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)

Seven

数据集的内外连接

01

数据集内部记录的连接

7.1 数据集内部记录的连接

- 在古代官员研究领域，官员仕途是一个热点话题
- 目前比较有专门针对清代官员仕途、履历的档案汇编《清代官员履历档案全编》，这套书1997年由华东师范大学出版社出版。但这套书具有工具书的性质，需要利用人名检索，操作起来颇为复杂
- 缙绅录数据库其实也有研究官员仕途的潜力，通过对官员个人信息进行识别，锁定官员并赋予其独一无二的个人编号，通过检索个人编号追踪官员的任职记录，锁定官员任职记录并在《清代官员履历档案全编》中查找相关官员的履历，可以完整而清晰地展现官员的仕途
- 以上通过创造官员个人编号追踪其任职记录的方法称之为数据集的内连接。

7.1.1 创建官员个人编号的核心函数 `arrange()`函数

- 在创建官员个人编号过程中，会用到一些核心的函数，分别是
  - `arrange()`
  - `group_by()`
  - `mutate()`
  - `cumsum()`

- 它们的基本语法为：

```
arrange(.data, ...,by_group)
# 排序函数
# .data为数据集，...为指定的条件（选择列为条件），`by_group`表示是否首先按照分组的变量进行排序，可选TRUE或者FALSE
```

7.1.2 `group_by()`函数

- 分组函数

```
group_by(.data, ...)
```

- .data为数据集，...为指定的条件

7.1.3 `mutate()`函数

- 变量生成、转换以及删除函数

```
mutate(.data, ...)
```

- .data为数据集，...要创建或修改的列。
- `mutate()`函数是`dplyr`包下的核心函数之一，使用者可以通过在控制台键入`?mutate`获取该函数的所有参数和用法

7.1.4 `cumsum()`函数

- 累加函数

```
cumsum(data$var)
```

- data为数据集，var为变量

7.1.5 创建官员个人编号的步骤

- 第一步，创建分析所用数据集。读取`tidyverse`包，利用`subset()`函数创建一个数据集，只提取1906年的数据

- 第二步，创建一个升序变量。“RecordNumber”是一个新升序变量，`1:nrow`函数的意思是按照行的顺序生成一个从1到n的序号变量

- 第三步，按数据库中的特定变量进行排序。

- 排序函数为`arrange()`，它可以将行按指定列的顺序来排序。在接下来的代码中，`xingming…RecordNumber`是排序的条件，这里我们选择的首要条件是`xingming`，然后R会根据姓名进行排序，如果有多条件，那么R会根据条件的先后进行排序，即先考虑`xingming`，再考虑身份二。把身份二和旗分放在第二和第三是因为有很多旗人是同名的，所以我们利用身份二和旗分变量进行二次甄别。同时不考虑`by_group`，令其等于默认，即FALSE。

7.1.6 创建官员个人编号的步骤

- 第四步，按数据库中的特定变量进行分组。
- 分组函数为`group_by()`，可以将行按指定条件进行分组。按照`group_by()`函数的格式，编写如下代码创建一个`PersonID`（个人编号）的新变量
- 第五步，对升序变量进行累加。`cumsum()`函数，可以对指定行、分组求累加值
- 分组和赋值完成后，`PersonID`里只有逻辑值，这里需要用到`cunsum()`累加函数给每一个`PersonID`一个独一无二的编号

7.1.7 创建官员个人编号的代码

```
library(tidyverse)
JSL1906 <- subset(JSL1900_1912_clean,阳历年份 == "1906")
JSL1906$RecordNumber <- 1:nrow(JSL1906)
JSL1906 <- arrange(JSL1906,xingming,身份二,旗分,出身一,年份季节,RecordNumber)
JSL1906 <- JSL1906 %>%
  group_by(xingming,旗分) %>%
  mutate(PersonID = ifelse(row_number() == 1, 1, 0))
# 这是一个复合函数，利用管道符进行连接。此复合函数分为三步
# 首先，将数据集按照"xingming"和"旗分"两个变量的顺序进行分组，
# 分组是确定不同官员编号的关键；
# 然后，利用`mutate()`函数添加新的变量；
# 最后，`mutate()`里用到了嵌套函数`ifelse()`，
# 如果某一行它是该分组的第一行，那么R给它赋值1，否则为0

JSL1906$PersonID_cumsum <- cumsum(JSL1906$PersonID)
```

02

利用官员个人编号分析官员任职年限

7.2 利用官员个人编号分析官员任职年限

- 我们利用官员个人编号分析官员的任职年限，并检验前述代码的效果。代码如下：

```
# 基于管道符的组合代码函数
JSL1906 <- JSL1906 %>%
  group_by(PersonID_cumsum) %>%
  mutate(YearsServed = abs((年份季节-年份季节[1])+0.25))

# 制表检验
JSL1906_winter <- subset(JSL1906, 季节号 == 4)
library(table1)
JSL1906_winter$YearsServed[JSL1906_winter$YearsServed == 0.25] <- "0.25年"
JSL1906_winter$YearsServed[JSL1906_winter$YearsServed == 0.5] <- "0.5年"
JSL1906_winter$YearsServed[JSL1906_winter$YearsServed == 0.75] <- "0.75年"
JSL1906_winter$YearsServed[JSL1906_winter$YearsServed == 1] <- "1年”

table1(~jiguansheng_sort|YearsServed, data = JSL1906_winter, overall = "total")
```

	0.25年 (N=776)	0.5年 (N=610)	0.75年 (N=1566)	1年 (N=8889)	total (N=11841)
籍贯省					
	324 (41.8%)	217 (35.6%)	308 (19.7%)	4465 (50.2%)	5314 (44.9%)
不详	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
外国	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
安徽	35 (4.5%)	30 (4.9%)	80 (5.1%)	351 (3.9%)	496 (4.2%)
奉天	7 (0.9%)	1 (0.2%)	14 (0.9%)	47 (0.5%)	69 (0.6%)
福建	10 (1.3%)	15 (2.5%)	68 (4.3%)	98 (1.1%)	191 (1.6%)
甘肃	1 (0.1%)	6 (1.0%)	22 (1.4%)	31 (0.3%)	60 (0.5%)
广西	6 (0.8%)	13 (2.1%)	41 (2.6%)	59 (0.7%)	119 (1.0%)
广东	16 (2.1%)	11 (1.8%)	50 (3.2%)	121 (1.4%)	198 (1.7%)
贵州	15 (1.9%)	11 (1.8%)	43 (2.7%)	94 (1.1%)	163 (1.4%)
河南	9 (1.2%)	13 (2.1%)	81 (5.2%)	155 (1.7%)	258 (2.2%)
黑龙江	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
湖北	14 (1.8%)	16 (2.6%)	58 (3.7%)	206 (2.3%)	294 (2.5%)
湖南	52 (6.7%)	26 (4.3%)	51 (3.3%)	330 (3.7%)	459 (3.9%)
吉林	2 (0.3%)	1 (0.2%)	1 (0.1%)	0 (0%)	4 (0.0%)
江南	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
江苏	41 (5.3%)	38 (6.2%)	141 (9.0%)	496 (5.6%)	716 (6.0%)
江西	22 (2.8%)	17 (2.8%)	82 (5.2%)	218 (2.5%)	339 (2.9%)
山东	24 (3.1%)	25 (4.1%)	90 (5.7%)	207 (2.3%)	346 (2.9%)
山西	11 (1.4%)	18 (3.0%)	35 (2.2%)	65 (0.7%)	129 (1.1%)
陕西	4 (0.5%)	9 (1.5%)	40 (2.6%)	77 (0.9%)	130 (1.1%)
顺天	32 (4.1%)	24 (3.9%)	57 (3.6%)	544 (6.1%)	657 (5.5%)
四川	45 (5.8%)	27 (4.4%)	51 (3.3%)	308 (3.5%)	431 (3.6%)
天津	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
新疆	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
云南	7 (0.9%)	7 (1.1%)	33 (2.1%)	66 (0.7%)	113 (1.0%)
浙江	62 (8.0%)	59 (9.7%)	116 (7.4%)	734 (8.3%)	971 (8.2%)
直隶	37 (4.8%)	26 (4.3%)	104 (6.6%)	217 (2.4%)	384 (3.2%)

图像说明

- 从上图我们可以看出，在籍贯省任职的官员任期稳定性最高
- 同时，在浙江、顺天、江苏任职的官员任期稳定性比较高，在所有任职达到一年的官员中（因为我们只取了1年的数据），浙江占比最高，为8.3%。顺天占比第二，为6.1%
- 当然，我们这里只用了一年的数据，在很大层面上会引起歧义。所以，我们建议大家在更大的年限跨度上去做尝试。

03

如何统计人数而非记载数

7.3 如何统计人数而非记载数
7.3.1 保留官员第一条任职记录

- 我们还可以利用`PersonID`将统计模式从记载数转变为人数
- 用人数来统计有什么好处？假如一个数据集有50w万条记录，但可能同一个人就有10条记录，所以整个数据集可能只有5万人
- 当我们用记载数来统计整个数据库的时候，有很多项目都会出现重复记录的情况，因为一个官员出现了多次
- 当我们用人数来做统计对象时，满汉比例、出身构成、官员社会和地理来源都会变得非常精确，不会出现重复计算的情况
- 代码示例如下：

```
# 此处用到`filter()`函数，筛选出PersonID == 1的行。
# 利用"PersonID"来删除同一个人的多条记录，
# 只保留第一条记录，这样就能保证留下来的所有记录是每个人的唯一一条记录
JSL1906_人数 <- JSL1906 %>% filter( PersonID == 1 )
```

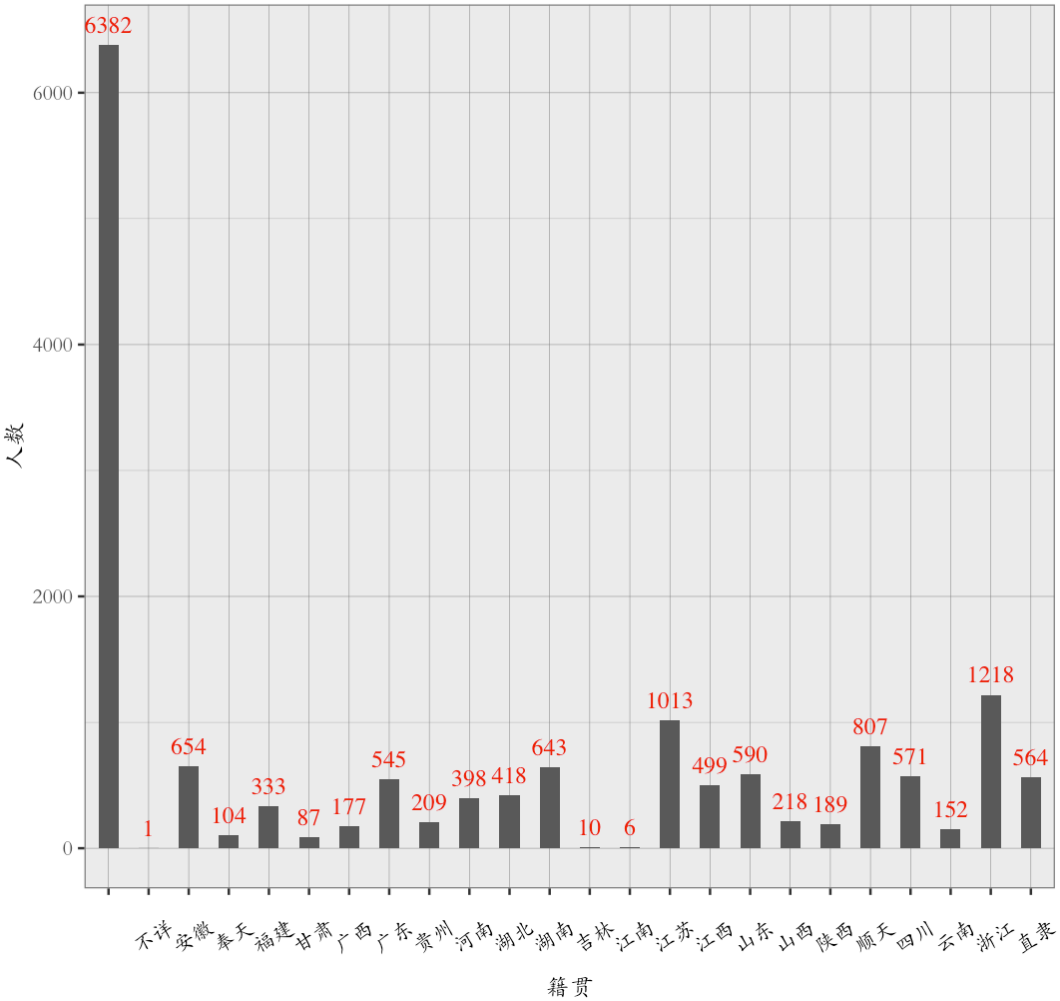
制图检验

```
# 已省略`theme()`函数

library(ggplot2)
P_BarChart_Jiguansheng <- ggplot(JSL1906_人数,aes(jiguansheng_sort)) +
  geom_bar(just= 0.5,width = 0.5)+
  labs(x = "籍贯", y = "人数",title = "1906年文官的地理来源（人数）")+
 geom_text(aes(label = ..count..,family = "serif"),
            stat = "count",
            size = 3.5,
            vjust = -0.8,
            hjust = 0.5,
            color = "red2")

P_BarChart_Jiguansheng # 显示图形
```

1906年文官的地理来源（人数）



图像说明

- 原始的数据集`JSL1906`大约有5万条记录，而经过删除个人重复记录后的数据集`JSL1906_人数`则约有1.5万条记录，展示了1.5万名不同官员的个人信息，如果不删除重复记录而直接对记载数做统计的话会导致结果精确性大打折扣

- 从上图中可以看出，1906年文官大多数都是`qiren`（因旗人籍贯省记载为"空白"）和籍贯为京师的人（在本籍地任职的官员，其籍贯省也为空白），这表现出清代选官“首崇满洲”的思想

- 除此之外，文官多来自浙江、江苏、广东等沿海城市，这也表明在文化发达的沿海省份，产生行政管理者的概率更高

- 利用人数可以从整个大数据集出发考察具体真实的满汉比例，出身构成和官员社会地理来源

7.3.2 保留官员最后一条任职记录

- 在部分情况下保留官员的第一条任职记录和最后一条记录几乎没有太大的差别，但是官员任职过程中可能更换字号和出身信息，也有可能随着官职的迁转其铨选方式也会发生改变，这些情况下保留官员最后一条任职记录再进行分析的准确性会更高一点

- 同时，由于创建官员个人编号时使用了组内排序方法，故研究官员的任期也需要保留最后一条任职记录

- 保留官员最后一条任职记录的代码如下：

```
JSL1906_last_record <- JSL1906 %>% group_by( xingming ) %>% filter( row_number() == n() )
# 只保留官员的最后一次任职记录
# 可以求得官员职业生涯总的任职年限
```

制表检验结果

```
table1(~jiguansheng_sort|YearsServed,
       data = JSL1906_last_record,
       overall = "total")
#通过制表来检验保留官员最后一次任职记录的效果
```

	0 (N=273)	0.25 (N=3507)	0.5 (N=1284)	0.75 (N=2152)	1 (N=7990)	total (N=15206)
籍贯省						
	93 (34.1%)	911 (26.0%)	405 (31.5%)	529 (24.6%)	3900 (48.8%)	5838 (38.4%)
不详	0 (0%)	1 (0.0%)	0 (0%)	0 (0%)	0 (0%)	1 (0.0%)
外国	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
安徽	10 (3.7%)	148 (4.2%)	74 (5.8%)	103 (4.8%)	313 (3.9%)	648 (4.3%)
奉天	3 (1.1%)	32 (0.9%)	8 (0.6%)	19 (0.9%)	44 (0.6%)	106 (0.7%)
福建	12 (4.4%)	116 (3.3%)	28 (2.2%)	78 (3.6%)	94 (1.2%)	328 (2.2%)
甘肃	3 (1.1%)	16 (0.5%)	13 (1.0%)	26 (1.2%)	28 (0.4%)	86 (0.6%)
广西	5 (1.8%)	44 (1.3%)	26 (2.0%)	48 (2.2%)	52 (0.7%)	175 (1.2%)
广东	8 (2.9%)	314 (9.0%)	42 (3.3%)	62 (2.9%)	115 (1.4%)	541 (3.6%)
贵州	2 (0.7%)	42 (1.2%)	25 (1.9%)	48 (2.2%)	91 (1.1%)	208 (1.4%)
河南	8 (2.9%)	120 (3.4%)	32 (2.5%)	90 (4.2%)	148 (1.9%)	398 (2.6%)
黑龙江	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
湖北	8 (2.9%)	104 (3.0%)	37 (2.9%)	80 (3.7%)	184 (2.3%)	413 (2.7%)
湖南	23 (8.4%)	165 (4.7%)	68 (5.3%)	109 (5.1%)	278 (3.5%)	643 (4.2%)
吉林	0 (0%)	7 (0.2%)	2 (0.2%)	1 (0.0%)	0 (0%)	10 (0.1%)
江南	0 (0%)	3 (0.1%)	1 (0.1%)	0 (0%)	0 (0%)	4 (0.0%)
江苏	11 (4.0%)	260 (7.4%)	95 (7.4%)	179 (8.3%)	469 (5.9%)	1014 (6.7%)
江西	7 (2.6%)	141 (4.0%)	46 (3.6%)	103 (4.8%)	200 (2.5%)	497 (3.3%)
山东	14 (5.1%)	225 (6.4%)	49 (3.8%)	106 (4.9%)	194 (2.4%)	588 (3.9%)
山西	6 (2.2%)	89 (2.5%)	27 (2.1%)	39 (1.8%)	62 (0.8%)	223 (1.5%)
陕西	6 (2.2%)	39 (1.1%)	29 (2.3%)	44 (2.0%)	72 (0.9%)	190 (1.2%)
顺天	9 (3.3%)	131 (3.7%)	57 (4.4%)	90 (4.2%)	510 (6.4%)	797 (5.2%)
四川	10 (3.7%)	149 (4.2%)	44 (3.4%)	68 (3.2%)	294 (3.7%)	565 (3.7%)
天津	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
新疆	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
云南	7 (2.6%)	32 (0.9%)	15 (1.2%)	45 (2.1%)	59 (0.7%)	158 (1.0%)
浙江	16 (5.9%)	239 (6.8%)	106 (8.3%)	169 (7.9%)	686 (8.6%)	1216 (8.0%)
直隶	12 (4.4%)	179 (5.1%)	55 (4.3%)	116 (5.4%)	197 (2.5%)	559 (3.7%)

注意事项

- 是否保留官员的最后一条任职记录需要示情况而定
- 如使用者不确定，可根据官员的第一条任职记录和最后一条任职记录生成不同数据集，进而进行分析和比较研究
- 考虑到缙绅录的编纂体例随着时间的推移而愈加完善，这意味着越后期的版本信息更加完整，因此我们推荐使用者在利用人数进行分析研究时保留官员的最后一条任职记录

04

连接两个数据集

7.4 连接两个数据集

7.4.1 数据集外连接的注意事项

- 在实际的数据集分析中，我们时常会遇到两个数据集相匹配的问题
- 我们前面介绍的数据集匹配`merge()`函数采用的是精准匹配方法，且我们所用到的数据集是从整个大的缙绅录数据库中截取出来的，两者的结构、变量设置都是一样的，所以匹配起来比较容易
- 但在实际的匹配时，我们时常会遇到两个不同类型、不同结构的数据库匹配，这种情况下，公共列的选取就比较困难
- 比如一个数据库是缙绅录，另一个数据库是人口数据库，人口数据库的内容是人的年龄、亲属数量、税务等等，这与缙绅录数据库的设置完全不一致
- 现在，我们需要将两个数据库连接起来，考察官员的亲属关系和个人生活，就只能利用到`xingming`这个变量去进行匹配，将两个数据库连接起来，再对数据进行慢慢甄别、检验。

7.4.2 以JSL为例进行外连接

- 现在我们仍然以缙绅录数据库为例，讲解数据库外连接的步骤
- 首先需要截取缙绅录数据库中1906年四川知县的数据、1910年四川知县的数据，以考察四年间四川知县的变动情况
- 请注意，如果我们要考察1906到1910年间四川地区知县是否一直在任的情况，我们需要统计的是人数而非记载数
- 因此，在准备数据阶段，我们不仅要截取指定年份的数据，还要创造`PersonID`变量，再保留官员的最后一条任职记录
- 如果不删除重复任职记录，那么一个数据会被连接多次，最后导致生成的数据库样本量远大于被连接的两个数据集

7.4.3 数据准备-1906四川知县

- 以下代码截取了1906年四川知县的数据，并创建了`PersonID`和保留最后一条任职记录，这一代码在前节(7.2.3)出现过，只是更换了数据集名称，所以不再一一解释。示例代码如下：

```
JSL1900_1912_clean %>%
  filter((阳历年份 == "1906" & 地区 == "四川省" & zhixian == "知县")) %>%
  select(阳历年份,地区,机构一,机构二,官职一,xingming,字号,籍贯省,籍贯县,身份二,旗分,出身一,年份季节) -> JSL1906_sichuan_zhixian
JSL1906_sichuan_zhixian$RecordNumber <- 1:nrow(JSL1906_sichuan_zhixian)
JSL1906_sichuan_zhixian <- arrange(JSL1906_sichuan_zhixian,xingming,身份二,旗分,出身一,年份季节,RecordNumber)
JSL1906_sichuan_zhixian <- JSL1906_sichuan_zhixian %>%
  group_by(xingming,旗分) %>%
  mutate(PersonID = ifelse(row_number() == 1, 1, 0))
JSL1906_sichuan_zhixian$PersonID_cumsum <- cumsum(JSL1906_sichuan_zhixian$PersonID)
JSL1906_sichuan_zhixian <- JSL1906_sichuan_zhixian %>%
  group_by(PersonID_cumsum) %>%
  mutate(YearsServed = abs((年份季节-年份季节[1])+0.25))
JSL1906_sichuan_zhixian <- JSL1906_sichuan_zhixian %>% group_by( xingming ) %>%
filter( row_number() == n() )
```

7.4.4 数据准备-1910四川知县

- 以下代码截取了1910年四川知县的数据，并创建了`PersonID`和保留最后一条任职记录，这一代码在前节(7.2.3)出现过，只是更换了数据集名称，所以不再一一解释。示例代码如下：

```
JSL1900_1912_clean %>%
  filter((阳历年份 == "1910" & 地区 == "四川省" & zhixian == "知县")) %>%
  select(阳历年份,地区,机构一,机构二,官职一,xingming,字号,籍贯省,籍贯县,身份二,旗分,出身一,年份季节) -> JSL1910_sichuan_zhixian
JSL1910_sichuan_zhixian$RecordNumber <- 1:nrow(JSL1910_sichuan_zhixian)
JSL1910_sichuan_zhixian <- arrange(JSL1910_sichuan_zhixian,xingming,身份二,旗分,出身一,年份季节,RecordNumber)
JSL1910_sichuan_zhixian <- JSL1910_sichuan_zhixian %>%
  group_by(xingming,旗分) %>%
  mutate(PersonID = ifelse(row_number() == 1, 1, 0))
JSL1910_sichuan_zhixian$PersonID_cumsum <- cumsum(JSL1910_sichuan_zhixian$PersonID)
JSL1910_sichuan_zhixian <- JSL1910_sichuan_zhixian %>%
  group_by(PersonID_cumsum) %>%
  mutate(YearsServed = abs((年份季节-年份季节[1])+0.25))
JSL1910_sichuan_zhixian <- JSL1910_sichuan_zhixian %>% group_by( xingming ) %>%
filter( row_number() == n() )
```

7.4.5 数据集外连接的函数 `fuzzy_join()`函数

- `fuzzy_join()`函数，可对两个数据集进行匹配，格式为：

```
`fuzzy_join( x, y, by = NULL, match_fun = NULL, multi_by = NULL, multi_match_fun = NULL,
index_match_fun = NULL, mode = "inner", ...)`
```

7.4.6 `fuzzy_join()`函数的参数说明

- `x`为主数据集，`y`为副数据集，`by = `为根据某个变量进行模糊匹配。
- `match_fun =` 给定两列的向量化函数，返回true或FALSE以判断它们是否匹配。可以是中指定的每对列的函数列表（如果是命名列表，则使用x中的名称）。如果只给定一个函数，则在所有列对上使用它。
- `multi_by =` 要联接的列，其中所有列将用于一起测试匹配项。
- `multi_match_fun =` 用于测试匹配项的参数，同时对每个数据帧中的所有列执行。
- `index_match_fun =` 用于匹配表的参数。
- `mode`匹配的模式，可以选左连接，右连接，外连接，内连接，全连接，需输入对应名称。

```
`fuzzy_join( x, y, by = NULL, match_fun = NULL, multi_by = NULL, multi_match_fun = NULL,
index_match_fun = NULL, mode = "inner", ...)`
```

7.4.7 连接两个数据集的简单尝试

- 根据`fuzzy_join()`函数编写代码:

```
# 安装和读取包
install.packages("fuzzyjoin")
library(fuzzyjoin)

# 利用`fuzzy_join()`函数匹配两个数据集
JSL1906_1910_zhixian <-  stringdist_inner_join(JSL1906_sichuan_zhixian,
                                               JSL1910_sichuan_zhixian,
                                               by = "xingming",
                                               method = "dl",
                                               max_dist = 1,
                                               distance_col = NULL )
```

代码说明

- `stringdist_inner_join`参数是`fuzzy_join`包中一个较为直接、简单的函数，其用法和`merge()`函数相似
- `by = `参数根据`xingming`列进行模糊匹配
- `method`表示计算距离的算法，这里选择的是`dl`算法，可选择的算法包括详见help
- `max_dist`表示链接的最大距离
- `distance_col`表示是否创建包含两个数据集不同点的一个新变量，默认是`NULL`。
- 这样我们就创建了一个新的数据集`JSL1906_1910_zhixian`，里面是四年间长期担任知县的官员，可以基于这个数据集探究为何这些官员能够长期担任知县

注意事项

- R语言在寻找这些满足条件的行的时候，采用内置的距离算法，计算出与公共列距离最近的样本量
- 距离算法只考虑计算出的距离，而不考虑其它因素，其计算过程中只以给定的信息为基准，在最大程度上找到与之相关的样本
- 可以看出，我们之前学习的`merge()`和`fuzzy_join()`函数逻辑是相同的。
- 但是，`fuzzy_join()`这一类连接方法还存在着一些局限性
- 比如不能对不同字体（繁体字、异体字）展开有效的甄别，从而影响数据库链接的效率和准确度
- 我们建议使用者在进行连接之前，先利用字符替换函数将数据库中的繁体字、异体字替换为简体字，然后对数据库的字符变量进行清理，尽量让两个数据库的变量格式达成一致，最后再进行连接

注意事项

- 清代典籍中带有大量的繁体字、异体字，在数据库分析中时常会遇到两个官员名字看起来完全一样，但分析软件依然无法识别，究其原因是两个官员名字字体结构上存在肉眼难以发现的笔画差别，导致匹配失败

- 针对这一问题，有部分学者提出了可行的解决方案，其中康文林、陈必佳两位学者的方法值得借鉴，他们以缙绅录数据库为例，列表展示了缙绅录数据库中最常见的姓和名，并利用数据库外连接的经验，展示了姓和名中异体字配对的累计百分比，进而减少了异体字甄别的时间，提高了分析效率

- 如果有使用者希望对历史数据库的匹配和链接进行更为深入地考察，可以参考康、陈两位学者的文章。

Thanks!