R数据读写与预处理

理工大学理学院–周世祥

2015年10月27日

目录

1 R数据读入

R的数据读入灵活方便，可以在R软件中直接输入，也可以读入外部文件。[1]

对于大数据量来说，一般需要从外部读入数据，外部的数据源很多，可以是网络、电子表格、数据库、文本文件、论文等形式，所以录入数据的方法也就很多。

关于R的导入导出，可以阅读“R Data Import/Export”，下面我们介绍一下R读入各种数据的不同方法，各种方法都有自己的优势，至于哪种方法最好要根据实际的数据情况来决定。

1.1 直接数据读入

1.1.1 c()函数

c函数是把各个值联成一个向量或列表，可形成数值型向量、字符型向量或其它类型向量，它的使用非常简单。该函数我们在前面已多次使用，例如：

```
  x=c(1,2,3,4)
  x

## [1] 1 2 3 4

   y=c("a","b","c")
  y

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

1.1.2 scan()函数

scan函数功能类似于c函数，scan实际上是一种键盘输入数据函数。当你输入scan()，然后按回车键，这时将等待你输入数据，数据之间只要空格分开即可（c函数要用逗号分开）。输入完数据，再按回车键，这时数据录入完毕。例如：

[1]这是第一稿，写的还是比较乱，只作为同学们课后学习参考，欢迎帮忙修改!

```
x = scan()
```

scan函数还可以读入外部文本文件。假如你现在有一个文本文件，dat.txt，读入这个文件的命令是：

```
x=scan(file="dat.txt")
```

假如原文件的数据之间有逗号等分隔符，用scan读入应该去掉这些分隔符，其命令是：

```
x=scan(file="dat.txt",sep=",")
```

在RStudio中，虽然可以直接查看，读取和修改数据，但还有一些操作还是需要使用命令来运行，例如：

- 查看当前环境下的数据ls()；
- 删除数据rm()；
- 查看所有预先提供的数据data()；
- 查看某个包所有预先提供的数据data(package=””)；
- 读入数据data(datasetname)

1.2 读R包中数据

如果需要从其他的软件包链接数据，可以使用参数package，例如我要查看SemiPar包里有哪些自带的数据，可以用以下代码来查看：

```
# install.packages("SemiPar")
data(package="SemiPar")
```

如果需要读入SemiPar包里copper数据，可以输入以下代码：

```
data(copper,package="SemiPar")
```

如果一个软件包已被library附加在库中，则这个数据库将自动地被包含在其中，如

```
library(SemiPar)
data()
 data(fossil)
```

1.3 从外部文件读数据

1.3.1 读文本文件

大的数据对象常常是从外部文件读入，而不是在R中直接键入的。读入文本文件数据的命令是

read.table(file,header=logical_value,sep=”delimiter”,row.names=”name”)

file是一个带分隔符的ASCII文本文件，header是一个表明首行是否包含了变量名，sep用来指定分隔符。

但它对外部文件常常有特定的格式要求：第一行可以有该数据框的各变量名，随后的行中第一个条目是行标签，其它条目是各个变量的值。在文件中，第一行比第二行少一个条目，这样做是被强制要求的，因此一个被看作数据框读入的文件格式应是这样的。

例如学生数据存在student.txt文本文件中，默认情况下，数值项(除了行标签)以数值变量的形式读入，对应非数值变量则以因子形式读入，如sex变量。

```
   s1=read.table("student.txt" ,fileEncoding="UTF-16LE")
    s1 #此处有误，加了一个选项才可以：fileEncoding="UTF-16LE"

##   class sex score
## 1     1  女    80
## 2     1  男    85
## 3     2  男    92
## 4     2  女    76
## 5     3  女    61
## 6     3  女    95
## 7     3  男    83
```

常常需要忽略掉行标签而直接使用默认的行标签。这种情况下，输入文件如下面一样省略行标签。这时，可以用如下命令读入：

```
  s2=read.table("student.txt",header=T,sep="",fileEncoding="UTF-16LE")
  s2

##   class sex score
## 1     1  女    80
## 2     1  男    85
## 3     2  男    92
## 4     2  女    76
## 5     3  女    61
## 6     3  女    95
## 7     3  男    83
```

需要说明的是，此处默认student.txt文件保存在当前工作目录下，如果文件不是保存在当前工作目录的话。有两种方法去处理：

第一，在R或者Rstudio更改student.txt文件所在的文件夹为当前工作目录；

第二，输入路径名，比如student.txt文件保存在E:\ R\ data目录下，则可以输入命

令read.table("E:\\R\\data\\student.txt",header=T)。注意，R里输入路径名，需要把“\ ”改为“\\”或者“/”。

1.3.2 读入EXCEL格式数据

对于一般常用的xls, xlsx数据表，由于该格式比较复杂，尽量避免直接导入。通常处理是将xls转换为csv文件，转换方法可以在EXCEL里点击另存为csv格式。

读入csv格式数据，可以通过read.csv()函数来读入。

read.csv(file="file.name", header = TRUE, sep = ",", ...)

其中，header表示是否含有列名，sep表示csv文件的分隔方式，一般是逗号分隔符。

例如读入student.csv格式的数据

```
  S2=read.csv(file="student.csv")
  S2
```

```
##   class sex score
## 1     1  女    80
## 2     1  男    85
## 3     2  男    92
## 4     2  女    76
## 5     3  女    61
## 6     3  女    95
## 7     3  男    83
```

当需要读入的文件不是保存在当前工作目录时，处理方法类似上文。

R也可以直接读入xlsx格式的文件，需要安装xlsx包，但OS X平台目前尚不支持该包，另外有名为xlsx的包来导入Excel数据(需要在电脑上安装Java运行环境)。它可以读取Excel文件为数据框，以及将数据框写入为Excel文件，而且还能处理Excel中的格式，比如合并单元格，设置列的宽度，设置字体和颜色等等。读取/写入操作分别使用read.xlsx()和write.xlsx()，还有两个相应的函数read.xlsx2()和write.xlsx2()，这两个函数效率相对会更高一些。

1.3.3 读入其他格式数据

要读入其他格式的数据，必须先安装和载入“foreign”包。

```
# install.packages("foreign")
library(foreign)
```

SAS数据：

对于SAS，R只能读入SAS Transport format(XPORT)文件。所以，需要把普通的SAS数据文件（.ssd和.sas7bdat）转换为Transport format(XPORT)文件,再用命令：read.xport( )。

例如：

```
# read.xport("dataname.xpt") # 读入SAS格式文件
```

SPSS数据：

read.spss( )可读入SPSS数据文件。例如：

```
 # read.spss("dataname.sav") # 读入 SPSS格式文件
```

也可以使用Hmisc包中的spss.get()函数。spss.get()是对read.spss( )的一个封装，可以自动设置很多参数，整个转换过程更加简单一致。

```
# install.packages("Hmisc")  #安装Hmisc包
# library(Hmisc) #载入Hmisc包
# mydataframe<-spss.get("dataname.sav", use.value.labels=TRUE)
```

Stata数据: R可读入Stata5,6,7的数据库。命令如下：

```
# read.dta("文件名.dta")
```

读入数据文件后，使用数据集名$变量名，即可使用各个变量。如：

```
# mean(data£age)
```

便是计算数据集data中的变量age的均数。

2 数据的写出

在将R工作空间里面的数据输出存储的时候，可以使用write()函数

```
write(x, file = "data",  ncolumns = if(is.character(x)) 1
else 5,  append = FALSE, sep = " ")
```

其中，x是数据，通常是矩阵，也可以是向量；file是文件名(默认时文件名为“data”)；append=TRUE时，在原文件上添加数据，否则（FALSE，默认值）写一个新文件。其他参数参见帮助文件。

若对于列表数据数据或数据库数据，可以用write.table()函数或write.csv()函数写纯文本格式的数据文件，或CSV格式的Excel数据文件。

write.table()函数和write.csv()函数的使用格式为

```
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 = "")
  write.csv(...)
  write.csv2(...)
```

例如，将上文的S2数据写出到当前工作目录，命名为S2.txt的文件里Write.table(S2, “S2.txt”)

此时，会在当前工作目录下新创建了一个名为S2.txt的文件例如将上文的S2数据写出到E:\\R\\data目录下，命名为S2.csv的文件里

Write.table(S2, “E:\\R\\data\\S2.csv”)

此时会在E:\\R\\data目录下新创建了一个名为S2.csv的文件。

随着各种技术不断发展，收集数据的技术和渠道日益广泛，这些收集到的数据构成了数据分析的源数据，但是由于各种各样的原因，如市场调查中的无回答、数据输入错误等，导致了源数据的各种质量问题。例如数据缺失、异常点的出现等，都会给数据分析带来困难。事实上，我们的数据分析过程中，有多达60%的时间都花在了实际分析前数据的准备上。许多需要处理的源数据可能都面临着以某种形式存在的类似问题。

3 数据预处理

下面我们看一个例子。在R的VIM包中，有一组关于哺乳动物的睡眠数据（sleep），该数据来源于Allison和Chichetti(1976) 的研究。提供了62种哺乳动物睡眠、生态学变量和体质变量的关系，一共62个观测值，10个变量。睡眠变量包含睡眠中做梦时长(Dream)，不做梦时长(NonD) 以及它们的和(Sleep)。体质变量包含身体体重(BodyWgt，单位为千克)、脑重(BrainWgt，单位为千克)、寿命(Span，单位为年) 和妊娠期(Gest，单位为天)。生态学变量包含物种被捕食的程度(Pred)、睡眠时暴露的程度(Exp) 和面临的总危险程度(Danger)。生态学变量以从1 (低) 到 5 (高)的5分制进行度量。

```
  # install.packages("VIM")
library(VIM)#
```

```
## Loading required package: colorspace
## Loading required package: grid
## Loading required package: data.table
## VIM is ready to use.
##  Since version 4.0.0 the GUI is in its own package VIMGUI.
##
##          Please use the package to use the new (and old) GUI.
##
## Suggestions and bug-reports can be submitted at: https://github.com/alexkowa/VIM/is
##
## Attaching package: 'VIM'
##
## The following object is masked from 'package:datasets':
##
##    sleep
```

```
data(sleep)
head(sleep)   #我们列举了前10个观测值如下:
```

```
##    BodyWgt BrainWgt NonD Dream Sleep Span Gest Pred Exp Danger
## 1 6654.000   5712.0   NA    NA   3.3 38.6  645    3   5      3
## 2    1.000      6.6  6.3   2.0   8.3  4.5   42    3   1      3
## 3    3.385     44.5   NA    NA  12.5 14.0   60    1   1      1
## 4    0.920      5.7   NA    NA  16.5   NA   25    5   2      3
## 5 2547.000   4603.0  2.1   1.8   3.9 69.0  624    3   5      4
## 6   10.550    179.5  9.1   0.7   9.8 27.0  180    4   4      4
```

[2]

安装VIM需要安装一堆依赖包。

```
package 'minqa' successfully unpacked and MD5 sums checked
package 'nloptr' successfully unpacked and MD5 sums checked
package 'RcppEigen' successfully unpacked and MD5 sums checked
package 'lme4' successfully unpacked and MD5 sums checked
```

[2]这里格式不好调整!

```
package ‘SparseM’ successfully unpacked and MD5 sums checked
package ‘MatrixModels’ successfully unpacked and MD5 sums checked
package ‘zoo’ successfully unpacked and MD5 sums checked
package ‘pbkrtest’ successfully unpacked and MD5 sums checked
package ‘quantreg’ successfully unpacked and MD5 sums checked
package ‘DEoptimR’ successfully unpacked and MD5 sums checked
package ‘lmtest’ successfully unpacked and MD5 sums checked
package ‘car’ successfully unpacked and MD5 sums checked
package ‘robustbase’ successfully unpacked and MD5 sums checked
package ‘vcd’ successfully unpacked and MD5 sums checked
package ‘e1071’ successfully unpacked and MD5 sums checked
package ‘VIM’ successfully unpacked and MD5 sums checked
```

问题：

- 若我们想构造一个新变量，直接列出做梦时长占全部睡眠时间的比率，或想改变某变量的值，该如何操作。
- NonD这个变量名难以理解，如何改成一个直观的名字代替之。
- Pred ,Exp和Danger录入时均为数值型，如何把它们变成因子型变量。
- 表中存在许多缺失值(NA)，仅仅只有62个观测值，如何弥补。
- 数据的合并，提取问题。

3.1 变量预处理

3.1.1 创建新变量

在研究中，需要对现有的一个或者几个变量进行变换创建一个新的变量。

第一种方法，左边为变量名，右边为其他变量的表达式：

var_Names =expr

第一种方法，对现有的指标直接进行运算变换

```
  # pgnp<-gnp/pop
  # psave<-(gnp-cons)/pop
```

第二种方法用transform()函数进行变换

```
#  data<-data.frame(gnp,cons,pop)
# transform(data,pgnp=gnp/pop,psave=(gnp-cons)/pop )
```

第三种方法用 with()函数进行变换。该函数的用法是with(data, expr, ...)，表示对data 执行expr运算。

```
  # (pgnp<-with(data,gnp/pop ))
# [1] 3.000000 2.000000 2.114286 2.191235 1.508197
# (psave<-with(data,(gnp-cons)/pop))
```

3.1.2 变量重编码

对变量进行重编码在数据预处理中经常会碰到的问题。比如：

将连续型变量进行分组，变成一个离散型变量。

将某个值替换成另外一个值。

要重编码可以使用R中的一个或者多个逻辑运算符，逻辑运算符返回的结果为TRUE或FALSE。

例子：以MASS 包中的 Cars93 数据为例，该数据是关于不同汽车制造商不同款式的汽车价格等数据，共有27个变量。现在假如我们只关心制造商和价格变量

```
library(MASS)
data(Cars93)
dat<-data.frame(manu=Cars93$Manufacturer,price=Cars93$Price)
head(dat)

##    manu price
## 1 Acura  15.9
## 2 Acura  33.9
## 3  Audi  29.1
```

```
## 4  Audi  37.7
## 5   BMW  30.0
## 6 Buick  15.7
```

由于价格是连续型数值变量，假如我们打算将价格按照(0,12) ,(12,20),(20,max(price))分成“cheap”，”okay”，”expensive”三种类型。

第一种方法利用[]将price分成不同的取值区间，然后用“cheap”，”okay”，”expensive”分别进行替换。

```
dat$pricegrade<-NA
dat$pricegrade[dat$price>=20]<-"expensive"
dat$pricegrade[dat$price>=12&dat$price<20]<-"okay"
dat$pricegrade[dat$price<12]<-"cheap"
head(dat)
```

```
##    manu price pricegrade
## 1 Acura  15.9       okay
## 2 Acura  33.9  expensive
## 3  Audi  29.1  expensive
## 4  Audi  37.7  expensive
## 5   BMW  30.0  expensive
## 6 Buick  15.7       okay
```

第二种方法使用within()函数，该函数与with()用法相似，不同的是该函数可以对数据框进行修改。

```
dat<-within(dat,{
 pricegrade<-NA
 pricegrade[price>=20]<-"expensive"
 pricegrade[price>=12&dat$price<20]<-"okay"
 pricegrade[price<12]<-"cheap"
   })
head(dat)
```

```
##    manu price pricegrade
```

```
## 1 Acura  15.9       okay
## 2 Acura  33.9  expensive
## 3  Audi  29.1  expensive
## 4  Audi  37.7  expensive
## 5   BMW  30.0  expensive
## 6 Buick  15.7       okay
```

第三种方法利用cut()函数先将连续取值的price分成三个区间，然后再用levels()函数将不同的区间用“cheap”，”okay”，”expensive”分别进行命名。

```
  dat$pricegrade<-cut(dat$price,c(0,12,20,max(dat$price)))
    levels(dat$pricegrade)<-c("cheap","okay","expensive")
    head(dat)

##    manu price pricegrade
## 1 Acura  15.9       okay
## 2 Acura  33.9  expensive
## 3  Audi  29.1  expensive
## 4  Audi  37.7  expensive
## 5   BMW  30.0  expensive
## 6 Buick  15.7       okay
```

另外，car包中recode()提供了非常简便的重编码方法。比如我们需要对前面的“cheap”，”okay”，”expensive”三种类型分别替换成“A”,”B”,”C”.

```
    # install.packages("car")
    library(car)

##
## Attaching package:  'car'
##
## The following object is masked from 'package:SemiPar':
##
##    spm
```

```
  recode(dat$pricegrade,"'cheap'='A';'okay'='B';'expensive'='C'")

##  [1] B C C C C B C C C C C B A B B B B B C B B C A A B B B C B B A A A B B
## [36] B C C A B B B B A A A B C C C C C C A A B B C C C B B A C A B B C B B B
## [71] C B A A B B C C A A A B A A B B C A B B C C C
## Levels: A B C
```

3.1.3 变量重命名

第一种方法，使用交互式编辑器：

在命令窗口输入fix(data)，R会自动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。

第二种方法，reshape包中有一个rename() 函数，可用于修改变量名。其调用格式为：

rename(dataframe, c(oldname1=“newname1”, oldname2=”newname2”, ···))

例如，将上例中数据框dat中pricegrade重命名为grade。

```
 # dat1 <- rename(dat, c(pricegrade="grade"))
 # head(dat1)
```

第三种方法，直接通过names() 函数来重命名变量。例如：

```
# names(dat)
# names(dat)[3] <- "grade"
# head(dat)
```

3.1.4 变量类型转换

R中提供了一系列用来判断某个对象数据类型和将其转换为另一种数据类型的函数。名为is.datatype() 这样的函数返回TRUE或FALSE，而as.datatype() 这样的函数则将其参数转换为对应的类型。

```
    a <- c(1,2,3);a
## [1] 1 2 3
 is.numeric(a)
## [1] TRUE
 is.vector(a)
## [1] TRUE
  a <- as.character(a);a
## [1] "1" "2" "3"
  is.numeric(a)
## [1] FALSE
  is.vector(a)
## [1] TRUE
  is.character(a)
## [1] TRUE
```

数据类型	辨别函数	转换函数
numeric	is.numeric()	as.numeric()
character	is.character()	as.character()
complex	is.complex()	as.complex()
double	is.double()	as.double()
integer	is.integer()	as. integer()
logical	is. logical()	as. logical()
NA	is.na()	as.na()

表 1: 辨别和转换数据对象类型的函数

3.1.5 日期型变量转换

日期值通常以字符串的形式输入到R中，然后转化为数值形式储存的日期变量。函数as.Date() 用于执行这种转化。语法为 as.Date(x, “input_format”)，其中x是字符型数据，input_format 则给出了用于读入日期的适当格式。

日期的默认输入格式为yyyy-mm-dd。语句：

```
madates <- as.Date(c("2010-06-22", "2013-09-14"))
```

将默认格式的字符型数据转换对应日期。相反，

```
  strDates <- c("01/05/1965","08/16/1975")
dates <- as.Date(strDates," %m/%d/%Y")
```

则使用mm/dd/yyyy的格式读取数据。

4 缺失值处理

缺失值从缺失的分布来讲可以分为完全随机缺失，随机缺失和完全非随机缺失。

完全随机缺失（missing completely at random, MCAR）指的是数据的缺失是随机的，数据的缺失不依赖于任何不完全变量或完全变量。缺失情况相对于所有可观测和不可观测的数据来说，在统计意义上是独立的。

随机缺失(missing at random, MAR)指的是数据的缺失不是完全随机的，即该类数据的缺失依赖于其他完全变量。也就是，一个观测出现缺失值的概率是由数据集中不含缺失值的变量决定的, 而不是由含缺失值的变量决定的。

完全非随机缺失(missing not at random, MNAR)指的是数据的缺失依赖于不完全变量自身，是与缺失数据本身存在某种关联，比如问题设计过于敏感造成的缺失。

从统计上说, 非随机缺失的数据会产生有偏估计, 因此不能很好地代表总体。其次, 它决定数据插补方法的选择。随机缺失数据处理相对比较简单, 但非随机缺失数据处理比较困难, 原因在于偏差的程度难以把握。事实上，绝大部分的源数据都包含不完整的观测值，因此怎样处理这些缺失值就很重要了。

在R中，缺失值以符号NA表示。同样我们也可以使用赋值语句将某些值重编码为缺失值，例如一些问卷中，年龄值被编码为99，在分析这一数据集之前，必须让R明白本例中的99表示缺失值。例如：

```
# dataframe£age[dataframe£age == 99] <- NA
```

任何等于99的年龄值都将被修改为NA。在进行数据分析前，要确保所有的缺失数据被编码为缺失值，否则分析结果将失去意义。

4.1 缺失值的检测

R提供了一些函数，用于识别包含缺失值的观测。函数is.na() 检测缺失值是否存在。假设你有一个向量：

```
y <- c(1,2,3,NA)
#然后使用函数:
is.na(y) #将返回 c(FALSE, FALSE, FALSE, TRUE) .

## [1] FALSE FALSE FALSE  TRUE
```

函数complete.cases() 可用来识别矩阵或者数据框中没有缺失值的行。若每行都包含完整的实例，则返回TRUE；若每行有一个或多个缺失值，则返回FALSE。由于逻辑值TRUE和FALSE分别等价于数值1和0，可用sum() 和mean() 来获取关于缺失数据有用的信息。

例子：依然以VIM包中的sleep数据为例。

```
data(sleep, package= "VIM") #读取VIM包中的sleep数据

  sleep[!complete.cases(sleep),] #提取sleep数据中不完整的行

##     BodyWgt BrainWgt NonD Dream Sleep Span Gest Pred Exp Danger
## 1  6654.000   5712.0   NA    NA   3.3 38.6  645    3   5      3
## 3     3.385     44.5   NA    NA  12.5 14.0   60    1   1      1
## 4     0.920      5.7   NA    NA  16.5   NA   25    5   2      3
## 13    0.550      2.4  7.6   2.7  10.3   NA   NA    2   1      2
## 14  187.100    419.0   NA    NA   3.1 40.0  365    5   5      5
```

```
## 19    1.410     17.5  4.8   1.3   6.1 34.0   NA    1    2      1
## 20   60.000     81.0 12.0   6.1  18.1  7.0   NA    1    1      1
## 21  529.000    680.0   NA   0.3    NA 28.0  400    5    5      5
## 24  207.000    406.0   NA    NA  12.0 39.3  252    1    4      1
## 26   36.330    119.5   NA    NA  13.0 16.2   63    1    1      1
## 30  100.000    157.0   NA    NA  10.8 22.4  100    1    1      1
## 31   35.000     56.0   NA    NA    NA 16.3   33    3    5      4
## 35    0.122      3.0  8.2   2.4  10.6   NA   30    2    1      1
## 36    1.350      8.1  8.4   2.8  11.2   NA   45    3    1      3
## 41  250.000    490.0   NA   1.0    NA 23.6  440    5    5      5
## 47    4.288     39.2   NA    NA  12.5 13.7   63    2    2      2
## 53   14.830     98.2   NA    NA   2.6 17.0  150    5    5      5
## 55    1.400     12.5   NA    NA  11.0 12.7   90    2    2      2
## 56    0.060      1.0  8.1   2.2  10.3  3.5   NA    3    1      2
## 62    4.050     17.0   NA    NA    NA 13.0   38    3    1      1
```

```
  sum(!complete.cases(sleep))
```

```
## [1] 20
```

```
  mean(complete.cases(sleep))
```

```
## [1] 0.6774194
```

结果列出了20个含有一个或多个缺失值的观测值，并有67.7%的完整实例。

4.2 缺失值的处理

4.2.1 行删除

可以通过函数 na.omit() 移除所有含有缺失值的观测。na.omit() 可以删除所有含有缺失数据的行。

```
  newsleep <- na.omit(sleep)
```

删除缺失值后，观测值变为42个。

行删除法假定数据是MCAR（即完整的观测值只是全数据集的一个随机样本）。此例中则为42个实例为62个样本的一个随机子样本。

如果缺失值所占比例比较小的话, 这一方法十分有效。然而, 这种方法却有很大的局限性。它是以减少样本量来换取信息的完备, 会造成资源的大量浪费, 丢弃了大量隐藏在这些对象中的信息。在样本量较小的情况下, 删除少量对象就足以严重影响到数据的客观性和结果的正确性。因此, 当缺失数据所占比例较大, 特别是当缺失数据非随机分布时, 这种方法可能导致数据发生偏离, 从而得出错误的结论。

4.2.2 均值替换法（Mean Imputation）

我们将变量的属性分为数值型和非数值型来分别进行处理。如果缺失值是数值型的, 就根据该变量在其他所有对象的取值的平均值来填充该缺失的变量值; 如果缺失值是非数值型的, 就根据统计学中的众数原理,用该变量在其他所有对象的取值次数最多的值来补齐该缺失的变量值。均值替换法也是一种简便、快速的缺失数据处理方法。使用均值替换法插补缺失数据, 对该变量的均值估计不会产生影响。

但这种方法是建立在完全随机缺失(MCAR) 的假设之上的, 而且会造成变量的方差和标准差变小。同时，这种方法会产生有偏估计, 所以并不被推崇。

4.2.3 多重插补（Multiple Imputation）

在面对复杂的缺失值问题时，MI是最常用的方法，它将从一个包含缺失值的数据集中生成一组完整的数据集。每个模拟的数据集中，缺失数据将用蒙特卡洛方法来填补。由于多重估算技术并不是用单一的值来替换缺失值, 而是试图产生缺失值的一个随机样本, 这种方法反映出了由于数据缺失而导致的不确定，R中的mice包可以用来执行这一操作。基于mice包的分析通常按照以下过程分析：

```
# install.packages("mice")
 library(mice)
```

```
## Loading required package:  Rcpp
## Loading required package:  lattice
## mice 2.22 2014-06-10
```

```
# imp <- mice(data, m)
# fit <- with(imp, analysis)
# pooled <- pool(fit)
# summary(pooled)
```

其中，imp是一个包含m个插补数据集的列表对象，同时还含有完成插补过程的信息。默认m值为5。 analysis是一个表达式对象，用来设定应用于m个插补数据集的统计分析方法，例如线性回归模型的lm()函数、广义线性模型的glm()函数，做广义可加模型的gam()等。 fit是一个包含m个单独统计分析结果的列表对象。

pooled是一个包含这m个统计分析平均结果的列表对象。

```
library(mice)
  data(sleep, package="VIM")
  imp <- mice(sleep, seed=6666)

##
##  iter imp variable
##   1   1  NonD  Dream  Sleep  Span  Gest
##   1   2  NonD  Dream  Sleep  Span  Gest
##   1   3  NonD  Dream  Sleep  Span  Gest
##   1   4  NonD  Dream  Sleep  Span  Gest
##   1   5  NonD  Dream  Sleep  Span  Gest
##   2   1  NonD  Dream  Sleep  Span  Gest
##   2   2  NonD  Dream  Sleep  Span  Gest
##   2   3  NonD  Dream  Sleep  Span  Gest
##   2   4  NonD  Dream  Sleep  Span  Gest
##   2   5  NonD  Dream  Sleep  Span  Gest
##   3   1  NonD  Dream  Sleep  Span  Gest
##   3   2  NonD  Dream  Sleep  Span  Gest
##   3   3  NonD  Dream  Sleep  Span  Gest
##   3   4  NonD  Dream  Sleep  Span  Gest
##   3   5  NonD  Dream  Sleep  Span  Gest
##   4   1  NonD  Dream  Sleep  Span  Gest
```

```
##   4   2  NonD  Dream  Sleep  Span  Gest
##   4   3  NonD  Dream  Sleep  Span  Gest
##   4   4  NonD  Dream  Sleep  Span  Gest
##   4   5  NonD  Dream  Sleep  Span  Gest
##   5   1  NonD  Dream  Sleep  Span  Gest
##   5   2  NonD  Dream  Sleep  Span  Gest
##   5   3  NonD  Dream  Sleep  Span  Gest
##   5   4  NonD  Dream  Sleep  Span  Gest
##   5   5  NonD  Dream  Sleep  Span  Gest
```

```
fit <- with(imp,lm(Dream ~ Span + Gest))
  pooled <- pool(fit)
  summary(pooled)
```

```
##                        est          se          t       df     Pr(>|t|)
## (Intercept)  2.501833581 0.262362748  9.5357805 42.22798 4.304335e-12
## Span        -0.006195764 0.012466066 -0.4970103 42.18735 6.217621e-01
## Gest        -0.003372136 0.001680167 -2.0070243 25.54580 5.543842e-02
##                     lo 95        hi 95 nmis       fmi    lambda
## (Intercept)  1.97244879 3.031218e+00   NA 0.1609651 0.1221460
## Span        -0.03134999 1.895847e-02    4 0.1612599 0.1224179
## Gest        -0.00682876 8.448785e-05    4 0.3035470 0.2510752
```

5 数据框的合并与拆分

有时如果希望分组进行统计分析，或者只分析其中的一部分数据，则可以通过拆分数据集来加以分析，有时又希望把不同组的数据合并起来分析。数据框的拆分与合并是一个互逆的过程。R里拆分数据框和合并数据框很简单，分别用函数unstack( )、stack( )。

例：以dataset 的内置数据PlantGrowth为例进行分析，这是关于植物生长的数据。

```
  data(PlantGrowth)
  PlantGrowth     # 为直观，将其重新排列
```

```
##    weight group
## 1    4.17  ctrl
## 2    5.58  ctrl
## 3    5.18  ctrl
## 4    6.11  ctrl
## 5    4.50  ctrl
## 6    4.61  ctrl
## 7    5.17  ctrl
## 8    4.53  ctrl
## 9    5.33  ctrl
## 10   5.14  ctrl
## 11   4.81  trt1
## 12   4.17  trt1
## 13   4.41  trt1
## 14   3.59  trt1
## 15   5.87  trt1
## 16   3.83  trt1
## 17   6.03  trt1
## 18   4.89  trt1
## 19   4.32  trt1
## 20   4.69  trt1
## 21   6.31  trt2
## 22   5.12  trt2
## 23   5.54  trt2
## 24   5.50  trt2
## 25   5.37  trt2
## 26   5.29  trt2
## 27   4.92  trt2
## 28   6.15  trt2
## 29   5.80  trt2
## 30   5.26  trt2
```

该数据共有三个样本观察值，又分成ctrl（对照组）、trt1（处理组1）、trt2（处理组2）三组，每组10个样本。

对于这种类型的数据，其实也可以数据框拆分成ctrl、trt1、trt2三个变量来储存数据。

```
 data(PlantGrowth)
unPG <- unstack(PlantGrowth)
 unPG
```

```
##    ctrl trt1 trt2
## 1  4.17 4.81 6.31
## 2  5.58 4.17 5.12
## 3  5.18 4.41 5.54
## 4  6.11 3.59 5.50
## 5  4.50 5.87 5.37
## 6  4.61 3.83 5.29
## 7  5.17 6.03 4.92
## 8  4.53 4.89 6.15
## 9  5.33 4.32 5.80
## 10 5.14 4.69 5.26
```

也可以用stack( )把unPG的三组数据形成分组变量。

```
 sPG=stack(unPG)
sPG
```

```
##    values  ind
## 1    4.17 ctrl
## 2    5.58 ctrl
## 3    5.18 ctrl
## 4    6.11 ctrl
## 5    4.50 ctrl
## 6    4.61 ctrl
## 7    5.17 ctrl
## 8    4.53 ctrl
```

```
## 9    5.33 ctrl
## 10   5.14 ctrl
## 11   4.81 trt1
## 12   4.17 trt1
## 13   4.41 trt1
## 14   3.59 trt1
## 15   5.87 trt1
## 16   3.83 trt1
## 17   6.03 trt1
## 18   4.89 trt1
## 19   4.32 trt1
## 20   4.69 trt1
## 21   6.31 trt2
## 22   5.12 trt2
## 23   5.54 trt2
## 24   5.50 trt2
## 25   5.37 trt2
## 26   5.29 trt2
## 27   4.92 trt2
## 28   6.15 trt2
## 29   5.80 trt2
## 30   5.26 trt2
```

5.1 数据集的合并

要横向合并两个数据框（数据集），可以使用merge()函数。在多数情况下，两个数据框是通过一个或多个共有变量进行联结的（inner join）。例如：

newdata < −merge(dataframeA, dataframeB, by= “ID”)

将dataframeA和dataframeB按照ID进行了合并。

若要直接横向合并两个矩阵或数据框，可以直接使用cbind() 函数，为了让它正常工作，每个对象必须拥有相同的行数，且要以相同的顺序排列。

total < − cbind(A, B)

5.2 数据集的抽取

5.2.1 保留变量

从一个大数据集中选择有限数量的变量来创建一个新的数据集。

```
# newdata<- dataframe[, c(n:m)]
```

从dataframe数据框中选择了第n到第m个变量，将行下标留空（,）表示默认选择所有行，并将它们保存到了新的数据框newdata中。

```
# vars <- c("var1", "var2", "var3", "var4" )
#  newdata <-dataframe[vars]
```

5.2.2 剔除变量

可以使用如下语句：

```
 # vars <- names(dataframe) %in% c("var1", "var2")
# x %in% table : 返回的是x中的每个元素是否在table中，是一个bool向量
#  newdata <-dataframe[!vars]
```

在知道要删除的变量是第几个变量的情况下，可以使用语句：

```
 # newdata <- dataframe[c(-n, -m)]  #删除第n和第m个变量
 # leadership£var1 <- leadership£var2 <- NULL
```

5.2.3 选择观测值

```
newdata1 <- sleep[1:20, ]  #选择前20个观测
 attach(sleep)
 newdata2 <- sleep[which(Sleep>=3 & Sleep <6),]
#选择睡眠时间在3~6小时之间的观测
 detach(sleep)
```

5.2.4 subset()函数

```
newdata3 <- subset(sleep, sleep>=3 & sleep<6, select=c(BodyWgt,Dream, Sleep,Span,Pred
newdata4<- subset(sleep, Pred=3 & sleep<6, select=BodyWgt:  Pred)
```

在第一个示例中，选择了所有睡眠时间在3 6小时之间的观测，且保留了BodyWgt, Dream, Sleep, Span, Pred, Exp, Danger变量。第二个示例选择了Pred为3且睡眠时间小于6小时的观测，且保留了变量BodyWgt到Pred之间的所有列。

参考文献

[1] 方匡南,朱建平,姜叶飞.R数据分析——方法与案例详解（双色）.电子工业出版社,201502.

[2] 王斌会.多元统计分析及R语言建模.暨南大学出版社.201405.