

R的数据结构及数据输入

本章内容

- 创建数据集
- 基本的数据结构（向量、矩阵、数组、数据框、因子、列表）
- 数据的输入和导入（支持多种数据源）
- 处理数据对象的常用函数

创建数据集

创建数据集

- 按照个人要求的格式来创建含有研究信息的数据集，这是任何数据分析的第一步。在R中，这个任务包括以下两步：
 - 选择一种**数据结构**来存储数据；
 - 将数据**输入或导入**到这个数据结构中。
- 创建数据集后，往往需要对它进行**标注**，也就是为变量和变量代码添加描述性的标签。

数据集概念（1）

- 数据集（**data set**）是一个数据的集合，通常以数据库表格的形式出现。

病例数据

病人编号 (PatientID)	入院时间 (AdmDate)	年龄 (Age)	糖尿病类型 (Diabetes)	病情 (Status)
1	10/15/2009	25	Type1	Poor
2	11/01/2009	34	Type2	Improved
3	10/21/2009	28	Type1	Excellent
4	10/28/2009	52	Type1	Poor

数据集概念（2）

- 不同的行业对于数据集的行和列叫法不同。
 - 统计学家称它们为观测（**observation**）和变量（**variable**）
 - 数据库分析师则称其为记录（**record**）和字段（**field**）
 - 数据挖掘/机器学习学科的研究者则把它们叫做示例（**example**）和属性（**attribute**）

数据集概念（3）

- R的数据结构包括向量、数组、数据框和列表。
- R可以处理的数据类型（**types**）
（也叫**模式（modes）**）包括数值型、字符型、布尔型、复数型（虚数）和原生型（字节）。

基本的数据结构

（向量、矩阵、数组、数据框、
因子、列表）

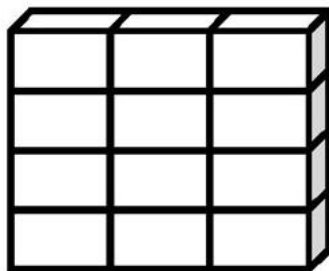
R的数据结构

向量、矩阵、数组的本质是**数组**，数组中的数据必须拥有**相同的模式**（数据类型）！！！！



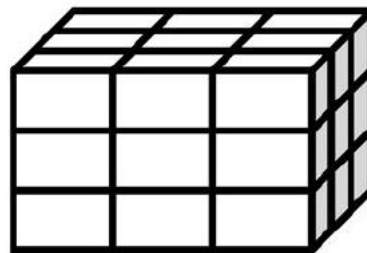
（一维数组）

(b) 矩阵



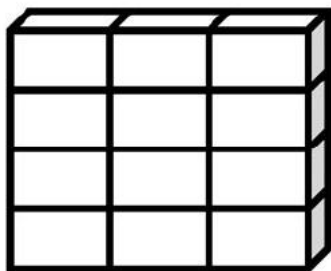
（二维数组）

(c) 数组



（多维数组）

(d) 数据框 (data frame)



各列的模式 (modes) 可以不同

(e) 列表



（列表即“广义表”）

向量（Vectors）

- 向量是用于存储数值型、字符型或逻辑型数据的一维数组。
- 执行组合功能的函数`c()`可用来创建向量。

```
# Creating vectors
```

```
a <- c(1, 2, 5, 3, 6, -2, 4)
```

```
b <- c("one", "two", "three")
```

```
c <- c(TRUE, TRUE, TRUE, FALSE, TRUE,  
FALSE)
```

- **标量（单个数值）是只含一个元素的向量，**
例如`f <- 3`、`g <- "US"`和`h <- TRUE`

访问向量中的元素

- **方括号[]**用于访问向量中的元素。
- 例： **a[c(2, 4)]**用于访问向量**a**中的第二个和第四个元素。
- 访问向量元素示例：

```
# Using vector subscripts
```

```
a <- c(1, 2, 5, 3, 6, -2, 4)
```

```
a[3]
```

```
a[c(1, 3, 5)]
```

```
a[2:6]
```

矩阵（Matrices）

- 矩阵是一个二维数组，每个元素都拥有相同的模式。通过函数**matrix**创建矩阵。
- 一般使用格式为：

```
myymatrix <- matrix(vector, nrow=number_of_rows, ncol=number_of_columns,  
                    byrow=logical_value, dimnames=list(  
                      char_vector_rownames, char_vector_colnames))
```

- 其中**vector**包含了矩阵的元素，**nrow**和**ncol**用以指定行和列的维数，**dimnames**包含了可选的、以字符型向量表示的行名和列名。选项**byrow**则表明矩阵应当按行填充（**byrow=TRUE**）还是按列填充（**byrow=FALSE**），默认情况下按列填充。

矩阵示例1

Creating Matrices

```
y <- matrix(1:20, nrow=5, ncol=4) # 1:20 相当于c(1,2,3,...,20)
```

y

要填充数据，行数，列数

```
cells <- c(1,26,24,68)
```

```
rnames <- c("R1", "R2")
```

```
cnames <- c("C1", "C2")
```

```
mymatrix <- matrix(cells, nrow=2, ncol=2, byrow=TRUE,  
                    dimnames=list(rnames, cnames))
```

mymatrix

```
mymatrix <- matrix(cells, nrow=2, ncol=2, byrow=FALSE,  
                    dimnames=list(rnames, cnames))
```

mymatrix

矩阵下标的使用

- 使用数组下标和**方括号**来按着矩阵中的行、列或单个元素。 $X[i,]$ 指矩阵 X 中的第 i 行， $X[,j]$ 指第 j 列， **$X[i, j]$** 指第 i 行第 j 个元素。选择多行或多列时，下标 i 和 j 可为数值型向量。

- 矩阵示例2

```
x <- matrix(1:10, nrow=2)
```

```
x
```

```
x[2,]
```

```
x[,2]
```

```
x[1,4]
```

```
x[1, c(4,5)]
```

数组（Arrays）

- 数组与矩阵类似，但是维度可以大于2。可通过**array**函数创建。

```
myarray <- array(vector, dimensions, dimnames)
```

- 其中**vector**包含了数组中的数据，**dimensions**是一个数值型向量，给出了各个维度下标的最大值，而**dimnames**是可选的、各维度名称标签的列表。
- 从数组中选取元素的方式与矩阵相同，使用**方括号**。例：**z[1,2,3]**。

数组示例

```
# Creating an array
```

```
dim1 <- c("A1", "A2")
```

```
dim2 <- c("B1", "B2", "B3")
```

```
dim3 <- c("C1", "C2", "C3", "C4")
```

```
z <- array(1:24, c(2,3,4),  
dimnames=list(dim1, dim2, dim3))
```

```
z
```

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


数据框（Data frames）

- 数据框不同的列可以包含不同模式（数值型、字符型等）的数据。是R中最常处理的数据结构。通过函数**data.frame()**创建。

```
mydata <- data.frame(col1, col2, col3,...)
```

- 其中的列向量**col1, col2, col3,...** 可为任何类型（如字符型、数值型或逻辑型）。每一列的名称可由函数**names**指定。

数据框示例1

#Creating a dataframe: 糖尿病患者数据

```
patientID <- c(1, 2, 3, 4)
```

```
age <- c(25, 34, 28, 52)
```

```
diabetes <- c("Type1", "Type2", "Type1", "Type1")
```

```
status <- c("Poor", "Improved", "Excellent", "Poor")
```

```
patientdata <- data.frame(patientID, age, diabetes,  
status)
```

```
patientdata
```

数据框示例2

#选取数据框中的元素

patientdata[1:2] #选取第1、2列

patientdata[c("diabetes","status")]

patientdata[2,] #选取第2行

patientdata\$age #选取age列

patientdata\$age[2] #选取age列的第二个

#记号\$用来选取一个给定数据框中的某个特定变量，相当于一般计算机语言中的“.”

attach()

- 函数**attach()**可将**数据框**添加到**R**的**搜索路径**中。R在遇到一个变量名以后(不必再使用**\$**)，将检查搜索路径中的数据框，以定位到这个变量。
- 例：（使用刚刚建立好的**patientdata**）
rm(age)
rm(status)
plot(age, status) #会找不到这两个对象，当然通过**\$**可以找到：plot(patientdata\$age, patientdata\$status)
attach(patientdata)
plot(age, status) #现在可以了

detach()

- 函数**detach()**将数据框从搜索路径中移除。
- 当**detach()**造成数据对象**重名**时，**原始对象将取得优先权**。
- 因此，函数**attach()**和**detach()**最好在分析一个单独的数据框，并且不太可能有多个同名对象时使用。

with(): attach()的替代

- 为了避免频繁attach、detach，可使用with。
- 例：

```
with(patientdata,{  
    plot(patientID,age)  
    plot(patientID,status)  
})
```

- 大括号{}之间的语句都是针对数据框patientdata执行，这样就无需担心名称冲突了。如果仅有一条语句，大括号可以省略。

因子（Factors）（1/3）

- 数据类型分为**类别**（nominal：糖尿病类型）、**有序**（ordinal：病情好中坏）、连续型（continuous：年龄）
- 在R中，**类别变量**和**有序变量**称为**因子**。因子在R中非常重要，因为它决定了数据的分析方式以及如何进行视觉呈现。

因子（Factors）（2/3）

- 函数**factor()**以一个整数向量的形式存储类别值。

```
diabetes <- c("Type1", "Type2", "Type1",  
"Type1")
```

```
str(diabetes) #str()函数会输出一个R对象的结构
```

```
#输出: chr [1:4] "Type1" "Type2" "Type1" "Type1"
```

```
diabetes <- factor(diabetes)
```

```
str(diabetes)
```

```
#输出: Factor w/ 2 levels "Type1","Type2": 1 2 1 1
```


因子（Factors）（3/3）

- 对于字符型向量，因子的水平默认依字母顺序创建。可以通过指定**levels**选项来覆盖默认排序。

```
status <- factor(status, order=TRUE,  
                  levels=c("Poor", "Improved", "Excellent"))
```

- 如果用**字符向量创建数据框**，R会将向量转换为因子。

列表 (Lists)

- 列表就是一些对象的有序集合。列表中可能是若干向量、矩阵、数据框，甚至其他列表的组合。例：

Creating a list

```
g <- "My First List"
```

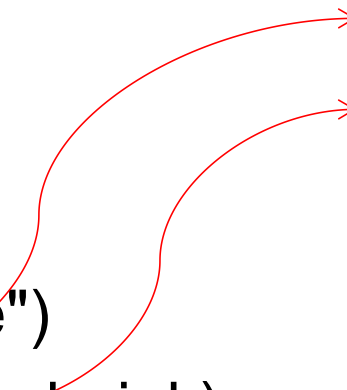
```
h <- c(25, 26, 18, 39)
```

```
j <- matrix(1:10, nrow=5)
```

```
k <- c("one", "two", "three")
```

```
mylist <- list(title=g, ages=h, j, k)
```

```
mylist
```



```
$title  
[1] "My First List"  
  
$ages  
[1] 25 26 18 39  
  
[[3]]  
      [,1] [,2]  
[1,]    1    6  
[2,]    2    7  
[3,]    3    8  
[4,]    4    9  
[5,]    5   10  
  
[[4]]  
[1] "one"  "two"  "three"
```

程序员注意！！（1/2）

- 对象名称中的点 (.) 没有特殊意义。但美元符号 (\$) 却有着和其他语言中的句点类似的含义，即指定一个对象中的某些部分。
- **R** 不提供多行注释或块注释功能。
- 将一个值赋给某个向量、矩阵、数组或列表中一个不存在的元素时，**R** 将自动扩展这个数据结构以容纳新值。例：

```
> x <- c(8, 6, 4)
> x[7] <- 10
> x
[1] 8 6 4 NA NA NA 10
```

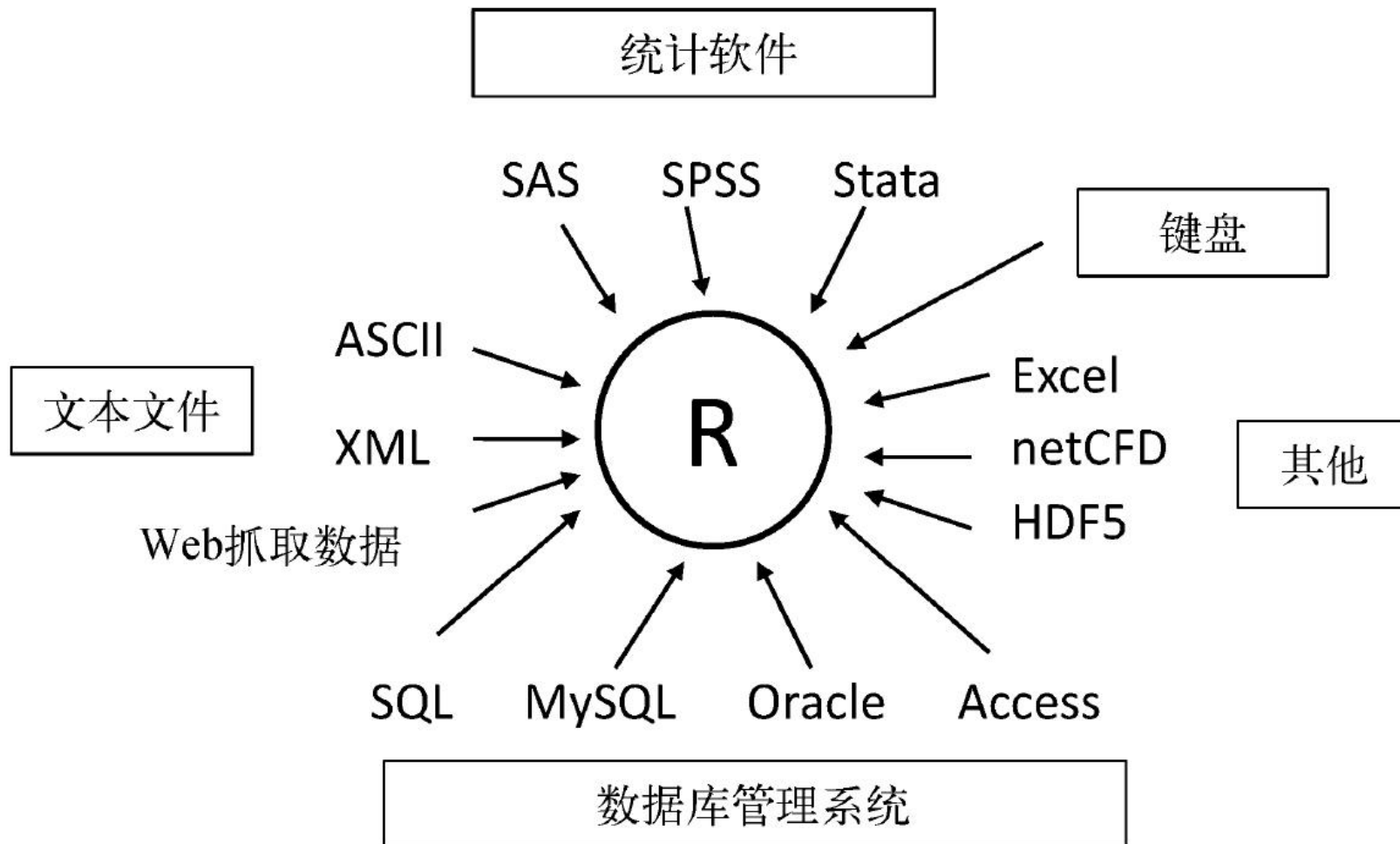
程序员注意！！（2/2）

即一个单个数值

- R中没有标量。标量以单元素向量的形式出现。
- R中的下标不从0开始，而从1开始。
- 变量无法被声明。它们在首次被赋值时生成。

数据的输入和导入

数据的输入



使用键盘输入数据

- `edit()`会自动调用一个允许手动输入数据的文本编辑器。例：

```
patientdata<-edit(patientdata)
```

- 函数`edit()`事实上是在对象的一个副本上进行操作的。如果不将其赋值到一个目标，所有修改将会全部丢失！
- 适用于小数据集

从带分隔符的文本文件导入数据 (**CSV**文件)

- **read.table()**从带分隔符的文本文件中导入数据

```
mydataframe <- read.table(file, header=logical_value,  
  sep="delimiter", row.names="name")
```

- **header**表明首行是否包含了变量名，**sep**指定分隔符，**row.names**是一个可选参数，用以指定一个或多个标识符。例1：

```
grades <- read.table("studentgrades.csv", header=TRUE, sep="," ,  
  row.names="STUDENTID")
```

例2: `flights <- read.table("flights14.csv", header=TRUE, sep=",")`

- 默认情况下，字符型变量将转换为因子。

导入Excel数据

- 读取一个Excel文件的最好方式，就是在Excel中将其导出为一个逗号分隔文件（csv），再使用上一页的方式导入R中。

导入XML数据

- R的XML包：
 - 可用来解析处理XML或是HTML数据
 - 还可以抓取网页数据
 - 最重要两个函数是xmlTreeParse()和getNodeSet()，前者负责抓取页面数据并形成树状结构，后者对抓取的数据根据XPath语法来选取特定的节点集合。

导入SPSS数据

- SPSS数据集可以通过foreign包中的函数read.spss()导入到R中，也可以使用Hmisc包中的spss.get()函数。
 - install.packages("Hmisc")
 - library(Hmisc)
 - mydataframe <- spss.get("mydata.sav", use.value.labels=TRUE)
- #mydata.sav是要导入的SPSS数据文件， use.value.labels=TRUE表示让函数将带有值标签的变量导入为R中水平对应相同的因子， mydataframe是导入后的R数据框。

导入SAS数据

- **SAS**的版本更新可能会导致**R**中导入**SAS**数据集的函数失效。可以采用以下方法：
 1. 以在**SAS**中使用**PROC EXPORT**将**SAS**数据集保存为一个逗号分隔的文本文件
 2. 再使用处理**csv**的方法

访问数据库管理系统（1/2）

- R有多种面向关系型数据库管理系统的接口
 - Microsoft SQL Server
 - MicrosoftAccess
 - MySQL
 - Oracle
 - DB2
 - PostgreSQL
 - Sybase
 - Teradata
 - SQLite

访问数据库管理系统（2/2）

- 例

```
library(RODBC) #使用微软ODBC接口
```

```
myconn <- odbcConnect("mydsn", uid="Rob",  
pwd="aardvark")
```

```
crimedat <- sqlFetch(myconn, Crime)
```

```
#sqlFetch()将Crime表复制到R数据框crimedat中
```

```
pundat <- sqlQuery(myconn, "select * from  
Punishment") #sqlQuery()非常强大，因为其中可以插入任意的有效SQL语句
```

```
close(myconn)
```

处理数据对象的常用函数

处理数据对象的常用函数（1/2）

函 数	用 途
<code>length(object)</code>	显示对象中元素/成分的数量
<code>dim(object)</code>	显示某个对象的维度
<code>str(object)</code>	显示某个对象的结构
<code>class(object)</code>	显示某个对象的类或类型
<code>mode(object)</code>	显示某个对象的模式
<code>names(object)</code>	显示某对象中各成分的名称
<code>c(object, object,...)</code>	将对象合并入一个向量

处理数据对象的常用函数（2/2）

函 数	用 途
<code>cbind(object, object, ...)</code>	按列合并对象
<code>rbind(object, object, ...)</code>	按行合并对象
<code>Object</code>	输出某个对象
<code>head(object)</code>	列出某个对象的开始部分
<code>tail(object)</code>	列出某个对象的最后部分
<code>ls()</code>	显示当前的对象列表
<code>rm(object, object, ...)</code>	删除一个或多个对象。语句 <code>rm(list = ls())</code> 将删除当前工作环境中的几乎所有对象*
<code>newobject <- edit(object)</code>	编辑对象并另存为newobject
<code>fix(object)</code>	直接编辑对象