第 3 章

Chapter 3

在 R 中加载及处理数据

学习成果

通过本章的学习，您将能够：

- 将不同类型的数据存储为向量(vector)、矩阵(matrixe)和列表(list)；
- 从 csv 文件、电子表格(spreadsheet)、Web、JASON 文档和 XML 中加载数据；
- 处理缺失及无效的数据；
- 在数据上运行 R 函数(sum()、min()、max()、rep()、grep()、substr()、strsplit()等)；
- 用 R 访问数据库，如 MySQL、PostgreSQL、SQLlite 和 JasperDB；
- 创建可视化，以加深对数据的理解。

3.1 概述

如今，企业应用程序产生了大量的数据。对这些数据进行分析可以得出有用的见解，从而帮助决策者做出更好和更快的决策。本章将介绍 R 支持的不同的数据类型，如数字、文本、逻辑值、日期等。同时介绍各种 R 对象，如向量、矩阵、列表、数据集等，以及如何使用 R 函数 sum()、min()、max()、rep()和字符串函数 substr()、grep()、strclip()等操作数据。探讨将 csv(comma separated values)文件、电子表格、XML 文档、JASON(JavaScript Object Notation)文档、Web 数据等导入 R，以及 R 与 MySQL、PostGreSQL、SQLite 等数据库的连接方式。数据分析中存在很多挑战，例如数据并不总是同质的，即数据的来源不同，并且格式也不同。在保证数据质量的同时会带来若干挑战，利益相关者也会从各种角度观察数据，并且会产生不同的需求。

3.2 分析数据处理的挑战

分析数据处理是商业智能的一部分，包括关系数据库、数据仓库、数据挖掘和报告挖掘，这是一种计算机处理技术，可以处理不同类型的业务，如销售、预算、财务报告、管理报告等，以上这些处理技术都需要大数据技术的支持。

商业分析结合了大数据技术，在商业数据分析过程中出现了不同的挑战。然而，这些挑战大多与数据有关，它们在项目的早期阶段就出现了。

3.2.1　数据格式

数据是商业分析的主要元素。商业分析使用数据集(sets of data)存储大量的数据。对研究人员或开发者而言，选择数据格式是分析数据处理中的首要挑战。分析数据处理需要一个完整的数据集，在没有数据集的情况下，开发人员会在进一步的处理中遇到问题。

R 是一种文档健全的编程语言，它将数据存储成对象的形式。R 有一个非常简单的语法，有助于处理任何类型的数据。R 具有许多软件包和功能，如可以处理数据格式类型不同的开放数据库的连接(ODBC)，ODBC 支持 CSV、MS Excel、SQL 等数据格式。

3.2.2　数据质量

保证数据质量是分析数据处理的另一个挑战，它要求业务分析师提供完美的信息推断、异常值及没有任何缺失值的输出。输入或输出较差的数据一定会给出不符合质量要求的结果。

在 R 的帮助下，业务分析师可以保证数据质量。不同的 R 工具可以帮助业务分析师删除无效数据、替换缺失值和删除数据中的异常值。

3.2.3　项目范围

基于分析数据处理的项目成本高，并且耗时长，因此在启动新项目前，业务分析师应该分析项目的范围，确定所需外部数据的数量、交付时间和与项目有关的其他参数。

3.2.4　利益方期望的输出结果的管理

在分析数据处理中，分析人员设计的项目会产生不同类型的输出，如 p-value、自由度等。但是，用户或利益方更希望看到输出。利益方不希望在数据处理、设想、假设、p-value、卡方值(chi-square value)或任何其他值中看到约束。因此，一个分析项目应努力满足利益方的所有期望。

业务分析师应该使用透明的方法和处理流程，也应该使用交叉验证的方法验证数据。如果业务分析师使用分析数据处理的标准步骤产生完美的输出，则不会遇到任何问题。数据输入、处理、描述性统计、数据可视化、报告生成和输出构成了分析数据处理的顺序，分析人员在对项目进行业务分析时应该遵循这个流程。

小练习

1. 什么是分析数据处理？

答：分析数据处理是业务智能化的一部分，包括关系数据库、数据仓库、数据挖掘和报告挖掘。

2. 列出分析数据处理中的挑战。

答：分析数据处理中的一些挑战包括数据格式、数据质量、项目范围、利益方期望的输

出结果的管理。

3. 分析数据处理的一般步骤是什么?

答:数据输入、处理、描述性统计、数据可视化、报告生成和输出是分析数据处理的一般步骤。

3.3 表达式、变量和函数

首先熟悉一下 R 的接口,从练习表达式、变量和函数开始。

3.3.1 表达式

观察几个算术运算,如表 3.1 给出的加法、减法、乘法、除法、求幂、取余(模运算)、整除和求平方根。

表 3.1 算术运算

运 算	操 作 符	描 述	示 例
加法	x + y	y 加上 x	> 4 + 8 [1] 12
减法	x − y	x 减去 y	> 10 − 3 [1] 7
乘法	x * y	x 乘以 y	> 7 * 8 [1] 56
除法	x / y	x 除以 y	< 8/3 [1] 2.666667
指数运算	x ^ y x ** y	x 的 y 次幂	> 2 ^ 5 [1] 32 或 >2 ** 5 [1] 32
模运算	x %% y	x 除以 y 取余	> 5 %% 3 [1] 2
整除	x%/%y	x 除以 y 取整	> 5 %/% 2 [1] 2
平方根运算	sqrt(x)	x 的平方根	> sqrt (25) [1] 5

3.3.2 逻辑值

逻辑值可以表示为 TRUE 和 FALSE 或者 T 和 F,要注意它们是大小写敏感的,等号操作符是==。

```
>8 <4
[1] FALSE
```

```
>3 * 2 ==5
[1] FALSE
>3 * 2 ==6
[1] TRUE
>F ==FALSE
[1] TRUE
>T ==TRUE
[1] TRUE
```

1. 引导活动

步骤 1：创建一个向量 x，由 1～10 共 10 个元素组成。3.5 节会讨论创建向量、访问向量元素和向量的算术运算等的方法。

```
>x <-c(1:10)
```

步骤 2：显示向量 x 的内容。

```
>x
[1] 1 2 3 4 5 6 7 8 9 10
```

步骤 3：打印输出大于 7 或小于 5 的元素。“|”是或(OR)操作符，使用 OR 操作符显示大于 7 或小于 5 的元素。

```
>x[(x>7) | (x<5)]
[1] 1 2 3 4 8 9 10
```

2. 解释

- 当元素值大于 7 时显示 TRUE，否则显示 FALSE。

```
>x>7
[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE TRUE TRUE
```

- 当元素值小于 5 时显示 TRUE，否则显示 FALSE。

```
>x<5
[1] TRUE TRUE TRUE TRUE FALSE FALSE FALSE FALSE FALSE FALSE
```

步骤 4：打印输出值大于 7 且小于 10 的元素。“&”是与(AND)操作符，使用 AND 操作符显示大于 7 且小于 10 的元素。

```
>x[(x>7) & (x<10)]
[1] 8 9
```

3.3.3　日期

默认的日期格式为 YYYY-MM-DD。

(1) 打印系统日期

```
>Sys.Date()
[1] "2017-01-13"
```

(2) 打印系统时间

```
>Sys.time()
[1] "2017-01-13 10:54:37 IST"
```

(3) 打印时区

```
>Sys.timezone()
[1] "Asia/Calcutta"
```

(4) 打印当天日期

```
>today <-Sys.Date()
>today
[1] "2017-01-13"
>format (today, format ="%B %d %Y")
[1] "January 13 2017"
```

(5) 将日期存储为文本数据类型

```
>CustomDate ="2016-01-13"
>CustomDate
[1] "2016-01-13"
>class (CustomDate)
[1] "character"
```

(6) 将存储为文本类型的日期转换成日期数据类型

```
>CustDate =as.Date(CustomDate)
>class(CustDate)
[1] "Date"
>CustDate
[1] "2016-01-13"
```

(7) 找出以下两个日期的差

```
>strDates <-c("08/15/1947", "01/26/1950")
```

(8) 将字符串转换成日期格式

```
>dates =as.Date(strDates, "%m /%d /%Y")
>dates
[1] "1947-08-15" "1950-01-26"
```

(9) 计算两个日期的差

```
>dates[2] -dates[1]
Time difference of 895 days
```

3.3.4　变量

(1) 为变量 Var 赋值 50

```
>Var <-50
```

或

```
>Var=50
```

(2) 打印变量 Var 的值

```
>Var
[1] 50
```

(3) 对变量 Var 执行算术运算

```
>Var +10
[1] 60
>Var / 2
[1] 25
```

变量可以被重新赋值,新赋值的类型可以与之前相同,也可以不同。

(4) 重新为变量 Var 赋一个 string 类型的值

```
>Var <-"R is a Statistical Programming Language"
```

打印变量 Var 的值。

```
>Var
[1] "R is a Statistical Programming Language"
```

(5) 为 Var 变量赋一个逻辑类型的值

```
>Var <-TRUE
>Var
[1] TRUE
```

3.3.5　函数

本节将介绍一些函数,如 sum()、min()、max()和 seq()。

1. sum()函数

sum()函数可以返回其参数表中所有值的和。

语法

```
sum(…, na.rm =FALSE)
```

其中,"…"表示数字、复数或逻辑向量。na.rm 接收一个逻辑值后,缺失值(包括 NaN(Not a Number))会被移除吗?

例如

① 对sum()函数的参数值“1”“2”和“3”进行求和。

```
>sum(1, 2, 3)
[1] 6
```

② 如果将NA作为sum()函数的一个参数，则输出将会是什么样的？

```
>sum(1, 5, NA, na.rm=FALSE)
[1] NA
```

如果na.rm是FALSE，NA或NaN作为任意一个参数，则会返回NA或NaN。

③ 如果将NaN作为sum()函数的一个参数，则输出将会是什么样的？

```
>sum(1, 5, NaN, na.rm=FALSE)
[1] NaN
```

④ 如果将NA和NaN作为sum()函数的参数，则输出将会是什么样的？

```
>sum(1, 5, NA, NaN, na.rm=FALSE)
[1] NA
```

⑤ 如果将选项na.rm设置为TRUE，则输出将会是什么样的？

如果na.rm是TRUE，则参数中的任何NA或NaN将会被忽略。

```
>sum(1, 5, NA, na.rm=TRUE)
[1] 6
>sum(1, 5, NA, NaN, na.rm=TRUE)
[1] 6
```

2. min()函数

min()函数可以返回参数表中所有值的最小值。

语法

```
min(…, na.rm=FALSE)
```

其中，“…”表示数字或字符参数。na.rm接收一个逻辑值后，缺失值(包括NaN)会被移除吗？

例如

```
>min(1, 2, 3)
[1] 1
```

如果na.rm是FALSE，参数表中的任意一个参数为NA或NaN，则其返回结果为NA或NaN。

```
>min(1, 2, 3, NA, na.rm=FALSE)
[1] NA
>min(1, 2, 3, NaN, na.rm=FALSE)
```

```
[1] NaN
>min(1, 2, 3, NA, NaN, na.rm=FALSE)
[1] NA
```

如果 na.rm 为 TRUE,则参数表中的任意一个 NA 或 NaN 将会被忽略。

```
>min(1, 2, 3, NA, NaN, na.rm=TRUE)
[1] 1
```

3. max()函数

max()函数可以返回参数表中所有值的最大值。

语法

```
max(…, na.rm=FALSE)
```

其中,"…"表示数字或字符参数。na.rm 接收一个逻辑值后,缺失值(包括 NaN)会被移除吗?

例如

```
>max(44, 78, 66)
[1] 78
```

如果 na.rm 是 FALSE,参数表中的任意一个参数为 NA 或 NaN,则会导致返回结果为 NA 或 NaN。

```
>max(44, 78, 66, NA, na.rm=FALSE)
[1] NA
>max(44, 78, 66, NaN, na.rm=FALSE)
[1] NaN
>max(44, 78, 66, NA, NaN, na.rm=FALSE)
[1] NA
```

如果 na.rm 为 TRUE,则参数表中的任意一个 NA 或 NaN 将会被忽略。

```
>max(44, 78, 66, NA, NaN, na.rm=TRUE)
[1] 78
```

4. seq()函数

seq()函数可以生成一个规则序列。

语法

```
seq(start from, end at, interval, length.out)
```

其中,start from 表示序列的起始值;end at 表示序列的最大值或结束值;interval 表示序列的增量;length.out 表示序列的期望长度。

例如

```
>seq(1, 10, 2)
```

```
[1] 1 3 5 7 9
>seq(1, 10, length.out=10)
[1] 1 2 3 4 5 6 7 8 9 10
>seq(18)
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
```

或者

```
>seq_len(18)
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
>seq(1, 6, by=3)
[1] 1 4
```

3.3.6 处理数据中的文本

R 中有许多内置的字符串函数可以用来处理文本或字符串。查找某个文本字符串的一部分,在文本中搜索某个字符串或连接字符串及其他类似的操作都属于处理文本的操作。表 3.2 介绍了一些重要的文本处理操作。

让我们来看看 R 是如何处理字符串的。

字符串必须用双引号括起来,如

```
>"R is a statistical programming language"
[1] "R is a statistical programming language"
```

表 3.2 R 中内置的文本操作

函　数	参　数	描　述
substr(a,start, stop)	• a 是一个字符向量 • start and stop 包含一个数字值	函数返回字符串中从 start 参数开始、结束于 stop 参数的一部分
strsplit(a, split, …)	• a 是一个字符向量 • split 也是一个字符向量,其包含一个用来拆分的规则表达式	函数将给定的文本串拆分为子串
paste(…,sep='', …)	• …定义了 R 对象 • sep 参数是一个用来分割对象的字符串	函数在将对象转换为字符串后连接字符串向量
grep(pattern, a)	• pattern 参数包含一个匹配模式 • a 是一个字符向量	该函数在给定的文本字符串中搜索文本模式后返回字符串
toupper(a)	• a 是一个字符向量	该函数将字符串转换成大写
tolower(a)	• a 是一个字符向量	该函数将字符串转换成小写

图 3.1 描述了 R 工作空间中的 strsplit()函数和 grep()函数。

下面详细介绍几个字符串函数。

1. rep()函数

rep()函数可以重复参数中给定的次数。在下面的示例中,字符串"statistics"被重复了

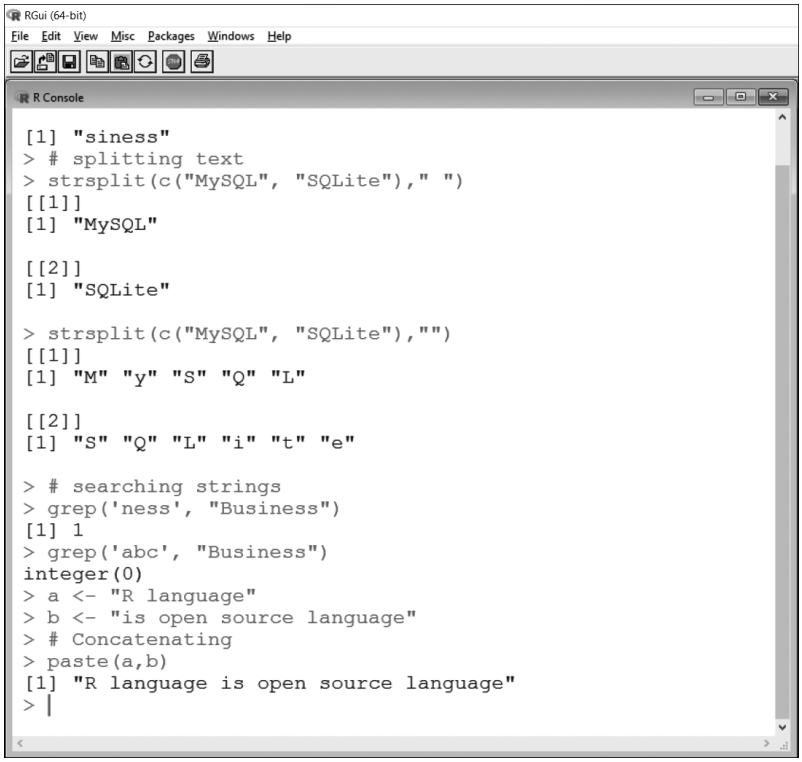


图 3.1　字符串操作函数示例

三次。

例如

```
>rep("statistics", 3)
[1] "statistics" "statistics" "statistics"
```

2. grep()函数

在下面的示例中，grep()函数用来查找"statistical"字符串所在的索引位置。

例如

```
>grep("statistical",c("R","is","a","statistical","language"),
fixed=TRUE)
[1] 4
```

3. toupper()函数

toupper()函数可以将给定的字符向量转换为大写。

语法

```
toupper(x)
```

x 是一个字符向量。

例如

```
>toupper("statistics")
[1] "STATISTICS"
```

或者

```
>casefold ("r programming language", upper=TRUE)
[1] "R PROGRAMMING LANGUAGE"
```

4. tolower()函数

tolower()函数可以将给定的字符向量转换为小写。

语法

```
tolower(x)
```

x是一个字符向量。

例如

```
>tolower("STATISTICS")
[1] "statistics"
```

或者

```
>casefold("R PROGRAMMING LANGUAGE", upper=FALSE)
[1] "r programming language"
```

5. substr()函数

substr()函数可以提取或替换字符向量中的子串。

语法

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

x表示字符向量;start表示提取或替换的起始位置;stop表示提取或替换的结束位置。

例如

从“statistics”中提取字符串"tic",从位置7开始提取,一直提取到位置9。

```
>substr("statistics", 7, 9)
[1] "tic"
```

3.4 R中缺失值的处理

在分析数据处理过程中,用户会遇到由于缺失值和无穷大值所引起的问题。为了获得准确的输出,用户应该删除或清洗缺失值。在R中,NA(Not Available)代表缺失值,Inf(Infinite)代表无穷大值。在处理过程中,R提供了识别缺失值的不同函数(表3.3)。

表 3.3　处理缺失值的函数

函　数	参　数	描　述
is.na(x)	x 是一个将要测试的 R 对象	该函数会检查对象，如果数据缺失，则返回 TRUE
na.omit(x, …)	x 是一个需要从中删除 NA 的 R 对象，“…”定义了其他可选参数	从中删除缺失值后，函数返回该对象
na.exclude(x, …)	x 是一个需要从中删除 NA 的 R 对象，“…”定义了其他可选参数	从中删除缺失值后，函数返回该对象
na.fail(x, …)	x 是一个需要从中删除 NA 的 R 对象，“…”定义了其他可选参数	如果对象中包含缺失值，则函数会产生一个错误；如果对象中不包含任何缺失值，则返回对象
na.pass(x, …)	x 是一个需要从中删除 NA 的 R 对象，“…”定义了其他可选参数	函数会返回没有变化的对象

下面的示例创建了一个有缺失值的向量 A[10,20,NA,40]（如图 3.2 所示）。is.na(A)对于缺失值返回 TRUE；na.omit(A)和 na.exclude(A)会移除缺失值并将其分别存储到向量 B 和向量 D 中；如果 A 中存在缺失值，则 na.fail(A)会产生错误，na.pass(A)会返回正常向量 A。

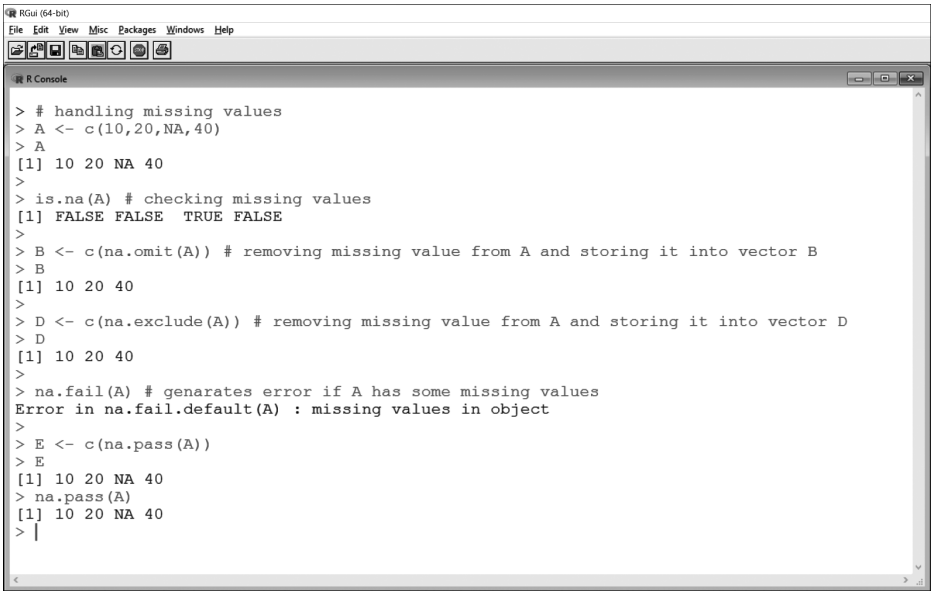


图 3.2　处理缺失值

3.5　利用 as 操作符改变数据的结构

有时，分析数据处理需要将数据从一种格式转换成另一种格式。通常，分析数据处理以表格的形式存储数据，其中只需要表的一部分或另一种结构存储表中的数据。在这种情况下，R 可以将表的结构转换为其他结构，如因子（factor）、列表（list）等。

R 中的 as 操作符为将数据集从一种结构转换成另一种结构提供了方便，使用该操作符的语法为

```
as.objecttype(objectname)
```

其中，objecttype 是对象的类型，如数据框、矩阵、列表等；objectname 是需要转换成另外一种格式的对象名。

as.numeric()函数和 as.character()函数分别用于将对象转换成数值型和字符型。

下面的示例利用两个向量 a 和 b 创建了一个数据框 D(如图 3.3 所示)，现在利用 as.list(D)函数将数据框转换成列表 B，利用 as.matrix(D)函数将数据框转换成 matrix。

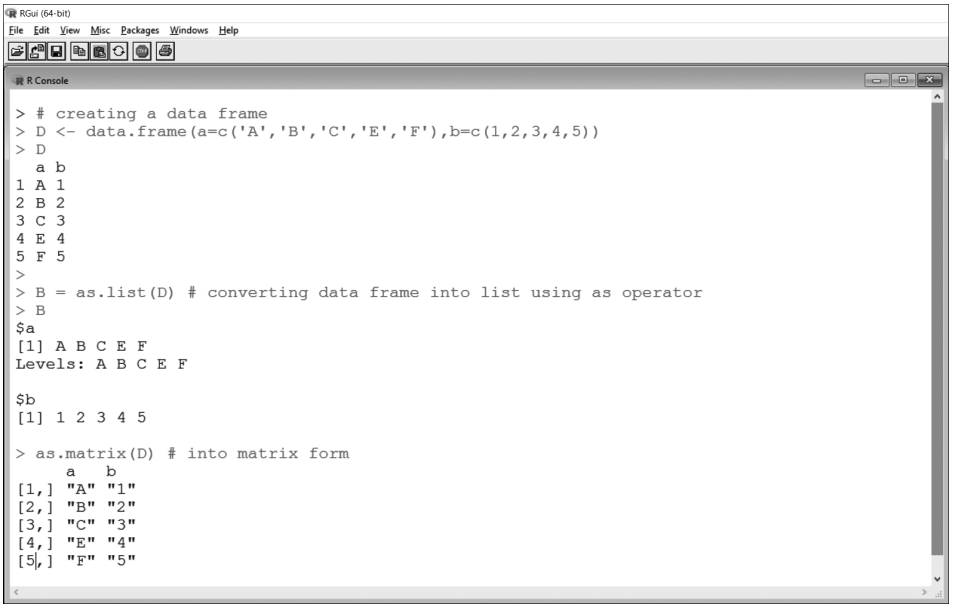


图 3.3　as 操作符的使用

小练习

1. na.omit()函数是什么？

答：na.omit()函数是 R 中内置的函数，它可以返回移除缺失值后的对象。

2. na.exclude()函数是什么？

答：na.omit()函数是 R 中内置的函数，它可以返回移除缺失值后的对象。

3. na.fail()函数是什么？

答：na.fail()函数是 R 中内置的函数，如果对象中包含缺失值，则它会显示错误，如果没有任何缺失值则返回对象。

4. 哪一个函数可用于检测 R 对象中的缺失值？

答：is.na()函数可以用于检测 R 对象中的缺失值，该函数会检测对象，如果对象中有缺失值，则返回 true。

5. as 操作符是什么？

答：as 操作符可以利用 R 将一个数据集从一种结构转换成另一种结构。

3.6　向量

一个向量可以有一个值列表。值可以是数字、字符串或逻辑值。一个向量中的所有值应该具有相同的数据类型。

关于 R 中的向量，需要记住的有以下几点：

- 向量像数组一样存储在 C 中；
- 向量的索引从 1 开始；
- 所有向量元素必须具有相同的模式，如整型、数值型（浮点型）、字符型（字符串）、逻辑值（boolean）、复数（complex）、对象（object）等。

下面创建几个向量。

（1）创建一个数值型的向量。

```
>c(4, 7, 8)
[1] 4 7 8
```

c 函数（c 是 combine 的缩写）创建了一个包含 3 个值（4，7 和 8）的新向量。

（2）创建一个字符串类型的向量。

```
>c("R", "SAS", "SPSS")
[1] "R" "SAS" "SPSS"
```

（3）创建一个逻辑值的向量。

```
>c(TRUE, FALSE)
[1] TRUE FALSE
```

一个向量中不能有不同数据类型的值，考虑将下面例子中的整型、字符串、布尔型值放在一个向量中。

```
>c(4, 8, "R", FALSE)
[1] "4" "8" "R" "FALSE"
```

> **小提示**：将所有的值转换成相同的数据类型，如字符型。

（4）声明一个名为 Project，长度为 3 的向量，并为其赋值。

```
>Project <-vector(length =3)
>Project [1] <-"Finance Project"
>Project [2] <-"Retail Project"
>Project [3] <-"Energy Project"
```

输出结果

```
>Project
[1] "Finance Project" "Retail Project" "Energy Project"
```

```
>length (Project)
[1] 3
```

3.6.1 顺序向量

顺序向量可以通过一个“开始:结束”标记进行创建。

目标

创建一个在1～5之间(包括1和5)的顺序数字向量。

```
>1:5
[1] 1 2 3 4 5
```

或者

```
>seq(1:5)
[1] 1 2 3 4 5
```

默认的增量为1,但是也允许增量不为1。

```
>seq (1, 10, 2)
[1] 1 3 5 7 9
```

或者

```
>seq (from=1, to=10, by=2)
[1] 1 3 5 7 9
```

或者

```
>seq (1, 10, by=2)
[1] 1 3 5 7 9
```

seq也可以按降序生成数字序列。

```
>10:1
[1] 10 9 8 7 6 5 4 3 2 1
>seq (10, 1, by=-2)
[1] 10 8 6 4 2
```

3.6.2 rep()函数

rep()函数用于将相同的常数放入长向量中,其语法是rep(z,k),它创建了一个具有k * length(z)个元素的向量,每个元素都等于z。

目标

演示rep()函数的功能。

动作

```
>rep (3, 4)
[1] 3 3 3 3
```

或者

```
>x <- rep (3, 4)
>x
[1] 3 3 3 3
```

3.6.3　向量访问

目标

创建一个变量 VariableSeq,并为它赋值一个字符串向量。

```
>VariableSeq <- c ("R", "is", "a", "programming", "language")
```

目标

要想访问向量中的值,需要指定对应值在向量中的索引,索引从 1 开始。

```
>VariableSeq[1]
[1] "R"
>VariableSeq[2]
[1] "is"
>VariableSeq[3]
[1] "a"
>VariableSeq[4]
[1] "programming"
>VariableSeq[5]
[1] "language"
```

目标

为已有的向量赋新值。例如,将"good programming"赋值给已有向量 VariableSeq 中索引为 4 的元素。

```
>VariableSeq[4] <- "good programming"
```

输出结果

```
>VariableSeq[4]
[1] "good programming"
```

目标

访问向量中的多个值。

① 从向量 VariableSeq 中访问第 1 个元素和第 5 个元素。

```
>VariableSeq[c(1, 5)]
[1] "R" "language"
```

② 从向量 VariableSeq 中访问第 1～4 个元素。

```
>VariableSeq[1:4]
[1] "R" "is" "a" "good programming"
```

③ 从向量 VariableSeq 中访问第 1 个、第 4 个和第 5 个元素。

```
>VariableSeq[c(1, 4:5)]
[1] "R" "good programming" "language"
```

④ 从向量 VariableSeq 中抽取所有的值。

```
>VariableSeq
[1] "R" "is" "a" "good programming"
[5] "language"
```

3.6.4 向量名

names()函数可以为向量元素命名。

这个工作分为两步完成,如下所示。

```
>placeholder <-1:5
>names(placeholder) <-c("r", "is", "a", "programming",
"language")
```

然后利用索引位置可以将向量元素抽取出来。

```
>placeholder
    r     is     a     programming     language
    1     2      3     4               5
>placeholder [3]
a
3
>placeholder [1]
r
1
>placeholder[4:5]
programming     language
           4            5
```

或者

```
>placeholder ["programming"]
programming
           4
```

目标

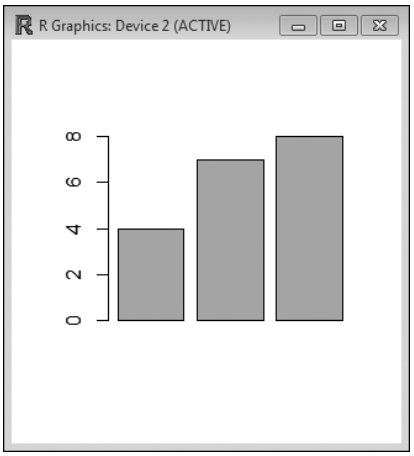
利用 barplot()函数绘制一张柱状图,barplot()函数使用一个向量的值绘制柱状图。

动作

所使用的向量为 BarVector。

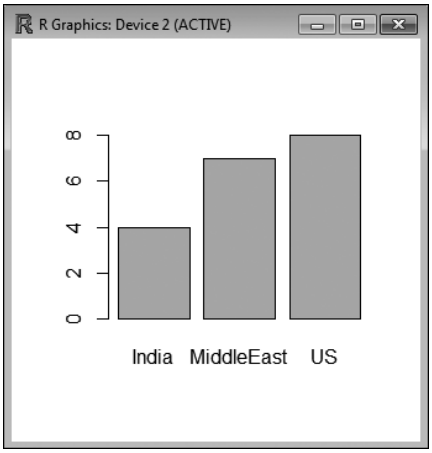
```
>BarVector <-c(4, 7, 8)
>barplot(BarVector)
```

输出结果



使用命名函数为向量元素命名，这些名称将会在柱状图中作为标签。

```
>names(BarVector) <-c("India", "MiddleEast", "US")
>barplot(BarVector)
```



3.6.5　向量的算术运算

定义一个有 3 个值的向量 x，对向量加一个标量值(单个值)，该标量值会加到每一个向量元素上。

```
>x <-c(4, 7, 8)
>x +1
5 8 9
```

但是，向量还是保持它本身的元素。

```
>x
[1] 4 7 8
```

如果向量需要更新为新值,则输入以下的表达。

```
>x <- x +1
>x
[1] 5 8 9
```

可以对向量执行如下算术操作。

```
>x -1
[1] 4 7 8
>x * 2
[1] 10 16 18
>x / 2
[1] 2.5 4.0 4.5
```

在两个向量上执行这些算术运算。

```
>x
[1] 5 8 9
>y <- c(1, 2, 3)
>y
[1] 1 2 3
>x +y
[1] 6 10 12
```

其他算术操作有:

```
>x -y
[1] 4 6 6
>x * y
[1] 5 16 27
```

检测两个向量是否相等,比较是逐个元素进行的。

```
>x
[1] 5 8 9
>y
[1] 1 2 3
>x==y
[1] FALSE FALSE FALSE
>x <y
[1] FALSE FALSE FALSE
>sin(x)
[1] -0.9589243 0.9893582 0.4121185
```

3.6.6 向量循环

如果执行的一个操作涉及两个向量,则要求它们的长度相同,较短的向量就会被循环,即重复到它足够长以匹配较长的向量。

目标

两个向量相加，其中一个向量的长度是 3，另一个向量的长度是 6。

```
>c(1, 2, 3) +c(4, 5, 6, 7, 8, 9)
[1] 5 7 9 8 10 12
```

目标

两个向量相乘，其中一个向量的长度是 3，另一个向量的长度是 6。

```
>c(1, 2, 3) * c(4, 5, 6, 7, 8, 9)
[1] 4 10 18 7 16 27
```

目标

绘制一个散点图。绘制散点图的函数是 plot()，该函数使用两个向量，即一个为 x 轴，另一个为 y 轴，目的是理解数字与其正弦值(sin)之间的关系。向量 x 是元素间隔为 0.1、取值为 1～25 的顺序值，y 存储向量 x 中的所有值的正弦值。

```
>x <-seq(1, 25, 0.1)
>y <-sin(x)
```

plot()函数取向量 x 中的值并将其绘制在横轴上，然后取 y 向量中的值并将其置于纵轴上(如图 3.4 所示)。

```
>plot(x, y)
```

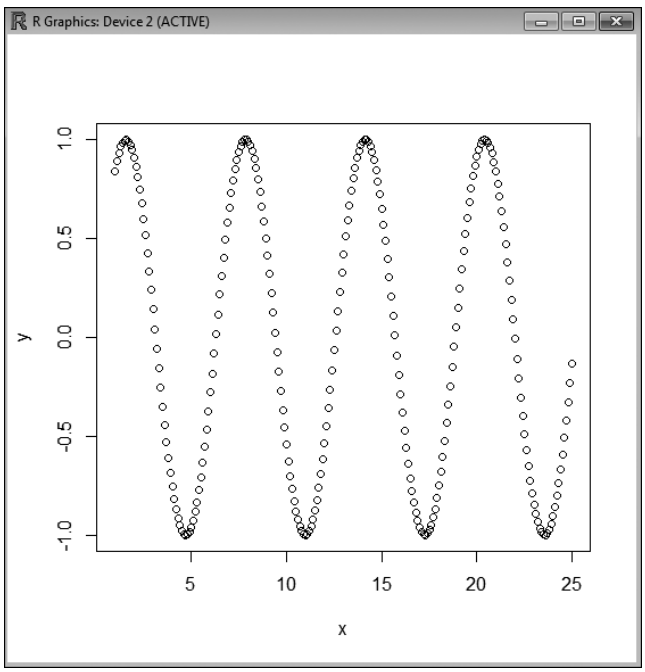


图 3.4　散点图