

第一章 R 语言基础

1.1 基础操作

1.1.1 简介及安装

就像任何别的学科一样，语言科学研究的进步必须依赖于合理的研究设计，科学的数据分析和清楚、透明的对结果的报道。为了在实验过程中去除“噪音”，第二语言加工研究经常需要设计大量的实验刺激材料，对不同的被试群体开展实验，由此获得大量的数据，从而对提出的研究假设进行科学的论证。但是，如何才能高效、科学地整理实验数据，再在此基础上构建简洁、准确的统计模型，并对模型进行解释呢？

R 语言给我们提供了简单直接的答案。掌握 R 语言将使我们很好地“武装”起来，从而能更加“气定神闲”地面对许多科学研究的问题。这不仅让我们具备极其重要的处理数据、分析数据的能力，更重要的是会让我们具备科学的思维和视角，去提出问题、分析问题和解决问题，或者去思考、验证和解决一些科学难题。那么到底什么是 R 语言？这几乎是所有的相关书籍一开始都会介绍的问题。总结起来，大都是这样说的：“R 是一种为统计计算和绘图而生的语言和环境，它是一套开源的数据分析解决方案，由一个庞大且活跃的全球化研究型社区维护”，“直到大数据的爆发，R 语言变成了炙手可热的数据分析的利器”。正是因为这个原因，有学者呼吁让 R 成为应用语言学研究者的学术通用语(Lingua Franca)(Mizumoto & Plonsky 2016)，这个呼吁跟最近二语习得研究领域有学者提出要尝试使用多元回归来代替方差分析的思想遥相呼应(见 Plonsky & Oswald 2017)，这一方面表明了语言研究者更加关注科学的发展，另一方面也说明二语研究在方法方面不断取得进步。

Mizumoto & Plonsky (2016)在阐述让 R 成为应用语言学研究的学术通用语时总结了 R 在四个方面的优势，包括：(1)数据分析的可重复性。研究者只需要分享他的代码就可以共享他整个数据分析的过程，极大地便利了研究的可重复性。同时，这也有利于不同的研究者，包括距离遥远的研究者开展科研合作。(2) 使用 R 是站在巨人肩膀上的工作。这是因为 R 有许多“聪明的人”开发了无数的可供直接利用的包(packages)，这些包让极其复杂的运算和统计分析变得简单。(3) 极强的数据可视化能力。在二语研究中，有时需要处理大量的反应时数据或者频数数据，数据可视化就显得非常重要。通过可视化一方面可以清楚地看出数据的分布规律，观察可能出现的异常值，离群点，等等；另一方面，还可以帮助解读数据，尤其是解读多个变量的交互效应，被认为“困扰着世界上许多聪明的大脑”的一件极其复杂的事情(Field et al. 2012)。(4) R 是一门编程语言，因此具有极强的灵活性(flexibility)和无限的才华(versatility)。

使用 R 与使用其他一些商业软件如 SPSS 最大的区别就在于我们需要使用键盘输入代码，因为 R 的特点就在于你必须告诉它干什么，它才能干什么。这似乎会让我们觉得这东西有点“傻”，这要看你怎么看，如果习惯了傻瓜式的一键操作，可能是会有这种感觉，但是反过来想，“告诉它干什么，它才能干什么”不正就是它的强大之处吗？R 的才华横溢之处正是来自于这里，对这一点，我相信后面大家会有更多的体会。但是无论如何，大家都得从输入开始，而且 R 还非常挑剔，输入时稍有错误，就无法运行。一开始大家不免觉得厌烦，甚至沮丧，甚至“绝望”。我给大家的建议就是一定要坚持下去，因为这东西怪就怪在你越使用它，你才会越喜欢它，而操控的感觉是一种真正的自由！这一点是很多 R 使用者共同的感受。

还是不要再说了，先把软件安装了吧。大家可以到 CRAN(Comprehensive R Archive

Newwork)下载 R，网址是：<https://cloud.r-project.org>。尽管 CRAN 由分布在世界各地的很多镜像服务器组成，用于分发 R 和 R 包。但建议不要选择离你近的服务器，而应该使用云镜像，因为它会自动找出离得近的服务器。打开网站后，根据自己的计算机系统选择要安装的 R。

安装好 R 后，建议大家都安装 Rstudio，它是用于 R 编程的一种集成开发环境，可以从这个网址下载安装：<http://www.rstudio.com/download>。Rstudio 安装好后会自动跟 R 关联起来，以后你只要打开 Rstudio，在那里执行所有的操作就行了。可以使用 Rstudio 菜单中的 tools 项下面的 global options，来根据自己的偏好设置 Rstudio 的页面网格等等。R 和 Rstudio 都会经常更新。按理应该经常保持更新，Rstudio 的更新比较简单，但是更新 R 却比较麻烦，因为我们在使用了 R 一段时间以后，已经在上面安装好了很多的包，如果重新下载安装新的 R 的话，之前安装好的所有包必须全部重新安装，这是一件很烦人的事情。尽管更新有麻烦，但是如果不经常更新就无法使用一些新的功能，大家只能权衡利弊了。

把 Rstudio 安装好后，打开 Rstudio，点击菜单中的 File，然后再点击 New File，选择 R Script 就会显示如下图所示的页面：

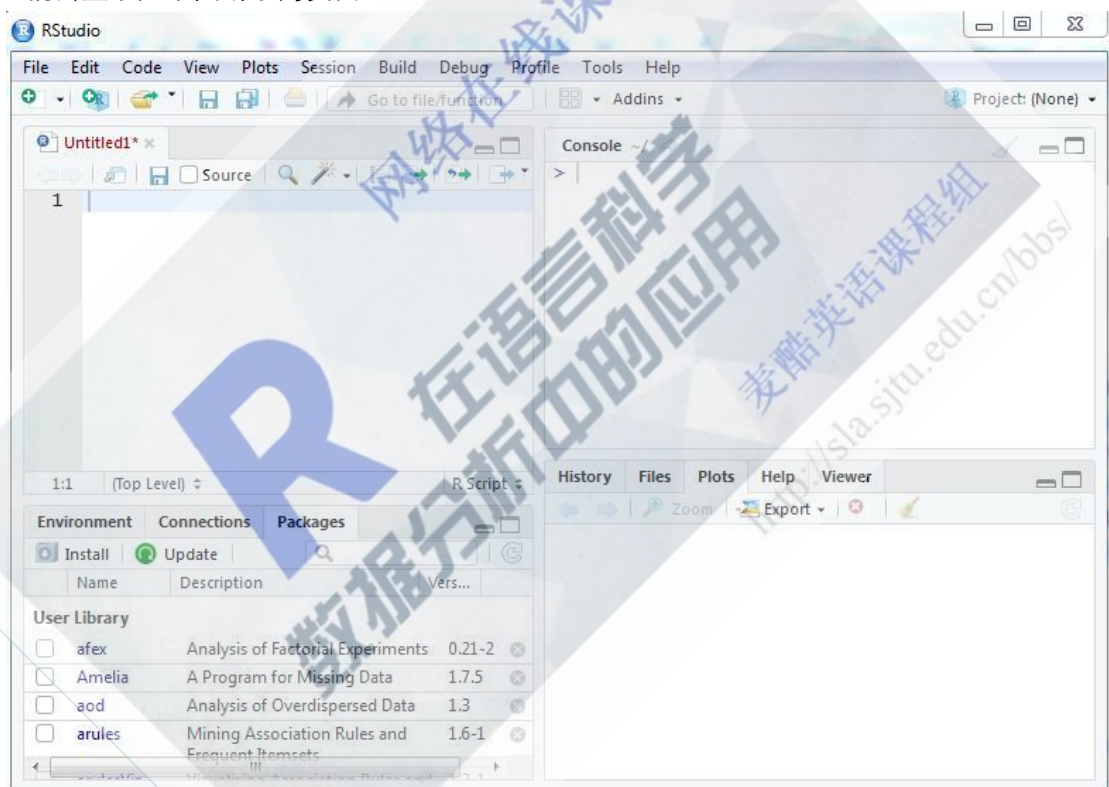


图 1-1 Rstudio 的工作区

当然，你的页面可能会有些不同，但都可以使用 Tools 菜单下面的 global options 自行设置。这里有三个关键区域：左上角，那里是代码编辑区，在那里输入代码，按“Ctrl + ENTER”键后就可以执行代码，执行的结果显示在右上角称作 console 的地方，即控制台。右下角可以显示生成的图片(plots)。

软件安装好了，可以开始了。那就再多啰嗦两句关于 R 的输入问题。我引用 Wickham &Grolemund (2017)的原话吧，这两个人可称得上是 R 语言的绝对大牛，他们的经验值得分享：

“当开始运行 R 代码时，你很可能会遇到问题，不用担心，每个人都会遇到问题。我已经写 R 代码多年了，但还是每天都会写出不能正常运行的代码。首先将你需要运行的代码与

书中的代码进行对比。R 极其挑剔，即使一个字母放错了位置，也可能会造成问题。确保每个括号都是完整的，不能只用一半如 “ (”，或者 “)”、确保双引号都有双边，不能只有一边。有时运行了代码却什么也没有发生，检查一下执行代码的控制台，如果有一个+号，那么说明 R 认为你没有输入完整，正在等待你完成输入，按 `Esp` 键中止当前执行的命令就可以重新开始。。。。。

关于输入，如果说我有什么建议的话，那就是大家可以利用零碎时间，记住 R 的一些快捷键，大家在后面会欣喜地发现可以使用快捷键是多么地省时省力。因为你在写代码的时候终于可以不必动用那“该死”的鼠标了，正在写代码的时候，突然要使用鼠标是很烦人的事情，“鼠标是写代码人的敌人”。大家网上随便搜搜就能找到完整的快捷键说明。比如，R 中输入最多的符号可能就是 `<-`，它的作用相当于 `=` 号，因为 `=` 不是 R 的标准语法，一般不会使用它。由于 `<-` 输入如此频繁，使用快捷键就会方便很多，即 `Alt + -`。另外，在代码的前面或者紧跟着代码，经常会看见 `#` 这个符号，它的意思是添加解释、注解或说明，目的是方便其他的人或者自己理解并记住代码的含义。注意的是，跟着这个符号后面的内容是不会执行的。

对了，软件安装好后，一般还会先安装好一些常用的包 (packages)。包是 R 变得强大和多才多艺的重要装备，它是 R 函数、数据、预编译代码以一种定义完善的格式组成的集合体 (见 Kabacoff 2015)。打开 Rstudio，点击左上角菜单中的 File，点击 New File，然后再点击 R Script，就会在 Rstudio 的左手上方打开 R 代码编辑区，在那里输入命令，按 “`Ctrl+Enter`” 执行命令，执行的结果显示在 Console，即 R 的控制台。第一次安装一个包，使用命令 `install.packages ()` 就可以了。比如，本书会反复使用的一个包叫做 tidyverse，先把这个包安装好：

```
install.packages ("tidyverse")
```

输入这个命令并执行后，只要你的电脑联网了就会自动安装这个包。在包安装好后，如果要使用这个包就必须加载：

```
library (tidyverse)
```

每次使用都要加载，注意是加载，不是安装。只有在加载后，才可以使用这个包提供的功能。后面我们会陆续讲到要安装和加载的包。下表是 R 语言使用的逻辑运算符，对初学者来说很有用。

表 1 R 语言使用的逻辑运算符(见 Kabacoff 2015: 68)

运 算 符	描 述
<code><</code>	小于
<code><=</code>	小于或等于
<code>></code>	大于
<code>>=</code>	大于或等于
<code>==</code>	严格等于*
<code>!=</code>	不等于
<code>!x</code>	非 <code>x</code>
<code>x y</code>	<code>x</code> 或 <code>y</code>
<code>x & y</code>	<code>x</code> 和 <code>y</code>
<code>isTRUE(x)</code>	测试 <code>x</code> 是否为TRUE

1.1.2 设置工作目录

工作目录(working directory)是 R 用来读取文件和保存结果的默认目录。使用 `getwd()` 函数可以查看当前的工作目录,而要设定当前的工作目录,则需要使用 `setwd()` 函数来完成。设定工作目录是使用 R 的一个好习惯。一方面,设定了工作目录后,如果要加载工作目录下的文件,比如数据文件,只需要输入数据的文件名就行了,不需要输入完整的路径,如果没有设定工作目录,当需要读入一个不在当前工作目录下的文件时,就要输入完整的路径;另一方面,一旦设定工作目录后,跟当前项目所有相关的文件可以默认保存在这个工作目录里面,下一次还可以从上一次会话结束的地方重新开始,不会跟别的项目之间的数据或者设置相互干扰。下面的命令是使用 `setwd()` 函数创建工作目录的一个实例:

```
setwd("E: / bookR")
```

这个命令把当前的工作目录设置为 E 盘的 bookR 文件夹,当然,要设置成功的话必须确保 E 盘里有一个名为 bookR 的文件夹。一旦设置成功后,就可以在这个文件夹里保存当前项目的所有内容。

1.1.3 数据导入和保存

1.1.3.1 R 的数据结构

还是得介绍一下 R 的数据结构,它包括向量、矩阵、数组和数据框。使用最多的是向量(vector)和数据框(data frame)。

向量是一维数组,可以贮存数值型、字符型或逻辑型数据,使用函数 `c()` 可创建向量。比如:

```
X <- c(365, 359, 336, 469, 343, 454)
Y <- c("form", "semantic", "related", "unrelated")
Z <- c(FALSE, FALSE, TRUE, TRUE, FALSE, TRUE)
```

上面创建的向量中,X 是数值型, Y 是字符型,而 Z 是逻辑型向量。需要注意的是,在输入字符型向量的时候,必须加双引号来标识,此外,一个向量只能有一种类型,不能多种混合在一块。向量在 R 中被经常使用,试想一下,如果我们要输入向量 X 中的 6 个数据的话,必须要一个一个输入,但是如果创建了向量,就可以只输入 X 来输入这 6 个数据。比如运行下面的代码:

```
X <- c(365, 359, 336, 469, 343, 454)
X
[1] 365 359 336 469 343 454
```

再比如,我们要安装一系列 R 的包,一个一个安装就会比较麻烦,如果使用向量就会简单很多,如:

```
install.packages(c("MASS", "ez", "multcomp", "xlsx"))
```

数据框是 R 语言最为常见的数据结构,它的结构与大家熟悉的 Excel 表格、SPSS 和 SAS 的数据集类似。数据框可以视作由向量构成,是通过 `data.frame()` 函数把一个一个向量组合起来创建而成的,如下:

```

LISTS <- c ("List1", "List2", "List3", "List4","List5")
SUBJ <- c ("WSY", "XZ", "GHJ", "YXH", "WYX")
CONDITIONS <- c ("frmsimilar", "frmcontrl", "semanticsimilar", "semanticcontrl","frmcontrl")
RTs <- c (365, 359, 336, 469, 343)
myData <- data.frame (LISTS, SUBJ,CONDITIONS,RTs)
myData

```

	LISTS	SUBJ	CONDITIONS	RTs
1	List1	WSY	frmsimilar	365
2	List2	XZ	frmcontrl	359
3	List3	GHJ	semanticsimilar	336
4	List4	YXH	semanticcontrl	469
5	List5	WYX	frmcontrl	343

可以看到，一个的向量组合成了这个名为 myData 的数据框，它融合了数值型以及字符串型的等向量。跟大家所熟悉的 Excel 等生成的数据表类似，数据框里的列(column)称作变量(variable)，一列一列就是一个一个变量，比如上面的 myData 这个数据框一共有四个变量，分别为 LISTS, SUBJ, CONDITIONS, RTs。而行(row)就是一个一个具体的观测值(observations)。理解 R 语言中数据框里列和行代表什么意思非常重要，是数据分析的最基础的知识。

1.1.3.2 R 的数据

我们当然可以按照上面创建 myData 数据框的方法，通过键盘把要分析的数据一个一个输入到 R，但是，一般不会这么做，效率太低了，直接通过键盘来输入数据也不是 R 的强项。常规的做法是使用别的数据处理软件如 Excel, SPSS, SAS 等整理好的数据直接导入到 R，再进行分析和处理。理论上 R 可以导入几乎所有种类的数据，包括从网站上提取的数据。但是对我们一般的研究人员来说，更多的时候是自己先把数据准备好，再导入到 R。因此，有必要简单介绍一下如何准备数据，以便更加高效方便地导入到 R 进行分析。

在 R 中使用最多的数据文件可能就是 csv 文件了(Comma-Separated Values, 逗号分隔值)，它的读取和操作都比较简单。一般会使用 excel 先编辑好数据，再另存为 csv 文件。比如下面的数据就是一个已经编辑好的 csv 数据文件：

表 1 用于 R 进行分析的数据表

SUBJ	COND	TYPE	TARGET	PRIME	RT	FREQ
86	TN	word	EACH	eahc	766	350781
86	CONTRL	word	THING	thnig	822.6	318999
86	SN	word	WHITE	whiet	827.2	256059
96	TN	word	NIGHT	ngiht	752	238723
96	CONTRL	word	PLAY	lpay	605.6	157416
96	SN	word	NORTH	nroth	1093.714	123217
91	TN	word	HEART	herat	997.5	107561
91	CONTRL	word	DARK	drak	906.5556	84951
91	SN	word	FAST	afst	915.6	51454
84	TN	word	CRIME	rcime	1345.625	49923
84	CONTRL	word	SHAPE	shaep	826.6	38066
84	SN	word	BAND	badn	934.875	37172
80	TN	word	ANGRY	angyr	828.4	29186
80	CONTRL	word	NOSE	noes	591.1	27939
80	SN	word	HORSE	ohrse	1093.571	27312
90	TN	word	HATE	haet	1037.222	26189
90	contrl	word	SALE	slae	584.6667	22803
90	SN	word	FRUIT	rfuit	900.375	22333

这个文件体现了 R 数据文件的几个原则：

- (1) 第一行全部是变量名称。比如上面的数据一共有 7 个变量，我的习惯是变量名都用大写，并且用简洁又能大致表达意思的名称最好。比如 SUBJ 表示的意思是被试(subject)，即这一列对应的都是被试标识，COND 表示实验条件(condition)，RT 表示反应时(reaction time)，等等。采用简洁好记的名称命名变量非常重要，因为数据分析就是对变量的操作，简洁好记的名称将使输入变得容易。另外，一般不要使用数字作为变量名。
- (2) 从第二行开始，每一行对应的都是一个观测数据，观察数据既有数值型的，也有字符串表示的分类数据。一般来说，如果不是有特别原因，字符串数据一般用小写。上面数据中 86 号被试对应的 TN 和 EACH 是大写的，那是因为它们在这个实验数据里有特别的意思，而 word 就是小写的。在 R 里面，不要用数字去标识分类变量，比如性别就用 male, female 来表示，不要用 1 去表示 male，2 表示 female。这个原则很重要，大家在后面会认识到。如果有缺失值就用 NA 标记，不要留空。
- (3) 好的数据框的标准是简洁、清楚，容易看懂。比如，变量名要短而有意义，不要有空格，不要有逗号或者点号。

数据一般有两种格式，一种称作为长数据(long format)，还有一种称作为宽数据(wide format)。长数据就是指同一名被试的多次测试结果保留在同一列里；宽数据则相反，同一名被试的多次测试结果分别保留在不同的列里。比如，仔细观察上表 1 就会注意到它是长数据，86 号被试重复出现了 3 次，它在 COND 三个条件(TN, SN, CONTRL)下的测试数据保留在同一列里即 RT 列。而下表 2 就不同，86 号被试在三个条件下的数据(TN, SN, CONTRL)被分别保存在不同的列里，它是一个宽数据表：

表 2 导入 R 的数据表

NUMBER	TN	CONTRL	SN	TYPE	TARGET	PRIME
86	1608.75	1182.5	1137.333	word	SHAPE	skuch
96	1808.8	1673.286	1536.5	nonword	BREAP	brepa
91	1331.571	1625.5	781	nonword	COWCE	cowec
84	876.8571	2448	1408	word	SALE	slae
80	1251	709	1472	word	SHAPE	shaep
90	1049.125	1001.571	2285	word	SLIDE	sldie
61	1000	606	869	word	THING	thnig
62	763.9	934.875	1172.667	word	TWIN	wtin

理解这两种数据的不同非常重要。一般来说，R 使用比较多的是长数据，某些时候根据需要也会转变成宽数据。能快速高效的实现这两种数据之间的转换是数据处理能力的表现。我们在 1.2.2.5 小节进行了详细介绍。有 SPSS 使用经验的读者可能会注意到，在 SPSS 数据分析中，宽数据使用比较多，尤其是重复测量的数据都是宽数据。

1.1.3.3 数据的导入和保存

要使用 R 进行数据分析就必须将数据导入 R，也就是读取保存在文本文件、数据库或网站上的数据。Wickham & Grolemund (2017) 把数据科学用一种简图表示：



图 1-2 数据科学的过程

如图 1-2 所示，导入数据是数据科学的前提。如果不能将数据导入到 R，分析处理数据也就无从谈起。这里介绍两种数据导入方法，一种是传统的数据导入，另一种是使用 readr 包导入数据。

1.1.3.3.1 传统的数据导入

(1) csv 格式数据的导入。 csv 格式的数据也是二语加工研究中最常见的数据格式，不管是在准备这种格式的数据，还是把它导入到 R，都非常便捷、高效。输入：

```
myData <- read.csv (file.choose ( ), header = TRUE)
```

使用这个方法导入数据是在你没有设置工作目录的情况下的做法，输入上面的命令按回车键就会跳出一个文件选择框，然后找到自己存放的数据文件再点击导入就行了。另外一个方法就是在 read.csv () 函数里输入读取文件的路径，这里的 header=TRUE 是告诉 R 导入的数据文件的第一行是变量名，由于它是默认设置，也可以省略：

```
myData <- read.csv ("E:/bookR/MAdata.csv", header = TRUE)
colnames (myData)
[1] "LIST"          "SUBJ"          "TRIAL"          "CHINITEMS"     "ACC"
[6] "RT"            "COND"          "ENGITEMS"       "ORDER"          "TYPE"
[11] "RELATEDNESS"
```

(2) Excel 生成的 xlsx 格式数据的导入。这种格式的数据导入更为复杂些，可以借助 xlsx 包来读取。因此第一步是下载并安装这个包，然后调用 read.xlsx() 函数来导入工作表。安装代码如下：

```
install.packages ("xlsx")
```

如果提示安装不成功，很可能是需要升级你电脑的 Java 程序。可以到相关网站下载最新的 Java 程序，把它安装好后，就可以重新安装 xlsx 这个包。在安装成功后，就可以使用 read.xlsx (file, n) 函数来导入数据表格，如以下代码所示：

```
library (xlsx)
workbook <- "E:/camb/Exp4.xlsx"
myData <- read.xlsx (workbook, 1)
```

可以看到，首先要使用 library() 函数加载 xlsx 这个包。这里要注意的是，第二行要输入的是你的数据文件存放的具体路径。比如，在这里数据文件是存放在 E 盘名为 camb 文件夹之下，数据文件名为 Exp4.xlsx。由于一个 Excel 文件可能用多个工作表，因此 read.xlsx (file, n) 中的 n 就是指明具体要导入哪个工作表。导入 xlsx 数据的一个缺点是速度可能比较慢，需要等一段时间。为了确认数据是否成功导入，可以使用 colnames() 查看：

```
colnames (myData)          #显示数据的列的名字，即变量名。
```

如果数据已经成功导入，执行这个命令后，就将呈现数据的列的名称，即数据框的变量名。当然，如果使用 Rstudio 的话，也可以在 Environment 框里看到关于数据的一些信息，比如一共有多少个变量(variables)，有多少个观测值(observations)。

(3) SPSS 数据集的导入。尽管我本人从来不会把数据先输入到 SPSS 后再把它导入到 R。可以使用 Hmisc 包中的 spss.get() 函数来把数据文件导入。因此，先安装 Hmisc 包：

```
install.packages ("Hmisc")
然后，使用以下代码导入数据：
```

```
library (Hmisc)
myData <- spss.get (choose.files (), use.value.labels = TRUE)
```


输入以上代码以后，电脑就会自动跳出一个文件选择框，通过它找到自己的 SPSS 数据文件贮存的位置，选定该文件后，数据就会导入到 R，并被指定为 myData。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。

也可以将在 R 中运行的数据框导出到电脑上贮存，方法是使用 write.table () 函数：

```
write.table (myData, file = "e: camb / RTs.txt") #Windows
```

```
write.table (myData, file = "/ camb / RTs.txt") #MacOSX
```

请注意要正确输入文件保存的路径，包括文件夹以及保存的文件名。上面两个命令都是把 R 中的 myData 导出到相应盘里的 camb 文件夹下面，并命名为 RTs.txt，即作为文本文件导出。

1.1.3.3.2 使用 readr 包导入数据

除上面传统的直接导入数据框的方法外，还有一种导入数据的方法，就是使用 readr 包，这种方法方便、快捷，也是现在最普遍采用的方法。详情见本章 1.2.2.3 小节。

1.2 数据管理

导入数据后，就应该对数据进行整理，确保数据整洁、好用。整洁的数据非常重要，它可以让我们把工作重点放在数据分析上，不用再花费心思去把数据转换成各种形式，或者做各种枝枝叶叶的处理，浪费很多的时间。简洁的数据的要求就像前面说的，确保每列都是一个变量(variable)，每行都是一个观测值(observation)。

下面分两个方面介绍数据的管理和操作。一是传统的数据框(data frame)的管理操作；一个是 tibble 的管理和操作。

1.2.1 数据框的数据管理

1.2.1.1 一些常规操作

我们曾经通过翻译判断任务(Translation Recognition Task)调查中国英语学习者词汇及概念表征发展。在这个任务里，被试先看到一个英语单词，在屏幕上停留 500 毫秒后消失，接着屏幕上呈现一个汉语单词，要求既快又准确地判断这个汉语单词是否是前面英语单词的正确翻译，记录被试的反应时(RT)和准确率(ACC)。我们先导入这个命名为 wordConcept 的 csv 格式的数据框：

```
wordConcept <- read.csv (choose.files ( ), header = TRUE)
```

或者

```
wordConcept <- read.csv ("E:/bookR/MAdat.csv", header = TRUE)
```

可以使用 str () 函数查看所导入的数据的一般信息：

```
str(wordConcept)
'data.frame': 2000 obs. of 11 variables:
 $ LIST      : Factor w/ 5 levels "List1","List2",...: 1 1 1 1 1 1 1 1 1 ...
 $ SUBJ      : int  2 2 2 2 2 2 2 2 2 ...
 $ TRIAL     : int  72 78 27 22 75 58 36 41 16 34 ...
 $ CHINITEMS : Factor w/ 253 levels "10deng.PNG","10la.PNG",...: 1 54 111 165 219 229 238
 239 245 253 ...
 $ ACC       : int  1 1 1 1 1 1 1 1 1 ...
 $ RT        : int  940 658 532 938 767 2286 692 682 885 818 ...
 $ COND      : Factor w/ 5 levels "correct","formcontrol",...: 1 1 1 1 1 1 1 1 1 ...
 $ ENGITEMS  : Factor w/ 51 levels "10light.PNG",...: 1 12 23 34 45 47 48 49 50 51 ...
 $ ORDER     : int  10 1 2 3 4 5 6 7 8 9 ...
 $ TYPE      : Factor w/ 3 levels "", "form", "semantic": 1 1 1 1 1 1 1 1 1 ...
 $ RELATEDNESS: Factor w/ 3 levels "", "related", "unrelated": 1 1 1 1 1 1 1 1 1 ...
```

str()函数显示了这个导入数据框的许多信息。第一行是总览，显示导入的是一个 data.frame (数据框)，总共有 11 个变量，2000 个观测数据，即相当于有 2000 行数据。\$ LISTs 是第一个变量，它是一个因子(factor)，共有 5 个水平，但只以缩略的形式显示前两个水平即 List1, List2。\$ SUBJ 是第二个变量，它是整数(int 即 integer，整数)，但只显示了被试号为 2 的被试。\$ TRIAL 是第三个变量，它也是整数，在实验中它表示所呈现的翻译材料出现的顺序。

为了检测数据是否成功，最常使用的是 colnames()函数：

```
colnames(wordConcept)
```

```
[1] "LIST"      "SUBJ"      "TRIAL"     "CHINITEMS" "ACC"
[6] "RT"        "COND"      "ENGITEMS"  "ORDER"     "TYPE"
[11] "RELATEDNESS"
```

获得的结果是导入数据的列(column)的名称，也就是数据框的各个变量，一共有 11 个变量。colnames()函数，既可以立即查看数据是否导入成功，也可以查看数据中各变量的名称。在变量很多的时候，这非常有用，因为我们在做数据分析的时候有时很难记住数据框里有哪些变量，以及它们分别是如何拼写的，但在分析或者运算时，却时时都可能用到数据的变量名。使用 head()函数，可以默认查看数据框的前 6 行数据：

```
head(wordConcept)
  LIST SUBJ TRIAL CHINITEMS ACC RT COND ENGITEMS ORDER TYPE RELATEDNESS
1 List1  2   72 10deng.PNG  1 940 correct 10light.PNG  10 correct correct
2 List1  2   78 1pingguo.PNG 1 658 correct 1apple.PNG  1 correct correct
3 List1  2   27 2mao.PNG  1 532 correct 2cat.PNG  2 correct correct
4 List1  2   22 3shizhong.PNG 1 938 correct 3clock.PNG  3 correct correct
5 List1  2   75 4gongdian.PNG 1 767 correct 4palace.PNG 4 correct correct
6 List1  2   58 5gangqin.PNG 1 2286 correct 5piano.PNG  5 correct correct
```

如果要查看最后 6 行数据，可以使用 tail(wordConcept)。还可以通过使用 n=x 来限定查看的行数。比如 head(wordConcept, n=10)则限定查看前 10 行，tail(wordConcept, n=10)则查看最后 10 行。如果使用 nrow()函数，则可以查看数据框一共有多少行数据：

```
nrow(wordConcept)
```

```
[1] 2000
```

结果显示上面导入的数据一共有 2000 行，即 2000 个观测值。在第二语言加工研究中，这个结果会经常用到，比如我们可以比较删除异常值后的行数和没有删除时的行数，从而算出删除异常值一共影响到了多少数据，在论文中这是必须如实报道的值。如果要选定数据框的具体某个数据，可以通过指定行和列的方法来实现，比如，输入：

```
wordConcept [6,6]
[1] 2286
```

在这个表达式里，中括号[]里逗号前表示的是行，逗号后表示的是列。因此，这里指定的是第 6 行第 6 列的数据。也可以通过变换逗号前后的数字，从而指定任何行或者列的数据。如果逗号前的数据为空，逗号后的数字是 6，则显示第 6 列的所有行的数据：

```
wordConcept [,6]
[1] 365 359 336 469 343 454 388 669 490 1257 777 658
[13] 829 981 709      684 674 748 757 772 885 938 532 818
...
```

同样，如果在逗号前指定行，逗号后为空，则表示指定行的所有列的数据。比如，输入：

```
wordConcept [6,]
LISTS SUBJECT TRIAL ITEMS ACC RT CONDITION TYPE RELATEDNESS
6 List1 100 139 4gongdian.PNG 1 454 correct correct correct
```

也可以使用数据框的名称加\$符号来指定具体的列，比如：

```
wordConcept $ RT #它的作用等同于 wordConcept[,6]
```

使用上述方法即“数据框+\$+列”的方法来指定具体的列，在实际的数据操作和运算中会经常使用到，是 R 语言的常规方法。如果一下要指定多个行或者列，可以结合向量操作函数 c()，比如，如果要指定选择第 5 行，第 10 行，第 15 行和第 20 行的数据，可以如下操作：

```
wordConcept [c(5,10,15,20),]
LISTS SUBJECT TRIAL ITEMS ACC RT CONDITION TYPE RELATEDNESS
5 List1 100 134 6zhu.PNG 1 343 correct correct correct
10 List1 100 157 9xihongshi.PNG 1 1257 correct correct correct
15 List1 106 74 5gangqin.PNG 1 709 correct correct correct
20 List1 106 108 9xihongshi.PNG 1 772 correct correct correct
```

如果要指定具体的列，方法类似。比如要指定第一、第二、第五、第六、第七列，则可以：

```
wordConcept [,c(1,2,5,6,7)]
LISTS SUBJECT ACC RT CONDITION
1 List1 100 1 365 correct
2 List1 100 1 359 correct
3 List1 100 1 336 correct
4 List1 100 1 469 correct
5 List1 100 1 343 correct
6 List1 100 1 454 correct
.
.
.
```


也可以通过直接指定列的名称来实现，如：

```
wordConcept [ ,c ("LIST", "SUBJ", "ACC", "RT", "COND")]
```

	LIST	SUBJ	ACC	RT	COND
1	List1	2	1	940	correct
2	List1	2	1	658	correct
3	List1	2	1	532	correct
4	List1	2	1	938	correct
5	List1	2	1	767	correct
6	List1	2	1	2286	correct
7	List1	2	1	692	correct
8	List1	2	1	682	correct
9	List1	2	1	885	correct
10	List1	2	1	818	correct

·
·
·

在数据分析中，更常使用的是按条件选定要操作的数据，比如对实验中反应时(RT)数据，一般只选择做出正确反应条件的数据，也就是说 ACC 对应为 1 的数据，操作方法如下：

```
wordConcept[wordConcept$ACC==1,]
```

	LISTS	SUBJECT	TRIAL	ITEMS	ACC	RT	CONDITION
1	List1	100	101	5gangqin.PNG	1	365	correct
2	List1	100	105	1pingguo.PNG	1	359	correct
3	List1	100	122	2mao.PNG	1	336	correct
4	List1	100	131	10qiang.PNG	1	469	correct
5	List1	100	134	6zhu.PNG	1	343	correct
6	List1	100	139	4gongdian.PNG	1	454	correct

或者只想选定 CONDITION 为 formsimilar 的数据，操作方法如下：

```
wordConcept[wordConcept$COND=="formsimilar",]
```

	LIST	SUBJ	TRIAL	CHINITEMS	ACC	RT	COND	ENGITEMS	ORDER	TYPE	RELATEDNESS
801	List1	2	28	11kanjian.PNG	1	1013	formsimilar	11sea.PNG	11	form	related
802	List1	2	83	12taiyang.PNG	1	608	formsimilar	12son.PNG	12	form	related
803	List1	2	74	13huai.PNG	1	525	formsimilar	13bed.PNG	13	form	related
804	List1	2	85	14nanhai.PNG	1	537	formsimilar	14toy.PNG	14	form	related
805	List1	2	87	15sange.PNG	1	573	formsimilar	15tree.PNG	15	form	related
806	List1	2	57	16ting.PNG	1	908	formsimilar	16heart.PNG	16	form	related
807	List1	2	60	17sharyang.PNG	1	1111	formsimilar	17coat.PNG	17	form	related
808	List1	2	70	18naozzi.PNG	1	571	formsimilar	18cup.PNG	18	form	related
809	List1	2	54	19yuruo.PNG	1	1157	formsimilar	19week.PNG	19	form	related
810	List1	2	53	20zhiwu.PNG	1	1064	formsimilar	20plane.PNG	20	form	related

需要引起注意的是，在 R 语言中等号是 “==” 而不是我们所熟悉的 “=”，如果写错的话 R 将报错。同时，如果列的名称是字符串(因子)，需要使用双引号括起来，否则 R 也会报错。上述两种按条件选定数据的方法，可以使用 subset () 函数来实现，请见 1.2.1.3.3 (subset () 函数)。

1.2.1.2 对数据框信息进行修改

在实际操作中，可能经常需要对数据做一些改动，比如修改变量的名称，或者对数据进行转换等等。如果涉及较大规模的改动，最聪明的办法当然是先直接在数据文件里如 excel 或者 csv 文件表格里修改，然后再把数据导入到 R。直接在 R 进行大规模数据修改并不是很好的办法。一般 R 能修改是涉及到少量数据的或者整列或者整行的修改。比如，要改变某一列的名称，把 wordConcept 中 ITEMS 改为 MATERIALS, 可以使用 fix () 函数：

```
fix (wordConcept)
```

执行命令后就会调用一个交互式的编辑器，单击变量名，然后在弹出的对话框中将其重命名。需要注意的是，我们在 R 进行的所有修改，只会改变导入 R 的当前的数据，并不会改变电脑里的源数据，如果要把当前的修改保存起来，则需要把数据导出，方法请见前文。修改变量也可以使用 tidyverse 包当中的 rename () 函数：

```
library (tidyverse)
wordConcept <- rename (wordConcept, RTs = RT)
```

先用 library 加载 tidyverse 包，再通过 rename 这个命令把 wordConcept 原来的 RT 变量修改为 RTs。使用哪个方法修改，取决于个人的习惯。

如果要增加一个列到现有的数据框 wordConcept，即在现有的数据框里创建一个新的变量，经如 SUMRT，让它等于 wordConcept 这个数据框中 TRIAL 和 RT 的和，实现方法如下：

```
wordConcept $ SUMRT <- wordConcept $ TRIAL + wordConcept $ RT
```

执行这个命令后，使用 colnames (wordConcept) 查看所有的列就会发现多了一个名为 SUMRT 的一列，它是 TRIAL 和 RT 数值之和：

```
colnames(wordConcept)
[1] "LIST"      "SUBJ"      "TRIAL"      "ITEMS"      "ACC"
[5] "RT"        "CONDITION" "TYPE"       "RELATEDNESS" "SUMRT"
```

也可以使用 transform () 函数来实现，方法如下：

```
wordConcept <- transform (wordConcept, SUMRT = TRIAL + RT)
```

使用 transform () 函数的好处就是，可以同时增加多个列，比如，如果还需要同时增加 MEANSUM 来作为 TRIAL 和 RT 和的平均数，则可以按如下方法实现：

```
wordConcept <- transform(wordConcept, SUMRT=TRIAL+RT, MEANSUM=(TRIAL+RT)/2)
colnames(wordConcept)
[1] "LISTS"      "SUBJECT"    "TRIAL"      "ITEMS"      "ACC"
[6] "RT"         "CONDITION"  "TYPE"       "RELATEDNESS" "SUMRT"
[11] "MEANSUM"
```

可见使用 transform () 函数是一个很方便的增加列即增加变量的方法。除此以外，当前使用最多的是 tidyverse 包中的 mutate () 函数，使用它来增加变量或者对变量之间进行各种运算，比如：

```
wordConcept <- mutate (wordConcept, RTs = RTs / 1000,
                        SUMRT = TRIAL + RTs)
```

使用 `mutate()` 函数的优雅之处，就是可以同时增加多个变量并进行变量之间的运算、操作。在这一点上，`mutate()` 与 `transform()` 有相似之处，但是现在比较流行的是 `mutate()` 函数，它是 `tidyverse` 包其中的一个非常重要的函数，在数据分析中广泛使用，`mutate()` 函数甚至可在同一个函数里对刚创建的变量进行各种操作和运算。

1.2.1.3 数据操作的常用函数

1.2.1.3.1 因子水平

总体上变量 (variables) 可以分为三大类：(1) 定类变量或称名义型变量 (nominal variables)。比如把词类分作动词、名词、形容词等，把性别分作男和女，等等，那么词类以及性别就属于名义型变量；(2) 定序变量或称有序变量 (ordinal variables)。比如，把学生的语言水平分成高、中、低，把高铁的座位分成一等、二等，那么语言水平以及高铁的座位就可称为定序变量。(3) 定比变量或称数值型变量 (ratio variables)。定序变量与数值型变量的区别在于定序变量如高、中、低等可以排序，但是不能加减，但是数值型变量可以加减。一般把名义型变量和定序变量统称为分类变量 (categorical variables)，这样的话，变量就分成了两类，即分类变量和数值型变量。

当把数据导入 R 的时候，传统的数据框会自动把一个分类变量和有序变量转变成因子 (factor)。因子在 R 中非常重要，当我们在分析数据或者视觉化呈现数据的时候，都需要灵活地使用到因子。“因子的水平”这个概念在统计分析和作图时经常要用到，一个因子常常会有多个水平，就像我们在实验设计时设计的自变量会有多个水平一样。比如上面介绍的 `wordConcept` 这个数据中的 `CONDITION` 就是一个因子，它总共有 5 个水平：`correct`, `formcontrol`, `formsimilar`, `semanticcontrol`, `semanticrelated`

可以使用 `levels()` 函数来查看一个因子的水平，比如：

```
levels(wordConcept$CONDITION)
[1] "correct"    "formcontrol" "formsimilar" "semanticcontrol"
[5] "semanticrelated"
```

因子中各个水平的顺序也非常重要，R 总是默认按英语的字母顺序来安排因子的水平，字母排在最前面就是因子的第一个水平。但是，我们可以根据需要进行修改一个因子水平的顺序，修改的方法是使用 `factor()` 函数，比如要修改 `CONDITION` 因子的水平，可以：

```
wordConcept$CONDITION <- factor(wordConcept$CONDITION,
  levels = c("formcontrol", "formsimilar", "correct", "semanticcontrol", "semanticrelated"))
```

如果只是想修改因子的默认水平，还可以使用 `relevel()` 函数，比如：

```
wordConcept$CONDITION <- relevel(wordConcept$CONDITION, ref = "formcontrol")
```

关于因子的这些知识在后面还会反复使用到，可能也只有真正到了统计分析或者绘图的时候，大家才可能真正理解何为因子以及因子水平是什么。

1.2.1.3.2 cbind()和 rbind()函数的使用

这两个函数都可对数据进行合并。其中cbind()函数用于横向(column)合并两个数据框，要求是每个数据框要有相同的行数，要特别注意的是两个数据框要以相同的顺序排序。格式如下：

```
Mergedata <- cbind (dataframeA, dataframeB)
```

举例如下：

首先创建数据框dataframeA：

```
LISTS <- c("List1", "List2", "List3", "List4", "List5")
SUBJECT <- c(100, 101, 102, 103, 104)
CONDITIONS <- c("frmsimilar", "frmcontrl", "frmsimilar", "frmcontrl", "frmsimilar")
RTs <- c(365, 359, 336, 469, 343)
dataframeA <- data.frame(LISTS, SUBJECT, CONDITIONS, RTs)
```

dataframeA

	LISTS	SUBJECT	CONDITIONS	RTs
1	List1	100	frmsimilar	365
2	List2	101	frmcontrl	359
3	List3	102	frmsimilar	336
4	List4	103	frmcontrl	469
5	List5	104	frmsimilar	343

再创建数据框 dataframeB:

```
RELATEDNESS <- c("related", "unrelated", "related", "related", "related")
TYPE <- c("form", "form", "semantic", "form", "semantic")
dataframeB <- data.frame (RELATEDNESS, TYPE)
```

dataframeB

	RELATEDNESS	TYPE
1	related	form
2	unrelated	form
3	related	semantic
4	related	form
5	related	semantic

再使用 cbind ()对这两个数据框进行合并：

```
Mergedata <- cbind(dataframeA, dataframeB)
```

Mergedata

	LISTS	SUBJECT	CONDITIONS	RTs	RELATEDNESS	TYPE
1 List1	100	frmsimilar	365		related	form
2 List2	101	frmcontrl	359		unrelated	form
3 List3	102	frmsimilar	336		related	semantic
4 List4	103	frmcontrl	469		related	form
5 List5	104	frmsimilar	343		related	semantic

可见, 通过 `cbind()` 函数对两个数据框进行合并, 相当于扩充了数据框的列, 即增加了变量, 因此顺序特别重要, 如果顺序不一致, 即使能合并成功, 但是使用合并后的数据框进行分析时就可能出现错误。

`rbind()` 函数是对数据进行纵向(row)合并, 相当于增加了数据框的行。因此, 两个数据框必须拥有相同的列, 即相同的变量, 但是它们的顺序可以不一样。读者可以自己尝试创建两个数据框, 进行纵向合并试验。相对来说, 使用 `rbind()` 函数来合并数据在研究中使用较多。比如, 在实验中分别收集了高、中、低水平三组不同的数据, 并分别建立了高、中、低水平三组不同的数据框, 现在要把它们合并在一个数据框, 以进行整体分析, 就可以使用 `rbind()` 进行合并。

1.2.1.3.3 subset() 函数

这是使用最广泛的函数之一, 尤其是在没有 `tidyverse` 包之前。它的作用是对数据按自己的目的取子集。比如, 对 `wordConcept` 这个数据框, 只选择被试反应正确的数据, 即 `ACC` 为 1 的数据:

```
myData <- subset(wordConcept, ACC == 1)
```

这样, `myData` 这个数据框就变成了被试所有反应正确的数据集了。通过下面的公式, 还可以算出一共删除了多少反应错误的数据:

```
(nrow(wordConcept) - nrow(myData))/nrow(wordConcept)
[1] 0.072
```

可见一共删除了 7.2% 的数据。还可以使用多种条件并列的方法来选择数据, 比如, 既要选择被试反应正确的数据, 同时又要选择被试的反应时(RT)小于 2000 大于 200 的数据:

```
myData <- subset(wordConcept, ACC == 1 & RT > 200 & RT < 2000)
```

进行此操作后, 新的数据框(`myData`)一共有 1844 行, 原来是 2000 行, 删除了 156 行数据。在使用多种条件对数据进行选择时, 要特别注意是使用 `&` (和), 还是使用 `|` (或)。比如, 既要选择被试反应正确的数据, 同时又要选择被试的反应时(RT)小于 500 或者大于 2000 的数据, 就要改成如下命令:

```
myData <- subset (wordConcept, ACC == 1 & RT<500 | RT >2000)
nrow (myData)
[1] 352
```

如果要去除某个条件下的数据，可以使用 != (不等于)这个符号：

```
myData <- subset (wordConcept, CONDITIONS != "correct" )
```

1.2.1.3.4 attach ()、detach ()、with ()函数

在前面大家已经注意到，每当我们使用数据框的子集的时候，都必须加上这个数据框的名字，再加上\$符号，比如 wordConcept\$RT, wordConcept\$COND,这有点麻烦,解决的办法就是在使用之前用 attach ()函数把数据框“加载”在当前的操作里，这样就可以直接引用它任何的子集了：

```
rm (list = ls (all = TRUE) )
wordConcept <- read.csv ("E: / bookR / MAdata.csv", header = TRUE)
attach (wordConcept)
mean (RT)
[1] 733.874
levels (COND)
[1] "correct"          "formcontrol"       "formsimilar"
[4] "semanticcontrol" "semanticrelated"
detach (wordConcept)
```

在第一行里使用 rm ()函数清理了一下当前的内存，因为前面进行的操作太多了，有点乱。再重新导入数据，一切重头开始。使用 attach ()函数把数据框“加载”进来，后面使用 mean ()函数计算平均值，levels ()查看分类变量有多少个水平。要是没有先把数据框加载进来的话，就必须跟前面一样操作，即 mean (wordConcept\$RT)、levels (wordConcept\$COND)。一旦不再使用当前数据框时，一个很好的习惯是使用 detach ()解除数据加载，这样就不会影响后续的操作。如果使用 with ()函数的话，则无须再使用 detach ()，with ()函数说明数据框只在 with ()函数的括号内发挥作用：

```
with (wordConcept, mean (RT) )
[1] 733.874
with (wordConcept, levels (COND) )
[1] "correct"          "formcontrol"       "formsimilar"
[4] "semanticcontrol" "semanticrelated"

with (wordConcept, boxplot (RT ~ COND) ) #create a boxplot
```

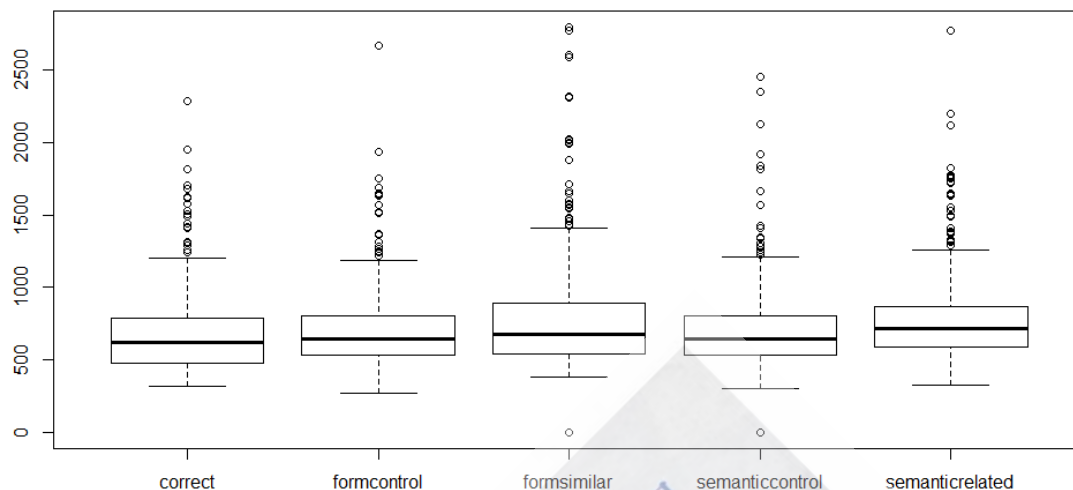



图 1-3 实验条件(COND)和反应时(RT)的箱体图

使用 `with ()` 函数，生成了一副箱体图，这个图展示了 COND 和 RT 的关系，COND 一共有 5 个条件。第二章会对箱体图进行详细介绍。

1.2.2 简单数据框 tibble 的操作和管理

R 语言从诞生到现在已经有很长的时间，其中数据框(data frame)是它最传统的功能，但是随着时代的发展，大家也越来越意识到使用传统的数据框在数据操作上会有很多的不便，但是要对它进行彻底的革命又是一件几乎不可能的事情，因此只能以革新数据包的形式进行一些改变。tibble 就是一种改变，它是一种简单的、弱类型的数据框，它与 data.frame 有相同的语法，但使用起来更方便。RStudio 官方把 tibble 项目集成到了 tidyverse 项目中，因此安装了 tidyverse 包也就可以使用 tibble 的所有功能。

1.2.2.1 与传统的数据框比较

tibble 保留了数据框(data.frame)中经过实践证明有效的部分，但也在数据框的基础上进行了很多改进，提供了一些更优于数据框的性能，包括打印，因子操作和提取子集。

首先，关于打印，它的显示进行了优化，只显示前 10 行的数据结果，并且在显示的时候会自动适合屏幕，这对大规模数据具有很大的优势。更重要的改进是在显示的时候，它会直接显示每一个变量的(即列)的类型，这直接实现了在传统的数据框里使用 `str ()` 实现的功能：

```
myData
# A tibble: 2,000 x 11
  LIST SUBJ TRIAL CHINITEMS ACC RT COND ENGITEMS ORDER TYPE
  <chr> <int> <int> <chr> <int> <int> <chr> <chr> <int> <chr>
1 List1 2 72 10deng.PNG 1 940 correct 10light.PNG 10 NA
2 List1 2 78 1pingguo.PNG 1 658 correct 1apple.PNG 1 NA
3 List1 2 27 2mao.PNG 1 532 correct 2cat.PNG 2 NA
4 List1 2 22 3shizhong.PNG 1 938 correct 3clock.PNG 3 NA
5 List1 2 75 4gongdian.PNG 1 767 correct 4palace.PNG 4 NA
6 List1 2 58 5gangqin.PNG 1 2286 correct 5piano.PNG 5 NA
7 List1 2 36 6zhu.PNG 1 692 correct 6pig.PNG 6 NA
8 List1 2 41 7she.PNG 1 682 correct 7snake.PNG 7 NA
9 List1 2 16 8juchang.PNG 1 885 correct 8theater.PNG 8 NA
10 List1 2 34 9xihongshi.PNG 1 818 correct 9tomato.PNG 9 NA
# ... with 1,990 more rows, and 1 more variable: RELATEDNESS <chr>
```

上面显示的就是一个 tibble 数据框，在第一列的下面多了一行，显示了这个列(即变量)的类型，如<chr>、<int>等，前者代表 character(字母)，后者代表 integer 即整数。这里要特别提到一点，传统的数据框在导入 R 以后，会自动把分类数据(字符串型)转变成因子(factor)，但是 tibble 并不会这么做，而是保留为 character，这一点非常重要。因子是传统的数据框非常重要的一种类型，但是 tibble 没有因子这种类型，而是保存了为 character，如果没有意识到这一点，或者不熟悉二者的转化，可能会给操作带来问题。比如，我们在对数据进行分析时，经常要查看分类变量的水平，在传统的数据框里很简单，就像前面 2.1.3.1 里提到的可以使用 levels() 函数来查看，但是由于 tibble 数据框不是把分类变量变成因子，因此无法使用 levels() 函数来查看因子水平。在 1.2.2.4 小节会进行介绍。

最后，关于 tibble 数据框提取子集的方法也是它跟传统的数据框不同的地方，我们在前面已经提到了部分内容，下面会更详细讲解。先看看如何导入 tibble 数据。

1.2.2.3 tibble 数据的导入

我们上面说的 tibble 和传统的数据框(data frame)的区别是指导入 R 之后它们之间的区别。也就是说，我们使用其他软件所准备好的数据，比如 Excel 表格或者 csv 表格等，可以按两种方式导入 R，一种是以传统的数据框形式，一种是以 tibble 的形式导入。第一种我们在前面已经详细介绍过。这里介绍如何以 tibble 形式导入到 R。

(1) csv 格式数据的导入。要把 csv 格式数据导入为 tibble，使用的是 read_csv() 函数，特别注意这里是一个小横线(-)，不是点。先使用这个函数导入一个数据：

```
library(tidyverse)
myData <- read_csv(file.choose())
或者
myData <- read_csv("E:/bookR/WrdNonWrd.csv")
myData
# A tibble: 160 x 7
  NUMBER COND      TYPE TARGET PRIME    RT  FREQ
  <int> <chr>    <chr> <chr> <dbl> <int>
1     28 Control word  ANGRY  twiek 1199. 29186
2     29 Control word  BAND  lese  995. 37172
3     30 Control word  BIKE  toal  822. 18223
4     31 Control word  COIN  desh  679. 4555
5     32 Control word  CRIME wholk 1178. 49923
6     33 Control word  DARK  bleg 1011 84951
7     34 Control word  EACH  ibnd  948. 350781
8     35 Control word  FAST  eben  856. 51454
9     36 Control word  FRUIT bleve 1244. 22333
10    37 Control word  GLAD  porf 1228 22069
# ... with 150 more rows
```

上面把一个名为 wrdNonwrd.csv 的数据导入到 R 成为 tibble 的形式。导入完后，输入导入的数据名，可以清楚显示前 10 行数据，后面还有 150 行没有显示。而且，跟上面说的一样，还显示了每个变量的种类。

(2) excel 格式数据导入。使用的函数是 `read_xlsx()`，但是要先安装包 `readxl`：

```
library(readxl)
MAdat <- read_xlsx("E:/bookR/MAitems.xlsx")
MAdat
# A tibble: 250 x 8
```

	NUMB	COND	CHINITEMS	ENGITEMS	ORDER	TYPE	RELATEDNESS	RT
	<dbl>	<chr>	<chr>	<chr>	<dbl>	<chr>	<chr>	<dbl>
1	12	correct	苹果	1apple	1	correct	correct	555.
2	23	correct	猫	2cat	2	correct	correct	711.
3	34	correct	时钟	3clock	3	correct	correct	703.
4	45	correct	宫殿	4palace	4	correct	correct	615.
5	47	correct	钢琴	5piano	5	correct	correct	645.
6	48	correct	猪	6pig	6	correct	correct	612.
7	49	correct	蛇	7snake	7	correct	correct	572.
8	50	correct	剧场	8theater	8	correct	correct	709.
9	51	correct	西红柿	9tomato	9	correct	correct	797.
10	1	correct	灯	10light	10	correct	correct	761.

```
# ... with 240 more rows
```

第一行是把包 `readxl` 加载进来，然后使用 `read_xlsx()` 函数导入。如果导入的更老一些的 excel 版本即 xls 文件，就使用 `read_xls()` 函数导入。前面说过，把 excel 文件导入为数据框时，速度往往比较慢，但幸运的是，使用 `readxl` 包会快很多。其他格式数据的导入，大家可以查看其他资料。

如果一个数据框不是 `tibble` 类型的数据框，可以按下面的命令把它转换：

```
x <- as.tibble(x)
```

上面的命令就把 `x` 数据框转换成了 `tibble` 类型的数据框。同样，如果想把一个 `tibble` 类型的数据框转换成传统的数据框，可以按下面的命令：

```
x <- as.data.frame(x)
```

上面的命令又把 `x` 转换成了传统的数据框。简单概括如下：

`read_csv()` 读取逗号分隔文件，`read_csv2()` 读取分号分隔文件，`read_tsv()` 读取制表符分隔文件，`read_delim()` 可读取使用任意分隔符的文件。`read_fwf()` 读取固定宽度的文件。详情，可以查看许多关于 R 基础的书籍。

当然，也可以使用 `readr` 包中两个非常有用的函数来把 R 中的数据导出到电脑：`write_csv()` 和 `write_tsv()`。在导入的时候，最重要的就是要详细指明要导入的数据框的名称以及贮存的路径和文件名，比如：

```
write_csv(Rec,"E:/bookR/translation.csv")
```

上面这行命令就是把 R 中一个名为 `Rec` 的数据框导出到电脑的 E 盘里的 `bookR` 文件夹中，并被命名为 `translation.csv`。

1.2.2.4 一些操作

我们前面介绍过的所有的应用于传统的数据框的函数和命令同样也可以应用于 tibble 数据框。但是有一个很大的区别就在于因子的操作。tibble 不像传统的数据框把字符形变量自动变成因子，tibble 仍然把它们保留为 character，因此，在 tibble 数据框里就无法查看一个分类变量的水平。但是，在实际的数据分析中，却要经常查看一个因子的水平，那应该怎么办呢？一种方法是使用 count () 函数，以上面刚导入的数据 MAdat 为例：

```
MAdat %>% count (COND)
```

```
# A tibble: 5 x 2
```

COND	n
<chr>	<int>
1 correct	50
2 formcontrol	50
3 formsimilar	50
4 semanticcontrol	50
5 semanticrelated	50

这里使用了一个符号 %>%，这个符号叫管道，它的作用是传输，这里相当于把数据 MAdat 传给后面的 count () 函数，结果可以看出 COND 一共有 5 个水平，每个水平有 50 个数据。也可以使用条形图查看：

```
MAdat %>% ggplot ( aes (COND) ) + geom_bar ()
```

大家自行查看，这个方法直接、简单。

接下来要介绍的是 tidyverse 包中的 5 个核心函数。这个 5 个函数非常重要，熟练使用它们可以帮助解决数据处理中的绝大多数问题。这五个函数当中的每个函数都有自己独特的功能，但是，如果把它们组合起来，功能就进一步得到的加强。要记住的是，在进行下面所有的操作之前，都必须先加载包：

```
library (tidyverse)
```

1.2.2.4.1 filter () 选子集

它类似于在前面介绍的 subset () 函数，作用就是可以按照我们的条件筛选出数据子集。比如我们在上面导入的数据 MAdat 当中的变量 COND 有五个条件，现在只想挑选出 formcontrol 和 formsimilar 两个条件的数据：

```
library (readxl)
```

```
MAdat <- read_xlsx ("E:/bookR/MAitems.xlsx")
```

```
MAdat2 <- filter (MAdat, COND == "formcontrol" | COND == "formsimilar")
```

```
MAdat2 %>% count (COND)
```

```
# A tibble: 2 x 2
```

COND	n
------	---

```

      <chr>      <int>
1 formcontrol    50
2 formsimilar    50

```

通过 `count()` 函数可以发现，经过 `filter()` 函数筛选后，COND 只有两个水平了。请思考为什么使用 `|` 符号把两个条件连接起来，而不是使用 `&` 连接符号呢？如果要在 5 个条件中，去除 `correct` 这个条件，那应该怎么办呢？如下：

```

MAdat3 <- filter (MAdat, COND != "correct")
MAdat3 %>% count (COND)

```

```

# A tibble: 4 x 2

```

```

  COND      n
  <chr>  <int>
1 formcontrol    50
2 formsimilar    50
3 semanticcontrol 50
4 semanticrelated 50

```

1.2.2.4.2 mutate () 修改数据框

前文已经介绍过 `mutate()` 函数的使用，它的作用相当于 `transform()` 函数。比如，想把上面的 MAdat 函数中的 RT 转变成它的标准分数即 z 分数，并增加一列，应该怎么办呢？执行下面的命令：

```

MAdat <- mutate (MAdat, RTzscore = scale (RT) )
MAdat

```

这个时候就可以看到在数据框的最左边增加一列叫做 `RTzscore`，它就是前面 RT 那一列的标准分。`scale()` 函数的作用是把数据变成标准分。

1.2.2.4.3 select () 选择列

有的时候，导入的数据有很多变量，有些变量并不是自己要的，可以作用 `select()` 函数选出自己要的数据，这样操作起来就会非常简单：

```

MAdat4 <- select (MAdat, COND, CHINITEMS, RT)

```

```

MAdat4

```

```

# A tibble: 250 x 3

```

```

  COND  CHINITEMS  RT
  <chr>  <chr>    <dbl>
1 correct 苹果    555.
2 correct 猫      711.
3 correct 时钟    703.
4 correct 宫殿    615.
5 correct 钢琴    645.
6 correct 猪      612.

```

```

7 correct 蛇          572.
8 correct 剧场          709.
9 correct 西红柿        797.
10 correct 灯           761.
# ... with 240 more rows

```

可以看到，现在数据只剩下了 3 列，更简单，更容易操作。也可以使用 - 符号来去除不需要的列，比如：

```

library(readxl)
MAdata <- read_xlsx("E:/bookR/MAitems.xlsx")
MAdata_one <- select(MAdata, -ENGITEMS)
MAdata_two <- select(MAdata, -(ENGITEMS:ORDER))

```

请读者运行上面的代码，看看分别去除了哪些列。

1.2.2.4.4 arrange () 排序

在传统的数据库里，排序的函数有 sort () 或者 order ()，但是操作不便，但是使用 arrange () 函数非常方便，默认的是按条件进行升序排序，可以同时放几个排序条件，先排第一个条件，再排第二个条件，依次类推：

```

library(readxl)
MAdata <- read_xlsx("E:/bookR/MAitems.xlsx")
arrange(MAdata, COND, CHINITEMS, RT)
# A tibble: 250 x 8

```

	NUMB	COND	CHINITEMS	ENGITEMS	ORDER	TYPE	RELATEDNESS	RT
	<dbl>	<chr>	<chr>	<chr>	<dbl>	<chr>	<chr>	<dbl>
1	41	correct	白色	46white	46	correct	correct	509
2	20	correct	班级	27class	27	correct	correct	659.
3	10	correct	杯子	18cup	18	correct	correct	560.
4	5	correct	床	13bed	13	correct	correct	578.
5	9	correct	大衣	17coat	17	correct	correct	698
6	39	correct	单车	44bike	44	correct	correct	572.
7	27	correct	蛋糕	33cake	33	correct	correct	664.
8	1	correct	灯	10light	10	correct	correct	761.
9	4	correct	儿子	12son	12	correct	correct	564.
10	33	correct	二	39two	39	correct	correct	626.

... with 240 more rows

arrange () 默认是按升序排，但是它可以跟 desc () 函数结合起来，实现降序排：

```
arrange(MAdata, desc(CHINITEMS))
```

执行这个命令就可以按 CHINITEMS 进行降序排。可见这是一个非常好用的函数。

1.2.2.4.5 summarize () 分组运算

这个函数实在太好用了，可以进行许多非常有用的运算，更重要的是它可以跟 group_by () 函数结合起来，使得 summarize () 函数可以根据分组进行各种运算。这里暂时不介绍，留到下个章节讲平均数运算时再说吧。

上面 5 个函数的功能非常强大，刚开始接触可能不会有感觉，但是到了后面，尤其是要面对大量数据进行运算的时候，它们的作用就体现了。可以这样说，掌握这 5 大函数就掌握了数据管理的重要工具。

1.2.2.4.6 sample_n () 和 sample_frac ()

sample_n () 随机选出指定个数或者说样本容量的样本数。比如从 wordConcept 数据框中随机选择 8 个样本：sample_n(wordConcept, 8)。或者选出占整个数据集总体百分比的样本数，比如从 wordConcept 数据框中选取占 5% 的样本数：sample_frac(wordConcept, 0.05)。

对了，还有一个有用的工具没有介绍，那就是管道，即 %>%，它的使用往往让一切操作显得非常连贯，事半功倍。对它的使用，后面慢慢接触、积累吧。

1.2.2.5 长、宽数据的相互转换

在整理数据的时候，我们可能经常要碰到把长数据转换成宽数据或者把宽数据转换成长数据的问题，尤其是在分析处理由 SPSS 导入的数据时可能更是如此，因为使用 SPSS 做重复测量的方差分析时使用的基本都是宽数据。

可以使用 gather () 和 spread () 两个函数来实现两种数据格式之间的相互转换。先导入数据：

```
rm (list = ls (all=TRUE) )  
library (readxl); library (tidyverse)  
maData <- read_xlsx ("E:/bookR/MApercept.xlsx")  
maACC <- aggregate (ACC ~ SUBJ + VOWCONTR + TYPE, sum, data = maData)
```

maACC				
	SUBJ	VOWCONTR	TYPE	ACC
1	1	bad-bed	change	2
2	2	bad-bed	change	0
3	3	bad-bed	change	2
4	4	bad-bed	change	5
34	1	bad-bed	nochange	8
35	2	bad-bed	nochange	8
36	3	bad-bed	nochange	7
37	4	bad-bed	nochange	4

可以看出，在 maACC 数据表中，每名被试的每一对 VOWCONTR(元音对比音)有两行数据，分别对应 TYPE 列 change 和 nochange。现在要把这个长数据转换成宽数据，把 TYPE 这一列分成两列，分别为 change 和 nochange，使用 spread () 函数：

```
maACC.1 <- spread (maACC, key = TYPE, value = ACC)
```

maACC.1

转换后的数据如下：

	SUBJ	VOWCONTR	change	nochange
1	1	bad-bed	2	8
11	2	bad-bed	0	8
21	3	bad-bed	2	7
31	4	bad-bed	5	4

可以看出，TYPE 被拆成两列，分别为 change 和 nochange，而原来数据 ACC 的值被填充到了这两列中。从上面的代码可以看出，spread() 函数有两个关键的参数对象分别是 key 和 value，key 就是要被拆解的变量，而 value 就是拆后每个变量对应的值。

gather() 函数正好与 spread() 相反，它的作用是把长数据转换成宽数据。比如，要把上面的 maACC.1 转成原来的长数据，可以执行下面的代码：

```
x <- maACC.1 %>%  
  gather('change', 'nochange', key = TYPE, value = ACC)
```

可以看出在实现两种数据的转换时，关键的是设置 key 和 value 这两个参数对应的值，而这要根据数据的实际情况进行设置。还需要引起注意的是 change 和 nochange 分别使用了单引号(或双引号)来标识，不然 R 也会报错。