Manipulação de dados em R

Leonardo Sangali Barone

April 03, 2017

Bases de dados relacionais com a gramática básica do pacote dplyr

Até agora, partimos de uma única base de dados e fizemos diversas transformações: mudamos de nomes de variáveis, computamos novas variáveis, selecionamos linhas e colunas, agrupamos e ordenamos. A partir de agora vamos aprender a combinar *data frames* diferentes usando as funções do tipo **join** do pacote *dplyr*. Vamos começar carregando o pacote:

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

Comparação dos pagamentos entre janeiro de 2011 e janeiro de 2017 no Programa Bolsa Família

No lugar de uma amostra dos dados, como no tutorial anterior, utilizaremos no começo deste tutorial os dados de pagamento do Programa Bolsa Família em um município de pequeno porte: Borá, cidade do interior de São Paulo.

Vamos começar importando os dados do mês de janeiro de cada um dos anos para o município de Borá, que foram extraídos dos dados baixados no Portal da Transparência.

Se você estiver com tempo em sala de aula, faça como exercício a construção desses dados (em vez de baixá-los do repositório do curso). A recomendação é baixar os dados manualmente no Portal da Transparência, abrí-los com a função *fread* do pacote *data.table* e selecionar as linhas de Borá com a função *filter* do pacote *dplyr*. Se não estiver com tempo, use o código abaixo.

```
library(readr)
pagamentos11 <- read_delim("https://raw.githubusercontent.com/leobarone/FLS6397/master/data/saques_amos
```

```
## Parsed with column specification:
## cols(
##   UF = col_character(),
##   `Código SIAFI Município` = col_integer(),
##   `Nome Município` = col_character(),
##   `Código Função` = col_integer(),
##   `Código Subfunção` = col_integer(),
##   `Código Programa` = col_integer(),
##   `Código Ação` = col_integer(),
```

```
##   `NIS Favorecido` = col_double(),
##   `Nome Favorecido` = col_character(),
##   `Fonte-Finalidade` = col_character(),
##   `Mês Referência Parcela` = col_integer(),
##   `Valor Parcela` = col_double(),
##   `Mês Competência` = col_character(),
##   `Data do Saque` = col_character()
## )
```

```
## Warning: 1 parsing failure.
##  row           col               expected  actual
## 7401 Valor Parcela no trailing characters ,022.00
```

```
pagamentos17 <- read_delim("https://raw.githubusercontent.com/leobarone/FLS6397/master/data/saques_amos
```

```
## Parsed with column specification:
## cols(
##   UF = col_character(),
##   `Código SIAFI Município` = col_integer(),
##   `Nome Município` = col_character(),
##   `Código Função` = col_integer(),
##   `Código Subfunção` = col_integer(),
##   `Código Programa` = col_integer(),
##   `Código Ação` = col_integer(),
##   `NIS Favorecido` = col_double(),
##   `Nome Favorecido` = col_character(),
##   `Fonte-Finalidade` = col_character(),
##   `Mês Referência Parcela` = col_integer(),
##   `Valor Parcela` = col_double(),
##   `Mês Competência` = col_character(),
##   `Data do Saque` = col_character()
## )
```

```
## Warning: 1 parsing failure.
##  row           col               expected  actual
## 7401 Valor Parcela no trailing characters ,022.00
```

Veja que os dados são semelhantes e têm a mesma estrutura. Esperamos, entretanto, que haja variação entre os anos e que os beneficiários e os valores pagos não sejam os mesmos, seja por que as pessoas entraram e saíram do programa ao longo do tempo, seja por que mudaram de município, seja por que os valores sofreram alteração em virtude de reajuste e mudanças na estrutura das famílias.

Como descobrir tais mudanças? Como saber quem estava em 2011 e também em 2017? Como calcular a variação dos valores para cada beneficiário?

Exercício

Examine as bases de dados "pagamentos11" e "pagamentos17" antes de começarmos a trabalhar com elas. Faça também as seguintes alterações:

- Renomeie as variáveis "NIS Favorecido", "Nome Favorecido" e "Valor Parcela" para "nis", "nome" e "valor", repectivamente.
- Transforme a variável valor em numérica.
- Selecione apenas as três variáveis renomeadas.
- Quantas linhas tem cada base de dados?

Resposta parcial ao exercício anterior (3 primeiros itens)

```
# 2011
pagamentos11 <- pagamentos11 %>%
  rename(nis = `NIS Favorecido`, nome = `Nome Favorecido`, valor = `Valor Parcela`) %>%
  mutate(valor = gsub(",", "", valor), valor = as.numeric(valor)) %>%
  select(nis, nome, valor)

# 2017
pagamentos17 <- pagamentos17 %>%
  rename(nis = `NIS Favorecido`, nome = `Nome Favorecido`, valor = `Valor Parcela`) %>%
  mutate(valor = gsub(",", "", valor), valor = as.numeric(valor)) %>%
  select(nis, nome, valor)
```

Left e Right Join

Tendo as bases preparadas e sabendo que os beneficiários variam entre os anos, podemos começar a "juntá-las".

O elemento essencial dos "joins", ou seja, das combinações de bases de dados, é que haja uma variável que associe observações em uma base com observações em outras. Algumas variáveis costumam ser candidatas naturais para tal tarefa: CPF, NIS e Título de Eleitor, para indíduos e Código de município e UF para unidades político-administrativas. Mas, veremos adiante, podemos ser criativos e utilizar diversas outras "chaves" para conectar tabelas. O identificador que temos disponível em nossos dados é o NIS.

"Letf join" e "right join" são os nomes dados às combinações que mantém todas as linhas de uma das bases de dados, mesmo sem haver correspondente na outra tabela, e inclui apenas os dados da segunda base que encontram correspondência na primeira tabela.

Vamos operar os dois "joins" e ver o que ocorre. Primeiro, "left join":

```
comb_left <- left_join(pagamentos11, pagamentos17, by = "nis")
```

Veja que informamos ao R que a variável que conecta as tabelas é "nis" informando o parâmetro "by".

Quantas linhas resultaram da combinação? Exatamente o mesmo número de linhas de "pagamentos11". Use o *View* para ver o resultado.

Note que há duas variáveis de valor, uma com final ".x" e outra com o final ".y". Isso ocorre por que as duas bases tem os mesmos nomes de variáveis. ".x" significa que a variável veio da primeira tabela do "join" e ".y", da segunda. O mesmo ocorre com a variável "nome". Vamos aproveitar e renomeá-las:

```
comb_left <- comb_left %>% rename(valor11 = valor.x, nome11 = nome.x,
                                  valor17 = valor.y, nome17 = nome.y)
```

Note que em diversas observações o valor e o nome para 2017 contém "NA", que é o símbolo do R para "missing values". O que isso significa? Significa que aquela observações existia em 2011, mas não em 2017. Ao não encontrar correspondência, o R manteve a observação, mas não inseriu nenhum valor.

Se o número de linhas da combinação é o mesmo de 2011, onde estão as observações de 2017 que faltam? Não foram incluídas. O "left join" garante que todas as observações de 2011 sejam mantidas, mesmo sem correspondência, mas exclui todas as observações de 2017 que não encontrem par em 2011.

Vamos deixar de lado rapidamente esta primeira combinação e realizar o "right join" das mesmas tabelas:

```
comb_right <- right_join(pagamentos11, pagamentos17, by = "nis")
```

Novamente, vamos renomear as variáveis:

```
comb_right <- comb_right %>% rename(valor11 = valor.x, nome11 = nome.x,
                                    valor17 = valor.y, nome17 = nome.y)
```

Veja que agora há "missing values" nos valores e nomes de 2011. O "right join" preserva todas as observações de 2017, mesmo sem correspondência em 2011, e não inclui nenhum de 2011 que não encontre correspondência.

Note que "left" e "right" são exatamente a mesma operação, mas com as tabelas em posição (x e y, 1a e 2a, etc) invertidas.

Inner e Full Join

E se quisermos apenas as observações que estão, com certeza, em ambos os anos? Usamos o "inner join". Veja o resultado (já renomeado):

```
comb_inner <- inner_join(pagamentos11, pagamentos17, by = "nis")
comb_inner <- comb_inner %>% rename(valor11 = valor.x, nome11 = nome.x,
                                    valor17 = valor.y, nome17 = nome.y)
```

Agora, não há "missing values". Permanecem na combinação apenas os casos que estão presentes em ambas tabelas.

Finalmente, o "full join" adota o critério oposto: inclui todas as observações de ambos tabelas, não importando se há ou não correspondência, e insere "missing values" onde não há correspondência:

```
comb_full <- full_join(pagamentos11, pagamentos17, by = "nis")
comb_full <- comb_full %>% rename(valor11 = valor.x, nome11 = nome.x,
                                  valor17 = valor.y, nome17 = nome.y)
```

Semi e anti joins

Os quatro tipos de "join" apresentados anteriormente cobrem a totalidade de situações de combinação entre tabelas a partir de um "chave", ou seja, de um índice ou variável que permita estabelecer a relação entre elas.

Há, porém, dois outros tipos de "joins" disponíveis no R bastante úteis.

Se quisermos trabalhar apenas em uma única base de dados, por exemplo, pagamentos11, mas queremos saber quais das observações de 2011 também estão na tabela de 2017, então utilizamos a função *semi_join*. O resultado será semelhante ao da aplicação de *inner_join*, mas sem que novas colunas com os dados de 2017 tenham sido criadas:

```
comb_semi <- semi_join(pagamentos11, pagamentos17, by = "nis")
```

Por fim, *anti_join*, tem comportamento semelhante a *semi_join*, mas, em vez de retornar as observações de 2011 que têm correspondência em 2017, retorna as que não têm par em 2017:

```
comb_anti <- anti_join(pagamentos11, pagamentos17, by = "nis")
```

É perfeitamente possível usar o operador %>% (pipe, como é chamado), para os "joins". Basta colocar a base na posição "x" (primeira a ser inserida) antes do operador. Veja um exemplo:

```
comb_left <- pagamentos11 %>% left_join(pagamentos17, by = "nis")
```

Exercício

Respire fundo e gaste um tempo refletindo sobre os "joins". Você acabou de aprender como operar bancos de dados relacionais e pode parecer bastante difícil num primeiro momento.

Combinação de tabela e agregações cumulativas

Vamos supor que queremos calcular os valores total, médio, máximo, etc, por município e, a seguir, apresentar esses valores como colunas para cada observação. Uma maneira eficiente de fazer isso é a usando a combinação de tabelas. Vamos er como voltando ao exemplo da amostra de saques do Programa Bolsa Família em 2017.

Exercício

Abra a base de dados e faça as transformações necessárias (renomear variáveis e transformar a variável valor em numérica) antes de prosseguir. Tente fazê-lo sem olhar a resposta abaixo.

Resposta ao exercício anterior

```
library(readr)
saques_amostra_201701 <- read_delim("https://raw.githubusercontent.com/leobarone/FLS6397/master/data/sa
```

```
## Parsed with column specification:
## cols(
##   UF = col_character(),
##   `Código SIAFI Município` = col_integer(),
##   `Nome Município` = col_character(),
##   `Código Função` = col_integer(),
##   `Código Subfunção` = col_integer(),
##   `Código Programa` = col_integer(),
##   `Código Ação` = col_integer(),
##   `NIS Favorecido` = col_double(),
##   `Nome Favorecido` = col_character(),
##   `Fonte-Finalidade` = col_character(),
##   `Mês Referência Parcela` = col_integer(),
##   `Valor Parcela` = col_double(),
##   `Mês Competência` = col_character(),
##   `Data do Saque` = col_character()
## )
```

```
## Warning: 1 parsing failure.
##  row           col               expected  actual
## 7401 Valor Parcela no trailing characters ,022.00
```

```
saques_amostra_201701 <- saques_amostra_201701 %>%
  rename(uf = UF,
         munic = `Nome Município`,
         cod_munic = `Código SIAFI Município`,
         nome = `Nome Favorecido`,
         valor = `Valor Parcela`,
         mes = `Mês Competência`,
         data_saque =`Data do Saque`) %>%
  select(uf, munic, cod_munic, nome, valor, mes, data_saque) %>%
  mutate(valor_num = as.numeric(gsub(",", "", valor)))
```

Pasos para agregações cumulativas

Em primeiro lugar, vamos construir uma tabela agrupada por município usando *group_by* e *summarise*:

```
valores_munic <- saques_amostra_201701 %>%
  group_by(cod_munic) %>%
  summarise(contagem = n(),
            soma = sum(valor_num),
            media = mean(valor_num),
            mediana = median(valor_num),
            desvio = sd(valor_num),
            minimo = min(valor_num),
            maximo = max(valor_num))
```

Note que agora temos dois *data frames*, o original e "valores_munic", que pode ser combinados utilizando a variável "cod_munic". Usando *left_join* podemos levar as colunas da nova tabela à base de dados original:

```
saques_amostra_201701 <- saques_amostra_201701 %>%
  left_join(valores_munic, by = "cod_munic")
```

Use *View* para observar que a base de dados original tem agora 7 novas colunas com informações agregadas por município (e que, portanto, se repetem para observações de um mesmo município).

Exercício

- Calcule o total de valores por UF em um novo *data frame*.
- Combine o novo *data frame* com o original para levar a coluna de total de valores ao último.
- A seguir, calcule quanto cada indivíduo na amostra representa, em termor percentuais (dica: crie uma nova variável utilizando *mutate*).