Manipulação de dados em R

Leonardo Sangali Barone

April 03, 2017

Manipulação de dados com a gramática básica do pacote dplyr

Um primeiro exemplo

Nosso primeiro exemplo será a base de dados dos saques efetuados pelos beneficiários do Bolsa Família em janeiro de 2017. Em primeiro lugar, podemos ir ao portal da transparência.

O arquivo de 2017 contém 1.6Gb. É um arquivo bastante grande, com 14 variáveis (colunas) e mais de 12 milhões de linhas. Arquivos grandes podem ser abertos no R usando a função *fread*, disponível no pacote *data.table*. Num futuro breve veremos o que são funções e pacotes. Por enquanto, basta saber que tornamos as funções de um pacote disponíveis se importarmos o pacote para nossa biblioteca usando a função *library*.

```
library(data.table)
library(dplyr)
```

```
## ------------------------------------------------------------------------------
## data.table + dplyr code now lives in dtplyr.
## Please library(dtplyr)!
## ------------------------------------------------------------------------------
##
## Attaching package: 'dplyr'
## The following objects are masked from 'package:data.table':
##
##     between, first, last
## The following objects are masked from 'package:stats':
##
##     filter, lag
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

Vamos agora abrir os dados com a função *fread*, que vimos em tutoriais passados.

```
saques <- fread("201701_BolsaFamiliaSacado.csv", encoding = "Latin-1")
```

Simples, não? Vamos ver o resultado da importação com a função *head*, que retorna as 6 primeiras linhas do banco de dados:

```
head(saques)
```

```
##    UF Código SIAFI Município Nome Município Código Função Código Subfunção
## 1: BA                    3897          SEABRA             8              244
## 2: PE                    2531          RECIFE             8              244
## 3: PI                    1167        PIRIPIRI             8              244
## 4: MA                     869        PINHEIRO             8              244
## 5: PE                    2427         GRAVATA             8              244
```

```
## 6: BA                   3897         SEABRA             8              244
##    Código Programa Código Ação NIS Favorecido
## 1:            1335        8442    16074176737
## 2:            1335        8442    20033347012
## 3:            1335        8442    16356178516
## 4:            1335        8442    16099396730
## 5:            1335        8442    16027702517
## 6:            1335        8442    23672637749
##                          Nome Favorecido              Fonte-Finalidade
## 1:         FLAVIANO SEBASTIAO DOS SANTOS CAIXA - Programa Bolsa Família
## 2: ELANE PATRICIA DA SILVA DAMASIO NUNES CAIXA - Programa Bolsa Família
## 3:               LUIZA FLORINDA DA SILVA CAIXA - Programa Bolsa Família
## 4:          EDNA MARIA FERREIRA PINHEIRO CAIXA - Programa Bolsa Família
## 5:                    ANA MARIA DA SILVA CAIXA - Programa Bolsa Família
## 6:          IVANILSON DOS SANTOS MACHADO CAIXA - Programa Bolsa Família
##    Mês Referência Parcela Valor Parcela Mês Competência Data do Saque
## 1:                      1        131.00         01/2017    27/01/2017
## 2:                      1        170.00         01/2017    19/01/2017
## 3:                      1         85.00         01/2017    25/01/2017
## 4:                      1         85.00         01/2017    31/01/2017
## 5:                      1        124.00         01/2017    26/01/2017
## 6:                      1         85.00         01/2017    30/01/2017
```

Para checar quantas linhas e colunas há nos dados importados, usamos a função *dim*:

```
dim(saques)
```

```
## [1] 12379599       14
```

E observar os nomes das variáveis:

```
names(saques)
```

```
##  [1] "UF"                     "Código SIAFI Município"
##  [3] "Nome Município"         "Código Função"
##  [5] "Código Subfunção"       "Código Programa"
##  [7] "Código Ação"            "NIS Favorecido"
##  [9] "Nome Favorecido"        "Fonte-Finalidade"
## [11] "Mês Referência Parcela" "Valor Parcela"
## [13] "Mês Competência"        "Data do Saque"
```

Nomes com espaços, acentos e caracteres especiais são bastante ruins de se trabalhar. Além disso, há um problema nesse banco de dados: a coluna "Valor Parcela", que contém os valores sacados, não foi lida automaticamente como número. Assim, vamos renomear algumas variáveis (NIS, UF, Mês do saque e código do município, por exemplo), e vamos construir uma nova variável "valor", que contenha números e não números interpretados como texto:

```
saques <- saques %>%
  rename(nis = `NIS Favorecido`, uf = UF, munic = `Nome Município`, mes = `Mês Referência Parcela`) %>%
  mutate(valor = as.numeric(gsub(",", "", saques$`Valor Parcela`)))
```

Se você prestar atenção no código acima, verá que ele é mais simples do que parece. Ele é um exemplo de como podemos usar o pacote *dplyr* para manipularmos dados.

Agora que as variáveis têm nomes mais curtos e fáceis de trabalhar e a coluna "valor" foi construída, podemos explorar os dados. Vamos fazer uma tabela de contagem de saques por UF, um histograma com a distribuição dos valores sacados e uma tabela com a soma dos valores por UF.

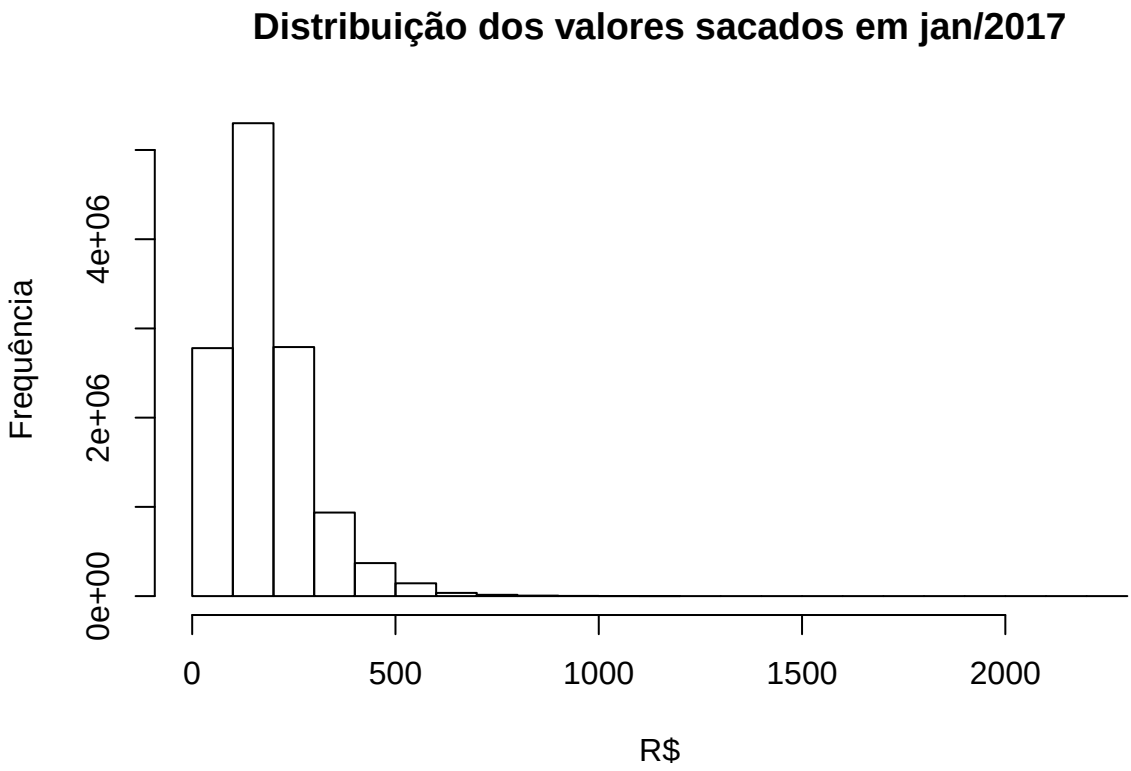
Primeiro, a tabela que contém a contagem por UF de todos os beneficiários que sacaram em janeiro de 2017:

```
saques %>% group_by(uf) %>% summarise(contagem = n())
```

```
## # A tibble: 27 × 2
##       uf contagem
##    <chr>    <int>
## 1     AC    78254
## 2     AL   360177
## 3     AM   336843
## 4     AP    68003
## 5     BA  1605196
## 6     CE   959198
## 7     DF    87466
## 8     ES   157285
## 9     GO   283330
## 10    MA   849080
## # ... with 17 more rows
```

Vejamos agora a distribuição de valores sacados em janeiro de 2017 em um histograma:

```
hist(saques$valor, main = "Distribuição dos valores sacados em jan/2017", xlab = "R$", ylab = "Frequênci
```



Legal, não?

Finalmente, vamos fazer o somatório dos valores sacado por UF em 2017:

```
saques %>% group_by(uf) %>% summarise(valores = sum(valor))
```

```
## # A tibble: 27 × 2
##       uf   valores
##    <chr>     <dbl>
## 1     AC  20791647
## 2     AL  67635154
```

```
## 3     AM  75822600
## 4     AP  14883503
## 5     BA 292856096
## 6     CE 172109845
## 7     DF  14143552
## 8     ES  25366385
## 9     GO  44317186
## 10    MA 180730880
## # ... with 17 more rows
```

Note que com poucas linhas de código processamos um bocado de dados e geramos informações úteis para a compreensão do programa bolsa família. Em um futuro breve, veremos como organizar, manipular e "misturar" bases de dados volumosas e extrair informações delas.

Introdução ao pacote dplyr

Um dos aspectos mais incríveis da linguagem R é o desenvolvimento de novas funcionalidades pela comunidade de usuários. Algumas das melhores soluções desenvolvidas são relacionadas à "gramática para bases de dados", ou seja, à maneira como importamos, organizamos, manipulamos e extraímos informações das bases de dados.

Neste tutorial vamos nos concentrar na "gramática" mais popular: o pacote *dplyr*. Já vimos um pouco como ele funciona no exemplo, com dados dos saques do Bolsa Família em janeiro de 2017. Voltemos a este exemplo, mas agora com uma versão mais simples dos dados extraído aleatoriamente do banco original, com apenas 10 mil saques.

```
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
```

Para explorar os dados vamos usar uma função nova, *glimpse*, aplicável a **tibbles**:

```
glimpse(saques_amostra_201701)
```

```
## Observations: 10,000
## Variables: 14
```

```
## $ UF                       <chr> "SP", "SP", "BA", "SP", "SP", "PE", "BA...
## $ Código SIAFI Município <int> 7107, 6313, 3541, 7245, 6563, 2497, 351...
## $ Nome Município           <chr> "SAO PAULO", "CARAPICUIBA", "IBICARAI",...
## $ Código Função            <int> 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, ...
## $ Código Subfunção         <int> 244, 244, 244, 244, 244, 244, 244, 244,...
## $ Código Programa          <int> 1335, 1335, 1335, 1335, 1335, 1335, 133...
## $ Código Ação              <int> 8442, 8442, 8442, 8442, 8442, 8442, 844...
## $ NIS Favorecido           <dbl> 16482884963, 20710655384, 12318885348, ...
## $ Nome Favorecido          <chr> "MARILENE SILVA ALMEIDA", "GILDENE RITA...
## $ Fonte-Finalidade         <chr> "CAIXA - Programa Bolsa Família", "CAIX...
## $ Mês Referência Parcela <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ Valor Parcela            <dbl> 227, 46, 85, 242, 124, 163, 171, 246, 1...
## $ Mês Competência          <chr> "01/2017", "01/2017", "01/2017", "01/20...
## $ Data do Saque            <chr> "20/01/2017", "23/01/2017", "27/01/2017...
```

Renomeando variáveis

Com certa frequência, obtemos dados cujos nomes das colunas são compostos, contêm acentuação, cecedilha e demais caracteres especiais. Dá um tremendo trabalho usar nomes com tais característica. O ideal é termos nomes sem espaço (você pode usar ponto ou subscrito para separar palavras em um nome composto), preferencialmente com letras minísculas sem acento e números, apenas. Vamos começar renomeando algumas variáveis no nosso banco de dados, cujos nomes vemos com o comando abaixo:

```
names(saques_amostra_201701)
```

```
##  [1] "UF"                     "Código SIAFI Município"
##  [3] "Nome Município"         "Código Função"
##  [5] "Código Subfunção"       "Código Programa"
##  [7] "Código Ação"            "NIS Favorecido"
##  [9] "Nome Favorecido"        "Fonte-Finalidade"
## [11] "Mês Referência Parcela" "Valor Parcela"
## [13] "Mês Competência"        "Data do Saque"
```

O primeiro argumento da função *rename* deve ser a base de dados cujos nomes das variáveis serão renomeados. Depois da primeir vírgula, inserimos todos as modificações de nomes, novamente separadas por vírgulas, e da seguinte maneira. Exemplo: nome_novo = nome_velho. Caso os nomes tenha espaço, como no nosso exemplo, é preciso usar o acento agudo antes e depois do nome antigo para que o R entenda onde ele começa e termina. Exemplo: nome|_novo = ‘Nome Velho‘. Veja o exemplo, em que damos novos nomes às variáveis “UF” e “Nome Município”

```
saques_amostra_201701 <- rename(saques_amostra_201701, uf = UF, munic = `Nome Município`)
```

Exercício

Renomeie as variáveis “Código SIAFI Município”, “Nome Favorecido”, “Valor Parcela”, “Mês Competência” e “Data do Saque” como “cod_munic”, “nome”, “valor”, “mes”, “data_saque”, respectivamente.

Uma gramática, duas formas

Se voltarmos ao exemplo do começo da apostila, veremos que usamos uma sintaxe ligeiramente diferente para a mesma tarefa, renomear variáveis. Vamos olhar para ela:

```
saques_amostra_201701 <- saques_amostra_201701 %>% rename(uf = UF, munic = `Nome Município`)
```

Usando o operador %>%, denominado *pipe*, retiramos de dentro da função *rename* o banco de dados cujas variáveis serão renomeadas. Essa outra sintaxe tem uma vantagem grande sobre a anterior: ela permite emendar uma operação de transformação do banco de dados na outra. Veremos adiante como fazer isso. Por enquanto, tenha em mente que o resultado é o mesmo para qualquer uma das duas formas.

Selecionando colunas

Algumas colunas são claramente dispensáveis em nosso banco de dados. Por exemplo, já sabemos que "Código Função", "Código Subfunção", "Código Programa" e "Código Ação" não variam entre as linhas, pois todas se referem ao Programa Bolsa Família. Vamos ficar apenas com as variáveis que já havíamos renomeado.

```
saques_amostra_201701 <- select(saques_amostra_201701, uf, munic, cod_munic, nome, valor, mes, data_saqu
```

ou usando o operador %>%, chamado **pipe**,

```
saques_amostra_201701 <- saques_amostra_201701 %>% select(uf, munic, cod_munic, nome, valor, mes, data_s
```

Operador %>% para "emendar" tarefas

O que o operador **pipe** faz é simplesmente colocar o primeiro argumento da função (no caso acima, o *data frame*), fora e antes da própria função. Ele permite lermos o código, informalmente, da seguinte maneira: "pegue o data frame x e aplique a ele esta função". Veremos abaixo que podemos fazer uma cadeia de operações ("pipeline"), que pode ser lida informalmente como: "pegue o data frame x e aplique a ele esta função, e depois essa, e depois essa outra, etc".

A grande vantagem de trabalharmos com o operador %>% é não precisar repetir o nome do *data frame* diversas vezes ao aplicarmos a ele um conjunto de operações.

Use o comando *rm* para deletar a base de dados e abra novamente. Vejamos agora como usamos o operador %>% para "emendar" tarefas:

```
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)
```

Em uma única sequência de operações, alteramos os nomes das variáveis e selecionamos as que permaneceriam no banco de dados. Esta forma de programa, tenha certeza, é bastante mais econômica.

Voltemos agora aos dados. Se observarmos as dimensões da nossa base dados, veremos que ela tem 10 mil linhas, mas apenas 7 colunas agora:

```
dim(saques_amostra_201701)
```

```
## [1] 10000     7
```

Transformando variáveis

Vimos no exemplo que a variável valor, apesar de conter números, foi lida como texto. Isso ocorre por que o R não entende o uso da vírgula como separador de mlhar. Como resolver um problema desses?

Usaremos a função *mutate* para operar transformações nas variáveis existentes e criar variáveis novas. Há inúmeras transformações possíveis e elas lembram bastante as funções de outros softwares, como MS Excel. Vamos ver algumas das mais importantes.

Um exemplo simples: vamor gerar uma nova variável com os nomes dos beneficiários em minúsculo usando a função *tolower*. Veja:

```
glimpse(saques_amostra_201701)
```

```
## Observations: 10,000
## Variables: 7
## $ uf         <chr> "SP", "SP", "BA", "SP", "SP", "PE", "BA", "BA", "SP...
## $ munic      <chr> "SAO PAULO", "CARAPICUIBA", "IBICARAI", "VOTUPORANG...
## $ cod_munic  <int> 7107, 6313, 3541, 7245, 6563, 2497, 3511, 3579, 709...
## $ nome       <chr> "MARILENE SILVA ALMEIDA", "GILDENE RITA DA SILVA", ...
## $ valor      <dbl> 227, 46, 85, 242, 124, 163, 171, 246, 170, 163, 312...
## $ mes        <chr> "01/2017", "01/2017", "01/2017", "01/2017", "01/201...
## $ data_saque <chr> "20/01/2017", "23/01/2017", "27/01/2017", "18/01/20...
```

```
saques_amostra_201701 <- saques_amostra_201701 %>% mutate(nome_min = tolower(nome))
```

ou, em uma forma alternativa,

```
saques_amostra_201701 <-mutate(saques_amostra_201701, nome_min = tolower(nome))
```

Use o comando View para visualizar o resultado da coluna criada à direita do banco de dados. Simples, não? Basta inserimos dentro do comando mutate a expressão da transformação que queremos.

Vamos a um exemplo um pouco mais difícil: substituir vírgula por vazio em um texto e, a seguir, indicar que o texto é, na verdade, um número. Em vez de criar uma nova variável "valor", vamos apenas alterar a variável já existente duas vezes. Com a função *gsub*, faremos a substituição da vírgula por vazio e com a função *as.numeric* faremos a transformação texto-número.

```
saques_amostra_201701 <- saques_amostra_201701 %>%
  mutate(valor_num = gsub(",", "", valor)) %>%
  mutate(valor_num = as.numeric(valor_num))
```

A operação reversa a *as.numeric*, que transforma número em texto, é *as.character*. Vamos explorar as funções de texto e tranformação de variáveis em outro tutorial.

Precisamos usar *mutate* duas vezes? Não. As duas formas abaixo são equivalentes à acima:

```
saques_amostra_201701 <- saques_amostra_201701 %>%
  mutate(valor_num = as.numeric(gsub(",", "", valor)))
```

```
saques_amostra_201701 <- saques_amostra_201701 %>%
  mutate(valor_num = gsub(",", "", valor), valor_num = as.numeric(valor_num))
```

Vamos ver um novo exemplo. Faremos agora duas operações separadas, cada uma resultando em uma nova variável: dividiremos o valor por 3.2 para transformar o valor em dólares; e somaremos R$ 10 ao valor, pelo simples exercício de ver a transformação.

```
saques_amostra_201701 <- saques_amostra_201701 %>%
  mutate(valor_dolar = valor / 3.2, valor10 = valor_num + 10)
```

Use o comando *View* para ver as novas variáveis no banco de dados.

As operações de soma, subtração, divisão, multiplicação, módulo entre mais de uma variável ou entre variáveis e valores são válidas e facilmente executadas como acima mostramos.

Nem todas as transformações de variáveis, porém, são operações matemáticas. Vamos transformar a variável

valor em uma nova variável que indique se o valor sacado é "Alto" (acima de R$ 300) ou "Baixo" (abaixo de R$ 500) com o comando *cut*:

```
saques_amostra_201701 <- saques_amostra_201701 %>%
  mutate(valor_categorico = cut(valor_num, c(0, 300, Inf), c("Baixo", "Alto")))
```

E se quisermos recodificar uma variável de texto? Por exemplo, vamos examinar a variável "mes". Ela contém o "Mês de Competência" do saque. Usemos a função *table* para examiná-la:

```
table(saques_amostra_201701$mes)
```

```
##
## 01/2017 11/2016 12/2016
##    9440     124     436
```

São 3 valores possíveis em nossa amostra: "11/2016", "12/2016" e "01/2017" em nossa amostra. Vamos gerar uma nova variável, ano, que indica apenas se a competência é 2016 ou 2017. Vamos começar fazendo uma cópia da variável original e depois substituiremos cada um dos valores:

```
saques_amostra_201701 <- saques_amostra_201701 %>%
  mutate(ano = mes,
         ano = replace(ano, ano == "11/2016", "2016"),
         ano = replace(ano, ano == "12/2016", "2016"),
         ano = replace(ano, ano == "01/2017", "2017"))
```

Um pouco trabalhoso, mas cumpre o objetivo. Uma maneira mais inteligente é usar o comando *recode*:

```
saques_amostra_201701 <- saques_amostra_201701 %>%
  mutate(ano = recode(mes, "11/2016" = "2016", "12/2016" = "2016", "01/2017" = "2017"))
```

Com as operações matemáticas, as transformações *as.numeric* e *as.character* e os comandos *cut*, *replace* e *recode* podemos fazer praticamente qualquer recodificação de variáveis que envolva texto e números. A exceção, por enquanto, serão as variáveis da classe *factor*, que já vimos em tutorais anteriores. Para os interessados em expressões regulares, recomendo a leitura do arquivo "help" da família da função *gsub*, que inclui *grep*, *regexpre* e outras.

Exercício

Use os exemplos acima para gerar novas variáveis conforme instruções abaixo:

- Faça uma nova divisão da variável "valor" a seu critério. Chame a nova variável de "valor_categorico2".
- Cria uma variável "valor_euro", que é o valor calculado em Euros.
- Recodifique "valor_categorico" chamando as categorias de "Abaixo de R$300" e "Acima de R$300". Chame a nova variável de "valor_categorico3".
- Usando a função *recode* Recodifique "mes" em 3 novos valores: "Novembro", "Dezembro" e "Janeiro". Chame a nova variável de "mes_novo".
- Usando a função *replace* Recodifique "mes" em 3 novos valores: "Novembro", "Dezembro" e "Janeiro". Chame a nova variável de "mes_novo2".

Filtrando linhas

Por vezes, queremos trabalhar apenas com um conjunto de linhas do nosso banco de dados. Por exemplo, se quisermos selecionar apenas os beneficiários do estado do Espírito Santo e salvarmos em um objeto chamado saques_amostra_ES:

```
saques_amostra_ES <- saques_amostra_201701 %>% filter(uf == "ES")
```

ou

```
saques_amostra_ES <-filter(saques_amostra_201701, uf == "ES")
```

Até o uso da função *filter*, não há nada de novo para nós. A novidade está na condição uf == "ES", que indica que apenas as linhas cuja variável *uf* assumo valor igual a ES devem ser consideradas. Em primeiro lugar, qual é a razão de usarmos duas vezes o sinal de igualdade (==)? Normalmente, usamos um igual para atribuir algo a um nome ou para definir algo igual a um valor. Neste caso, estamos comparando duas coisas, ou seja, estamos verificando se o conteúdo de cada linha é igual a um valor.

Além da igualdade, poderíamos usar outros símbolos: maior (>). maior ou igual (>=), menor (<), menor ou igual (<=) e diferente (!=).

Também utilizamos aspas em "ES". Como estamos comparando os valores para cada linha a um texto, devemos usar as aspas.

Vamos supor agora que apenas os estados do Centro-Oeste nos interessam. Vamos criar um novo *data frame*, chamado saques_amostra_CO, que atenda a este critério:

```
saques_amostra_CO <- saques_amostra_201701 %>%
  filter(uf == "MT" | uf == "MS" | uf == "Df" | uf == "GO")
```

Note que, para dizer que queremos as quatro condições atendidas, utilizamos uma barra vertical. A barra é o símbolo "ou", e indica que todas as observações que atenderem a uma ou outra condição serão incluídas.

Vamos supor que queremos estabelecer agora condições para a seleção de linhas a partir de duas variáveis. Por exemplo, queremos incluir observações do Mato Grosso e que também tenham ano de competência (variável que criamos acima) igual a 2016. O símbolo da conjunção "e" é "&". Veja como utilizá-lo:

```
saques_amostra_MT_2016 <- saques_amostra_201701 %>% filter(uf == "MT" & ano == "2016")
```

Ao usar duas variáveis diferentes para filter e a conjunção "e", podemos escrever o comando separando as condições por vírgula e dispensar o operador "&":

```
saques_amostra_MT_2016 <- saques_amostra_201701 %>% filter(uf == "MT", ano == "2016")
```

Você pode combinar quantas condições precisar. Se houver ambiguidade quanto à ordem das condições, use parênteses das mesma forma que usamos com operações aritméticas.

Exercício

- Crie um novo *data frame* apenas com as observações cujo mês de competência é janeiro.
- Crie um novo *data frame* apenas com as observações cujo valor é superior a R$ 500.
- Crie um novo *data frame* apenas com as observações da região Sul.

Agrupando

Por enquanto, por mais que transformássemos as variáveis do banco de dados ou selecionássemos linhas, as unidades continuavam a ser os saques realizados por cada beneficiário. E se, no entanto, nos interessar trabalhar no nível mais agregado? Voltemos ao exemplo do início do tutorial.

Vamos começar produzindo um novo *data frame*, mas que agora contenha a informação de quantos saques foram realizados em cada UF:

```
contagem_uf <- saques_amostra_201701 %>%
  group_by(uf) %>%
  summarise(contagem = n())
```

Veja que usamos simultaneamente 2 funções, *group_by* e *summarise*. Eles tem significado literal: na primeira, inserimos as variáveis pelas quais agruparemos o banco de dados. Na segunda, as operações de "sumário", ou seja, de condensação, que faremos com o banco de dados e com as demais variáveis. No exemplo acima, apenas contamos, usando a função n(), quantas linhas pertencem a cada uf, que é a variável de grupo.

Vamos complicar um pouco mais. Suponhamos que, além da contagem, tenhamos interesse na soma, média, mediana, desvio padrão, mínimo, máximo dos valores no mesmo resultado. Neste caso, devemos inserir novas operações na função *summarize*, separadas por vírgula:

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

Use *View* para observar o resultado.

A sessão *Useful Summary Functions* do livro *R for Data Science* traz uma relação mais completa de funçoes que podem ser usandas com *summarise*. O "cheatsheet" da RStudio oferece uma lista para uso rápido.

Exercício

Usando a variável "mes_novo", calcule a contagem, soma e média de valores para cada mês.

Mais de um grupo

E se quisermos agrupar por mais de uma variável? Veja como fazer um agrupamento por "mes"" e "uf", reportando apenas a contagem de saques em cada combinação de grupos:

```
contagem_uf_mes <- saques_amostra_201701 %>%
  group_by(uf, mes) %>%
  summarise(contagem = n())
```

Note que, agora, cada uf é repetida duas ou três vezes, uma para cada mês. Cada grupo gera uma nova coluna e as linhas representam exatamente a combinação de grupos de cada variável presente nos dados.

Finalmente, podemos utilizar múltiplas variáveis de grupo em conjunto e também gerar um sumário com diversas varáveis, como no exemplo a seguir, que combina parte dos dois anteriores:

```
valores_uf_mes <- saques_amostra_201701 %>%
  group_by(uf, mes) %>%
  summarise(contagem = n(),
            soma = sum(valor),
            media = mean(valor),
            desvio = sd(valor))
```

Novo *data frame* ou tabela para análise?

O uso das funções *group_by* e *summarise* pode ter 2 propósitos: produzir uma tabela para análise, como fizemos acima, ou gerar um novo *data frame*. Basicamente, os usos dependem do tamanho redução que geramos no banco de dados. Por exemplo, podemos gerar os totais de valores transferidos para cada município

(que, se tivessemos o banco completo, geraria um banco de aprox. 5,5 mil linhas) para, a seguir, inseri-lo nos dados originais como coluna. Por enquanto, ainda não aprendemos a relacionar dois *data frames* entre si, mas vejamos como seria a base de dados com municípios como linhas:

```
saques_amostra_munic <- saques_amostra_201701 %>%
  group_by(munic) %>%
  summarise(contagem = n(),
            soma = sum(valor),
            media = mean(valor))
```

Ordenando a base de dados

Quando trabalhamos com bases de dados muito grandes, faz pouco sentido ordená-las. Entretanto, quando trabalhamos numa escala menor, com poucas linha, como nos exemplos acima, convém ordenar a tabela (veja que, neste ponto, faz pouco sentido diferenciar tabela de *data frame*, pois tornam-se sinônimos) por alguma variável de interesse.

Se quisermos ordenar, de forma crescente, a tabela de valores por uf pela soma de valores, basta usar o comando *arrange*:

```
valores_uf <- valores_uf %>% arrange(soma)
```

Apenas para ilustrar, poderíamos ter usado o comando *arrange* diretamente ao gerar a tabela:

```
valores_uf <- saques_amostra_201701 %>%
  group_by(uf) %>%
  summarise(contagem = n(),
            soma = sum(valor),
            media = mean(valor),
            mediana = median(valor),
            desvio = sd(valor),
            minimo = min(valor),
            maximo = max(valor)) %>%
  arrange(soma)
```

Se quisermos rearranjar uma tabela, agora em ordem decrescente de média de valores, por exemplo, usamos *desc*:

```
valores_uf <- valores_uf %>% arrange(desc(soma))
```

Para usar mais de uma variável ao ordenar, basta colocá-las em ordem de prioridade e separá-las por vírgula. No exemplo abaixo ordenamos pela mediana (descendente) e depois pelo máximo:

```
valores_uf <- valores_uf %>% arrange(desc(mediana), maximo)
```