R语言在CGED-Q JSL中的运用

陈俊
香港科技大学社会科学部
2025年5月(V2.1.1)

Three

逻辑表达式和字符串处理
的相关函数

01

创建新变量

3.1.1 基础运算符

- 下表是R语言和Rstudio的基础运算符

R语言中常用的运算符

基础运算符	含义_1	判定运算符	含义_2	逻辑运算符	含义_3
+	加	<	小于	==	严格等于
-	减	>	大于	!=	不等于
*	乘	<=	小于等于	|	或者
/	除	>=	大于等于	&	和

- 在分析过程中，比较常用的是`==`、`!`、`|`、`&`,这些运算符被称之为逻辑运算符

3.1.2 删除数据库中的缺失值

- 数据库中存在缺失值是正常的现象，缺失值的存在会影响分析结果。
- 因此，在分析之前，我们需要剔除掉部分缺失值，在缙绅录数据库中，缺失值存在的形式是官员的名为`空白`

3.1.3 删除缺失值的步骤

```
kongbaiming <- which(JSL1900_1912$名=="空白")
```

- “komgbaiming”
- “<-”
- “which()”
- “JSL1900_1912”
- “$”
- “名”
- “==”
- “空白”

- 表示新变量的名称
- 表示等于
- which()函数能够返回指定值在逻辑向量中的位置，利用which函数指定条件，R就能够返回满足条件的记录所在的行数
- 所用数据集的名称
- 引用符号，表示引用该数据集的某一个变量
- 所引用的变量的名称
- 表示严格等于
- 判定条件，如果是字符，要加上引号，如果是数值则不用

```
JSL1900_1912_clean <- JSL1900_1912[-kongbaiming,]
```

- “JSL1900_1912_clean”
- “JSL1900_1912[-kongbaiming,]”

- 新数据集的名称
- 原数据集删除掉名是“空白”的记录

注意事项

- `JSL1900_1912_clean`是一个新的数据库的名称，为了便于理解，我们在取名时用了完整的名称，导致整个数据库的名字比较长
- 使用者可根据自身需要简化新数据库的名称
- 剔除缺失值是一个基础的环节，比较重要，所以我们放到创建新变量之前来讲
- 之后我们的数据分析基本上都会在已经剔除缺失值的`JSL1900_1912_clean`中进行
- 当然，此处剔除缺失值我们仅举了一个例子，即名是`空白`的记录条数，如果使用者仔细观察缙绅录数据库，会发现还有一些变量上记载的是`涂黑`或 `塗黑`，这也是缺失值表现形式的一种
- 使用者可以利用以上方法剔除掉其他的缺失值，提高数据分析的准确性

3.1.4 转换变量的类型

- 目前大多数历史数据库都是字符串类型
- 即便有一些记录看起来是数值，但其存储方式仍然可能是字符串
- 因此，在进行变量之间的运算之前，需要将字符串类型的数据库转换为数值型，不然运算时RStudio会报错
- 将字符串类型转换为数值类型的函数是`as.numeric()`函数，这一函数的格式为：

```
`data$x <- as.numeric(data$x)`
```

- 在该函数格式中，`data`为数据库或数据集，`$`为引用符号，这条代码的意思是将`data`中的`x`变量转化为数值类型。下面，我们根据这一代码的格式编写一个可以转换缙绅录数据库变量类型的实例代码

3.1.5 转换变量的代码示例

```
# 将`JSL1900_1912_clean`数据库中的变量“阳历年份”，
# 利用`as.numeric()`函数转换变量类型，进而生成一个新变量“阳历年份numeric”
JSL1900_1912_clean$阳历年份numeric <- as.numeric(JSL1900_1912_clean$阳历年份)

# 利用`typeof()`函数返回变量的属性，检验变量类型转换是否成功
typeof(JSL1900_1912_clean$阳历年份numeric)

# 将“阳历年份”变量换成了“季节号”
# “季节号”转化为数值类型后，1代表春季、2代表夏季、3代表秋季、4代表冬季
JSL1900_1912_clean$季节号numeric <- as.numeric(JSL1900_1912_clean$季节号)

# 利用`typeof()`函数返回变量的属性，检验变量类型转换是否成功
typeof(JSL1900_1912_clean$季节号numeric)
```

3.1.6 创建新变量

- 在完成了剔除缺失值、转换变量类型的工作后，我们可以在数据库中随意建立两个以上变量的联系，这种联系通过添加运算符实现

- 我们现在需要在缙绅录数据库中建立一个新变量“年份季节”，用以展示缙绅录数据库中各版本的年代、季节，可以通过以下代码实现：

```
JSL1900_1912_clean$年份季节 <- ( JSL1900_1912_clean$阳历年份numeric +
( ( JSL1900_1912_clean$季节号numeric/4 )-0.25) )
```

3.1.7 代码思路

- 代码的逻辑为：

```
`data$x <- (data$y+((data$z/4)-0.25))`
```

- 在`JSL1900_1912_clean`数据库中，用“阳历年份numeric”的数值加上“季号份numeric”数值的1/4，再减去0.25，最后生成新的数值来表达缙绅录版本的年代和季节

- 比如，1906年夏季的缙绅录，它的“年份季节”便是`（1906+2/4）-0.25`，即是1906.25

- 那为什么要在最后减去0.25，而不是直接用“阳历年份”加上“季节号”的1/4呢？

- 因为如果不减去0.25，数值进位会让冬季版本被归类为次年春季版本

- 比如，1907年冬季的版本，如果不减去0.25，就变成`1907+4/4`，即1908.00，而1908.00是表示1908年春季的版本

3.1.8 查看新变量

- 利用`table()`函数查看变量

```
table(JSL1900_1912_clean$年份季节)
# `table()`函数能够返回某个变量的记录数。
# 运行这条代码后，RStudio会在区域三显示出每个年份季节的记录数
```

- 在创建新变量的过程中，我们利用运算符建立了两个变量之间的联系，这也是量化的一种思路

- 当然，我们所利用的两个变量，其本身就是数字，所以很容易利用运算符建立联系

- 当两个变量是字符型变量时，它们便不适合进行运算，这时就需要利用到其他函数创建联系，这个我们在后文会讲到

02

逻辑表达式

3.2.1 历史数据库中的逻辑判定

- 逻辑表达式是利用逻辑运算符和关系表达式所编写的代码逻辑表达式所产生的是一个逻辑值，主要用于逻辑判定

- 在历史数据库的分析中，我们经常会用到逻辑表达式来判定一个记录的类型

- 比如，在历史人口数据库中，判定一个人是否拥有孩子就可以用到逻辑表达式，当某个人的记录上，后代数量这一栏中只要标注了非0的数量，无论这个数字有多大，其是否拥有孩子这一栏上都会标注上是；反之，则会标注上否。可见，逻辑表达式在历史数据库中运用得十分广泛。

3.2.2 `ifelse()`函数

- 下面我们介绍一个简单的逻辑表达式`ifelse()`函数:

```
`ifelse(test,yes,no)`
```

- `test`即为R需要执行的条件，通常是某个变量大于小于或等于某个值，简单来说就是一个判定条件。如果满足这个判定条件，RStudio则返回`yes`，`yes`可以更改为`True`或者`1`等等。如果不满足判定条件，则返回`no`，`no`可以替换为`False`或者`0`等等

3.2.3 `ifelse()`函数判定JSL的版本年代

- 比如，在缙绅录数据库中，我们编写一个代码来判定缙绅录的版本年份是否在1900年之后：

```
ifelse(年份<1900,T,F)
```

3.2.4 `ifelse()`函数判定官员民族属性

- 在清代官员的研究中，有一个比较重要的议题是官员的满汉比例。
- 我们可以通过`ifelse()`函数来判定一个官员是否为“旗人”，进而深入研究清代官员满汉比例的变化

3.2.5 `ifelse()`函数判定官员是否为旗人

- 具体代码如下：

```
JSL1900_1912_clean$qiren <- ifelse(( JSL1900_1912_clean$旗分!="" |
JSL1900_1912_clean$身份二!="" | JSL1900_1912_clean$姓=="" ),1,0)
```

- 该代码结合了缙绅录数据库中的三个变量来判定官员的民族属性
  - 第一，“旗分”，部分旗人的旗分有记载，比如“汉军正蓝”等等
  - 第二，“身份二”，少部分旗人有身份标识，如宗室、觉罗等等
  - 第三，“姓”，缙绅录数据库中，基本上所有旗人都没有姓氏记载

- 三个关系表达式之间用`|`（或者）连接，三个条件任意满足其中一个即可判定为`1`（是），反之则返回`0`（否）

3.2.6 查看判定旗人的变量

- 利用`table()`函数查看变量

```
# 若`qiren`变量等于1的话，就改变其值为“旗”
JSL1900_1912_clean$qiren[JSL1900_1912_clean$qiren == 1] <- "旗"

# 若`qiren`变量等于0的话，就改变其值为“民”
JSL1900_1912_clean$qiren[JSL1900_1912_clean$qiren == 0] <- "民"

# 在区域三返回逻辑值是`1`（旗人）和`0`（民人）的记录数
table(JSL1900_1912_clean$qiren)
```

- 区域三返回：

```
 旗     民
154873 484559
```

03

转换变量

3.3.1 将字符型变量转换为数值型变量

- 前面我们介绍了一种方法，利用`as.numeric()`函数将字符型变量转换为数值型
- 但这种方法的实质是改变变量的形式，让其能够参与计算
- 另外，这是一种纵向的处理方式，即让该纵列变量下的所有记录全部转换为数值类型
- 那么，如何让指定的字符串记录转换为数值型呢？以下代码是一个例子

```
# 将缙绅录数据库中“出身一”变量为“狀元”的记录，转换为数值1
JSL1900_1912_clean$出身一[JSL1900_1912_clean$出身一 == "狀元" ] <- 1
# 将“出身一”变量为“榜眼”的记录，转换为数值2
JSL1900_1912_clean$出身一[JSL1900_1912_clean$出身一 == "榜眼" ] <- 2
# 将“出身一”变量为1的记录，转换为数值“狀元”
JSL1900_1912_clean$出身一[JSL1900_1912_clean$出身一 == 1 ] <- "狀元"
# 将“出身一”变量为2的记录，转换为数值“榜眼”
JSL1900_1912_clean$出身一[JSL1900_1912_clean$出身一 == 2 ] <- "榜眼"
```

- 将“狀元”变量转化为`1`的作用在于：如果要建立一个官员升迁的模型，需要展示出身对官员升迁的影响，那转换后的数值`1`则比原记录“狀元”更加适合引入到量化模型中

3.3.2 将数值型变量转换为字符型变量

- 同样的逻辑，将数值型变量转换为字符型，代码示例如下：

```
# 将“季节”变量中的数值转换为字符，
# `1`转变为春、`2`转变为夏、`3`转变为秋、`4`转变为冬
JSL1900_1912_clean$季节[JSL1900_1912_clean$季节 == 1 ] <- "春"
JSL1900_1912_clean$季节[JSL1900_1912_clean$季节 == 2 ] <- "夏"
JSL1900_1912_clean$季节[JSL1900_1912_clean$季节 == 3 ] <- "秋"
JSL1900_1912_clean$季节[JSL1900_1912_clean$季节 == 4 ] <- "冬"
```

- 利用`table()`函数查看春、夏、秋、冬的数量

```
table(JSL1900_1912_clean$季节)
```

3.3.3 将离散型变量转换为数值型变量—制作数据框

- 可以看出，前两种转换变量的方式都有很大的相似之处，代码的格式也基本是同一种类型。这是因为缙绅录的数值都是连续型变量，具有规律性。但当一个数据集是离散型变量，该如何转换？

- 下面是来自NBA官网的数据，统计一些球员在职业生涯中的各项数据，利用`data.frame()`函数制作数据框

```
NBA <- data.frame(
  Player = c("Kobe", "Jordan", "James", "Garnett", "Duncan", "Kidd", "Anthony",
"Pippen", "Curry"),
  Points = c(33643, 32292, 34087, 26071, 26496, 17529, 26314, 18940, 16419),
  Rebounds = c(7074, 6672, 9352, 14662, 15091, 8725, 7251, 7494, 3158),
  Assists = c(6306, 5633, 9298, 5445, 4225, 12091, 3244, 6135, 4621),
  Games = c(1346, 1072, 1258, 1462, 1392, 1391, 1114, 1178, 699),
  Points_per_game = c(25.0, 30.1, 27.1, 17.8, 19.0, 12.6, 23.6, 16.1, 23.5))
```

3.3.4 根据变量分类-1

- 我们根据球员的生涯总得分将球员进行分类，生涯总得分突破1万分的球员被归类为“一万分先生”，突破2万分被归类为“两万分先生”，突破3万分被归类为“三万分先生”。可通过如下代码实现

```
NBA$Points_origin <- NBA$Points
# 保留原始数据列

NBA$Points[NBA$Points >= 30000] <- 3
NBA$Points[NBA$Points >= 20000 & NBA$Points < 30000] <- 2
NBA$Points[NBA$Points >= 10000 & NBA$Points < 20000] <- 1
NBA$Points <- factor(NBA$Points, levels = c(1,2,3), labels = c("一万分先生","两万分先生","三万分先生"))
```

3.3.5 根据变量分类-2

- 同理，根据球员的生涯总助攻数将球员进行分类，生涯总助攻突破5000的球员被归类为“助攻专家”，突破9000被归类为“助攻机器”，突破15000被归类为“助攻王”。可通过如下代码实现：

```
NBA$Assists_origin <- NBA$Assists
# 保留原始数据列

NBA$Assists[NBA$Assists <= 5000] <- 1
NBA$Assists[NBA$Assists > 5000 & NBA$Assists <= 9000] <- 2
NBA$Assists[NBA$Assists > 9000 & NBA$Assists <= 15000] <- 3
NBA$Assists <- factor(NBA$Assists, levels = c(1,2,3), labels = c("助攻专家","助攻机器","助攻王"))
```

3.3.6 `factor()`函数

- 离散型变量转变为分类字符变量，其核心思想是首先转变将数值分类，再转换为因子，最后将因子转换为分类字符变量。`factor()`函数作为一个转换因子的函数，在其中起到了中转站的作用，其格式为：

```
`factor(x, levels, labels)`
```

- `x`代表需要定义的变量，`levels`代表值，`labels`代表标签

3.3.7 将已经归类好的字符型变量转换回数值型变量

```
NBA$Points_numerical <- factor(NBA$Points, levels = c("一万分先生","两万分先生","三万分先生"), labels = c(1,2,3))
```

3.3.8 导出表格

- 利用`flextable`包导出表格

```
library(flextable)
library(dplyr)
NBA_ft <- flextable(NBA)
NBA_ft <- flextable(NBA) %>% add_header_row( values = "NBA球星核心数据统计表",
colwidths = ncol_keys(NBA_ft), # 将表头设置为标题，确保表格导出时标题可以被保留
top = TRUE) %>%  # 添加到表格顶部
  bold(i = 1, part = "header") %>%  # 第一行表头加粗
  fontsize(i = 1, size = 14, part = "header") %>%  # 表头增大字号
  bold(part = "header")  %>%  # 表头加粗
  border(i = 1, part = "header",
         border.top = fp_border(width = 0, color = "transparent")) %>%  # 移除第一行表头的上边框
  autofit() %>%
  add_footer_lines("数据来源:< http://www.stat-nba.com/ > 截至2023年") %>%
  align(align = "center", part = "all")

# 保存表格
save_as_image(NBA_ft, path = "/Users/jc/Documents/R export/image/NBA_ft.png")
```

NBA球星核心数据统计表

Player	Points	Rebounds	Assists	Games	Points_per_game	Points_origin	Points_numerical	Assists_origin
Kobe	三万分先生	7,074	助攻机器	1,346	25.0	33,643	3	6,306
Jordan	三万分先生	6,672	助攻机器	1,072	30.1	32,292	3	5,633
James	三万分先生	9,352	助攻王	1,258	27.1	34,087	3	9,298
Garnett	两万分先生	14,662	助攻机器	1,462	17.8	26,071	2	5,445
Duncan	两万分先生	15,091	助攻专家	1,392	19.0	26,496	2	4,225
Kidd	一万分先生	8,725	助攻王	1,391	12.6	17,529	1	12,091
Anthony	两万分先生	7,251	助攻专家	1,114	23.6	26,314	2	3,244
Pippen	一万分先生	7,494	助攻机器	1,178	16.1	18,940	1	6,135
Curry	一万分先生	3,158	助攻专家	699	23.5	16,419	1	4,621

数据来源：< http://www.stat-nba.com/ > 截至2023年的数据

04

字符串处理的相关函数

3.4.1 串联字符

`paste()`函数

- 串联字符的代码为`paste()`函数，其格式

```
`paste("x", "y", sep=,collapse=)`
```

- `x`代表需要串联的一个变量，`y`代表需要串联的另一个变量，`sep`代表串联完成后字符内的连接符，`collapse`代表字符串间的连接符，一般用令它等于默认值`NULL`。

串联姓名

- 现在我们将缙绅录数据库中官员的姓和名连接起来（缙绅录数据库照录原文，两者在数据库中是分开的），代码示例如下：

```
JSL1900_1912_clean$xingming <-paste(JSL1900_1912_clean$姓,
JSL1900_1912_clean$名, sep="",collapse=NULL)
```

- 除了姓和名之外，使用者还可以尝试将官员的“籍贯省”和“籍贯县”连接起来，比如“湖南”与“湘乡”连接为“湖南湘乡”，因代码逻辑一致，此处就不一一列举

3.4.2 提取和判断字符

提取函数`str_extract()`

- R中的提取函数为`str_extract()`函数，用以提取需要但不能确定位置的记录。格式为：

```
`str_detect(x, "y")`
```

- `x`代表变量，`y`代表需要提取的字符

提取“员外郎”

- 现在，用该函数判断官员的官职是否为“員外郎”，实例代码：

```
install.packages("stringr")
library(stringr)
str_extract(JSL1900_1912_clean$官职一 ,"員外郎")
```

判断函数`str_detect()`

- R中的提取函数为`str_extract()`函数，用以提取需要但不能确定位置的记录。格式为：

```
`str_detect(x, "y")`
```

- `x`代表变量，`y`代表需要判断的字符
- 提取字符函数的返回值只有需要提取的值和NA，NA不便处理；而判断字符函数返回值是`ture`和`false`，是逻辑值。因此，在寻找特定字符时，常用判断函数

判定“总督”群体

- 熟悉`str_detect()`函数之后，就可以利用该函数结合`ifelse()`函数进行官职查找和判定
- 比如查找数据库中的“总督”群体，先利用`str_detect()`函数判断官职是否为“总督”
- 随后，给满足“总督”条件的记录赋值1，否则赋值0
- 这样，便可以对总督这一群体进行单独研究
- 实例代码如下：

```
JSL1900_1912_clean$zongdu <- ifelse( str_detect(JSL1900_1912_clean$官职一,"總督"),1,0)
```

判定“知县”、“候补官员”、“额外官员”群体

- 同理，寻找出“知县”群体、“候补官员”群体、“额外官员”群体：

```
JSL1900_1912_clean$zhixian <- ifelse( str_detect(JSL1900_1912_clean$官职一,"知縣"),1,0)
JSL1900_1912_clean$houbu <- ifelse( str_detect(JSL1900_1912_clean$官职一,"候補"),1,0)
JSL1900_1912_clean$ewai <- ifelse( str_detect(JSL1900_1912_clean$官职一,"額外"),1,0)
```

- 运行以上代码后，我们就可以在区域一看到数据库增加了`zongdu`、`zhixian`、`houbu`、`ewai`四个变量

- 纵向变量下只有1和0的逻辑值，显示为`1`则表明某官员的官职为对应变量名称，比如，一官员的`zhixian`变量为1，那表明该官员官职为知县

- 锁定了特定官员群体之后，便可进行更深入的分析

3.4.3 替换字符 `gsub()`函数

- 替换函数gsub()函数，用于替换字符，其格式为：

```
`gsub("y", "z", x)`
```

- 其中，`y`代表需要替换的字符，`z`代表替换成什么，`x`代表变量

替换JSL数据库中的“鑲藍”为“镶蓝”

- 替换函数gsub()函数，用于替换字符，其格式为：

```
gsub("鑲藍","镶蓝", JSL1900_1912_clean$旗分)
```

- 替换字符主要用于替换数据库中的异体字
- 因字符串数据库中繁体字、异体字、简体字互不相通，所以需要用替换函数进行统一
- 使用者可根据研究需要仿照代码示例替换掉字符中的异体字，总体逻辑一致，此处不一一列举

Thanks!