  append=TRUE时，在原文件上填加数据；否则(FALSE，缺省值)写一个新文件，其它参数见帮助文件。
- write.csv(),write.csv2()可以看做write.table()的变体，**w**
  ()，与参数 sep=“,” 的 write.table()是等效的。

保存为文本文件

- write.csv()类函数保存上述dfdata对象的代码如下：

write.csv(dfdat, file = "123.csv")

```
"","Name","Age","Height","Weight"
"1","Alice",13,56.5,84
"2","Becka",13,65.3,98
"3","James",12,57.3,83
"4","jeffrey",13,62.5,84
"5","John",12,59,99.5
```

write.csv(dfdat, file = "124.csv")

```
"";"Name";"Age";"Height";"Weight"
"1";"Alice";13;56,5;84
"2";"Becka";13;65,3;98
"3";"James";12;57,3;83
"4";"jeffrey";13;62,5;84
"5";"John";12;59;99,5
```

write.csv()和write.csv2()两者的区别在前者使用“.”作为小数点、逗号（“，”）作为分隔符，而后者使用逗号（“,”）作为小数点、分号（“;”）作为分隔符。

3.2 数据选择

- 数据操作基本函数

length	返回对象中元素的个数
names	显示数据的名称，对于数据框则是列名字
levels	因子向量的水平
dim	数据的维度
nrow	矩阵或数据框的行数
ncol	列数
rownames	数据的行名字
colnames	列名字
class	数据类型
mode	数据模式
head	数据的前n行
tail	数据的后n行
summary	显示对象的概要
attr	x的属性类型

3.2 数据选择

- 数据集查看函数例子

```
# 查看本书中自带的数据集 从网站地址url 下载后读取
# rt<-read.csv('https://github.com/whcsu/rpstat/blob/main/JSdata.csv',header
# = TRUE)
```

```
##查看本书中自带的数据集
JS=read.csv("JSdata.csv",header = T)
> dim(JS)
[1] 39 8
#查看所有变量名字
> names(JS)
[1] [1] "id"     "name"   "sex"    "birth"
"title"  "height" "weight" "salary"
#查看数据集前几行，默认为6
> head(JS,3)
```

3.2 数据选择

- 数据集查看函数例子

```
#变量属性（int整数,num数值）
str(JS)
## 'data.frame':    39 obs. of  8 variables:
##  $ id     : chr  "2021A001" "2021A002" "2021A003" "2021A004" ...
##  $ name   : chr  "王天赐" "高琪琪" "朱德宗" "杨子琪" ...
##  $ sex    : chr  "男" "女" "男" "女" ...
##  $ birth  : chr  "1972/4/8" "1973/5/12" "1995/7/18" "1985/1/8" ...
##  $ title  : chr  "教授" "副教授" "讲师" "讲师" ...
##  $ height: int  165 163 187 166 166 188 173 152 158 167 ...
##  $ weight: int  66 65 87 67 69 89 69 57 62 71 ...
##  $ salary: num  30.1 23.8 9.7 12.8 24.3 28.1 14.3 15.3 16.4 10.3 ...
```

3.2 数据选择

- 数据集查看函数例子

```
unique(JS$title)
#查看分类变量的水平
## [1] "教授"    "副教授" "讲师"    "助教"

#分类水平，不同水平的个数（=unique+sum功能）
table(JS$title)

#   副教授     讲师     教授     助教
#     15        9       11        4
```

思考题：

1、head(mtcars)仅显示数据集mtcars前几个观测值。

A、5

B、6 

C、7

D、8

3.2 数据选择

- 向量取子集
- 学习原子向量的取子集是最简单的，原子向量的取子集操作可以很容易地被引申运用到高维和其他更复杂的数据结构。
- 以下用一个简单的向量x来讲解不同的取子集方式。

```
##小数点后面的数实际标明了向量中元素的位置。
x <- c(2.1, 4.2, 3.3, 5.4)
```

你可以用如下六种索引方式对一个向量进行取子集操作

向量取子集

- **1. 正整数索引** 返回向量中特定位置的元素：

```
>x <- c(2.1, 4.2, 3.3, 5.4)
>x[c(3, 1)]
 [1] 3.3 2.1
>x[order(x)]
 [1] 2.1 3.3 4.2 5.4
>x[c(1, 1)] # 重复的索引返回重复的值
 [1] 2.1 2.1
>x[c(2.1, 2.9)] # 实数默认被去尾为整数
[ 1] 4.2 4.2
```

向量取子集

- **2. 负整数索引** 不会返回向量中特定位置的元素：

```
>x <- c(2.1, 4.2, 3.3, 5.4)
>x[-c(3, 1)]
[1] 4.2 5.4
```

注意：正整数和负整数不可以在同一个取子集操作中结合使用

```
>x[c(-1, 2)]
```

Error in x[c(-1, 2)] : only 0's may be mixed with negative subscripts

向量取子集

- **3.逻辑向量索引** 选择对应值为TRUE的元素。**这可能是最有用的取子集操作，**因为我们在代码中常常得到逻辑向量。

```
>x <- c(2.1, 4.2, 3.3, 5.4)
>x[c(TRUE, TRUE, FALSE, FALSE)]
 [1] 2.1 4.2
>x[x > 3]
[1] 4.2 3.3 5.4
```

如果使用的逻辑向量的长度比被取子集的向量长度短，逻辑向量会被**循环**到与该向量相同的长度。

```
>x[c(TRUE, FALSE)] #等同于
> x[c(TRUE, FALSE, TRUE, FALSE)]
  [1] 2.1 3.3
```

向量取子集

- **4.空索引** 返回原向量。这对向量取子集没有什么用处，可是对于矩阵，数据框和数组却非常有用(某行或某列留空)。

```
>x <- c(2.1, 4.2, 3.3, 5.4)
>x[]
  [1] 2.1 4.2 3.3 5.4
```

5.零索引 返回一个长度为零的向量。这个不常用，但是可以用来生成测试数据 。

```
>x <- c(2.1, 4.2, 3.3, 5.4)
>x[0]
  numeric(0)
```

向量取子集

- **6.字符串向量索引**。如果向量有名字，你也可以使用字符串向量索引返回与名字相匹配的元素留空)。

```
>x <- c(2.1, 4.2, 3.3, 5.4)
>(y <- setNames(x, letters[1:4]))
 a   b   c   d
2.1 4.2 3.3 5.4
>y[c("d", "c", "a")]
 d   c   a
5.4 3.3 2.1
```

列表取子集

- 对列表取子集与对向量取子集原理相同。

- 使用函数names(list)查看列表中每个item的名称，如果创建列表是无名列表项，那么该函数返回NULL；可以对该函数names(list)赋值，为列表的每个item命名。

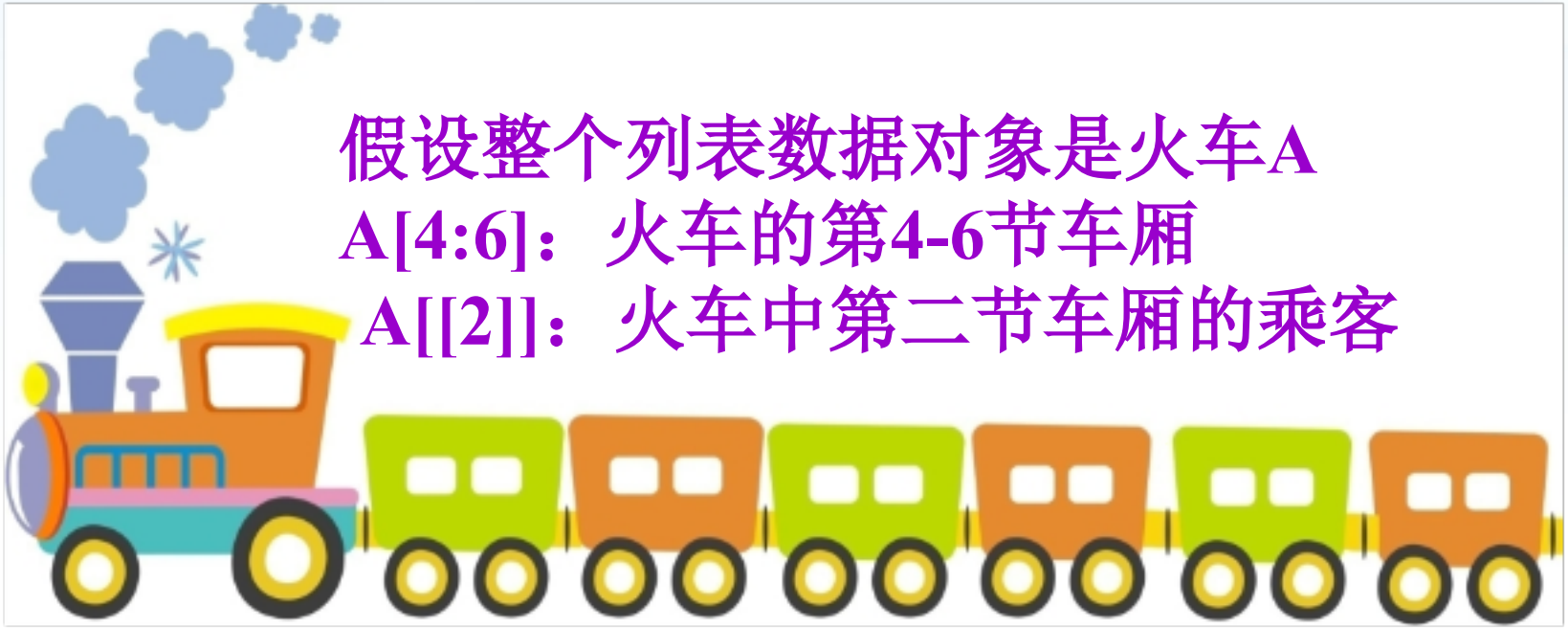
- 列表没有维度，列表和向量一样，有长度（length），可以使用函数length()获取列表的长度，列表的长度是列表的顶层item的数量，不包括嵌套的列表项。

列表取子集

在数据对象是列表时，[“单方括号”和[[“双方括号”和结果是不一样的。

用[将会返回一个列表；

[[和$则会提取一个列表中的元素。



列表取子集

>mylist <- list(c(1:3), month.abb, matrix(c(-1,-2,-3,-4),nrow=2))

```
> mylist
[[1]]
[1] 1 2 3

[[2]]
 [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"

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

>names(mylist) <- c('first','second','third')

```
> mylist
$first
[1] 1 2 3

$second
 [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"

$third
     [,1] [,2]
[1,]   -1   -3
[2,]   -2   -4
```

列表取子集

- 选择列表的第1项，第2项，分别使用正整数下标、负整数下标、元素名称和逻辑索引表示

```
>mylist[1:2]
>mylist[-3]
>mylist[c('first','second')]
>mylist[c(TURE,TRUE,FALSE)]
  $first [1]
     1 2 3
  $second [1]
  "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug"
"Sep" "Oct" "Nov" "Dec”
```

使用单个中括号索引列表项，返回的结果是一个**新的列表**，要访问列表元素的内容，有两种方式：使用**嵌套的中括号（传入正整数，代表列表项的下标，或传入字符串，代表列表项的名称），或指定元素的名称**。

列表取子集

- 访问列表元素

1、**通过嵌套的中括号和下标，**获取列表的第一个项目的值

```
>mylist[[1]]
[1] 1 2 3
```

2、使用列表的名称

```
mylist$first
[1] 1 2 3
```

3、访问列表项中的元素(在访问列表的元素之后，可以通过中括号访问列表项的元素值)

```
>mylist$first[2]
  [1] 2
```

矩阵取子集

- 矩阵是将数据按行和列组织数据的一种数据对象。与向量相似，矩阵的每个元素都拥有相同的数据类型。通常用列来表示不同的变量，用行来表示不同的观测。
- 可以使用类似向量的方法选择矩阵子集。

```
>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
```

矩阵取子集

```
> a  A  B  C
[1,] 1  4  7
[2,] 2  5  8
[3,] 3  6  9
```

a[1,2] #获取第一行第二列的元素

a[2,3] #获取第二行第三列的元素

a[1,] #获取第1行的所有数据

a[,2] #获取第2列的所有数据

a[2,c(1,3)] #获取第2行的1，3列的所有数据

a[1,2,drop=FALSE] #使返回值为矩阵而不是默认的向量

矩阵取子集


```
> a
     A B C
[1,] 1 4 7
[2,] 2 5 8
[3,] 3 6 9
```

>a$A #不能用$符号来取矩阵列元素

Error in a$A : $ operator is invalid for atomic vectors

> a[,'A'] #可以用中括号[]加列名字(如果有的话)来提取

[1] 1 2 3

矩阵取子集

```
> a<-matrix(rep(1:4,4),4,4);
> a
     [,1] [,2] [,3] [,4]
[1,]    1    1    1    1
[2,]    2    2    2    2
[3,]    3    3    3    3
[4,]    4    4    4    4
```

思考题：

已知矩阵a<-matrix(rep(1:4,4),4,4);若要将a转换成如下矩阵，应使用的命令是（）。

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

A. a[-(c(1,4)),-(3:4)]　　B. a[-(3:4),-c(1,4)]

C.a[-1,][-4,] [,3:4]　D. a[3:4,2:3]