Básico da Linguagem e Data Frame

Leonardo Sangali Barone

March 13, 2017

Começando pelo meio: data frames

Uma característica distintiva da linguagem de programação R é ter sido desenvolvida para a análise de dados. E quando pensamos em análise de dados, a protagonista do show é a *base de dados* ou, como vamos conhecer a partir de agora, **data frame**.

Por esta razão, em vez de aprender como fazer aritmética, elaborar funções ou executar loops para repetir tarefas e outros aspectos básicos da linguagem, vamos começar olhando para o R como um software concorrente dos demais utilizados para análise de dados em ciências sociais, como SPSS, Stata, SAS e companhia.

As principais características de um data frame são: (1) cada coluna representa uma variável (ou característica) de um conjunto de observações; (2) cada linha representa um observação e contém os valores de cada variável para tal observação. Vejamos um exemplo:

Candidato	Partido	Votos
Beatriz	PMDB	350
Danilo	SOL	1598
Pedro	PTB	784
Davi	PSD	580
Mateus	PV	2

Note que em uma linha os elementos são de tipos de diferentes: na primeira coluna há uma nome (texto), na segunda uma sigla de partido (texto, mas limitado a um conjunto de siglas) e na terceira votos (número inteiros).

Por outro lado, em cada coluna há somente elementos de um tipo. Por exemplo, há apenas números inteiros na coluna votos. Colunas são variáveis e por isso aceitam registros de um único tipo. Se você já fez um curso de estatísticas básica ou de métodos quantitativos deve se lembrar que as variáveis são classificadas da seguinte maneira:

1- Discretas

- Nominais, que são categorias (normalmente texto) não ordenadas
- Ordinais, que são categorias (normalmente texto) ordenadas
- Inteiros, ou seja, o conjunto dos números inteiros

2- Contínuas, números que podem assumir valores não inteiros

Se destacamos uma coluna do nosso data frame, temos um **vetor**. Por exemplo, a variável "Votos" pode ser presentado da seguinte maneira: {350, 1598, 784, 580, 2}. Um data frame é um conjunto de variáveis (vetores) dispostos na vertical e combinados um ao lado do outro.

Data frame e vetores são **objetos** na linguagem R.

Vamos ver como o R representa vetores e data frames na tela. Antes disso, é preciso "abrir" um data frame.

Pausa para pacotes

Uma das características mais atrativas da linguagem R é o desenvolvimento de **pacotes** pela comunidade de usuários. Pacotes são conjuntos de funções (aka comandos) e, por vezes, guardam também dados em diversos formatos.

Vamos carregar um pacote chamado *datasets*, que contém diversos conjuntos de dados úteis para fins didáticos. Para carregar um pacote (e, portanto, tornar as funções disponíveis para uso naquela sessão de R) usamos a função *library*:

```
library(datasets)
```

De volta aos dataframes

Com a função *data* podemos carregar um conjunto de dados disponível na sua sessão de R. Utilizaremos a base *mtcars*, que contém dados da revista *Motor Trend US* sobre características (variáveis!!!) de 32 automóveis (esse é um dos conjuntos de dados mais populares em cursos introdutórios de R).

```
data(mtcars)
```

Pronto! Logo mais veremoscomo abrir conjuntos de dados de outras fontes (arquivos de texto, outros softwares, etc), mas já podemos começar a trabalhar com *data frames*.

Antes de avançar, vamos usar o **help** (documentação) do R para descobrir o que há no *data frame* chamado *mtcars*:

```
?mtcars
```

Você pode ler com calma antes de avançar.

Se quiseremos olhar para os dados que acabamos de carregar utilizamos a função *View* (com V maiúsculo, algo pouco usual em R):

```
View(mtcars)
```

Com apenas 32 observações, não fica tão difícil "olhar" para os dados. Mas e se estívessemos trabalhando com, por exemplo, o total de candidatos vereadores no Brasil em 2016 (aprox. meio milhão de candidatos)? Seria útil utilizar o comando *View*?

Do editor de planilhas ao R - parte 1

A partir desse ponto no curso vamos resistir à tentação de "olhar" para os dados. O hábito de quem utiliza com editores de planilha como MS Excel ou Libre Office, ou ainda com algums softwares de análise de dados como SPSS e Minitab, é trabalhar "olhando" para os dados, ou seja, para os valores de cada célula de uma base dados.

Você perceberá em pouco tempo que isso não é necessário. Na verdade, é contraproducente. Nas primeiras aulas vamos nos munir de ferramentas que nos permitirão conhecer os dados sem olhá-los diretamente.

Head no lugar de View

Por exemplo, podemos substituir a função *View* pela função *head*. Veja o resultado:

```
head(mtcars)
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
```

```
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

Com apenas as 6 primeiras linhas do *data frame* temos noção de todo o conjunto. Sabemos rapidamente que os nomes dos carros são o nome de cada uma das linhas, e que o nome das colunas indicam qual característica está armazenada coluna (lembra-se da documentação de *mtcars* que você acabou de ler).

Alternativamente, podemos usar a função *str* (atalho para "structure"):

```
str(mtcars)
```

```
## 'data.frame':    32 obs. of  11 variables:
##  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
##  $ disp: num  160 160 108 258 360 ...
##  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
##  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
##  $ qsec: num  16.5 17 18.6 19.4 17 ...
##  $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
##  $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
##  $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
##  $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

Com *str* sabemos qual é a lista de variáveis (colunas) no *data frame*, de qual tipo são – no caso, todas são numéricas e vamos falar sobre esse tema mais tarde – e os primeiros valores de cada uma, além do número total de observações e variáveis mostrados no topo do **output**.

Há outras maneiras de ver obter o número linhas e colunas de um *data frame*:

```
nrow(mtcars)
```

```
## [1] 32
```

```
ncol(mtcars)
```

```
## [1] 11
```

```
dim(mtcars)
```

```
## [1] 32 11
```

nrow retorna o número de linhas; *ncol*, o de coluna; *dim* as dimensões (na ordem linha e depois coluna) do objeto.

names, por sua vez, retorna os nomes das variáveis do *data frame*

```
names(mtcars)
```

```
##  [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"   "gear"
## [11] "carb"
```

Argumentos ou parâmetros das funções

Note que em todas as funções que utilizamos até agora, *mtcars* está dentro do parêntesis que segue o nome da função. Essa **sintax** é característica das funções de R. O que vai entre parêntesis são os **argumentos** ou **parâmetros** da função, ou seja, os "inputs" que serão transformados.

Uma função pode receber mais de um argumento Pode também ter argumentos não obrigatórios, ou seja, não é necessário informá-los se você não quiser alterar os valores pré-definidos. Por exemplo, a função *head* contém o argumento *n*, que se refere ao número de linhas a serem **impressas** na tela, pré-estabelecido em 6 (você pode conhecer os argumentos da função na documentação do R (usando *?* antes do nome da função). Para alterá-lo para 10, por exemplo, basta fazer:

```
head(x = mtcars, n = 10)
```

```
##                    mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360        14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D         24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230          22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
```

x é o argumento que já havíamos utilizado anteriormente e indica em que objeto a função *head* será aplicada. Dica: você pode omitir tanto "x =" quanto "n =" se você já conhecer a ordem de cada argumento no uso da função. Veja que neste caso estamos utilizando o símbolo "=" sem fazer a atribuição de dados a um objeto, mas para atribuir valores (ou objetos) aos argumento de uma função. Para não haver confusão é preferível usar o símbolo "<-" para atribuição e o "=" para as demais situações.

Pausa para um comentário

Podemos fazer comentários no meio do código. Basta usar # e tudo que seguir até o final da linha não será interpertado pelo R como código. Por exemplo:

```
# Imprime o nome das variaveis do data frame mtcars
names(mtcars)
```

```
##  [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"   "gear"
## [11] "carb"
```

```
names(mtcars) # Repetindo o comando acima com comentario em outro lugar
```

```
##  [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"   "gear"
## [11] "carb"
```

Comentários são extremamente úteis para documentar seu código. Documentar é parte de programar e você deve pensar nas pessoas com as quais vai compartilhar o código e no fato de que com certeza não se lembrará do que fez em pouco tempo (garanto, você vai esquecer).

Construindo vetores e data frames

Vamos esquecer o *data frame* com o qual estávamos trabalhando até agora. Para remover um objeto do **Workspace** fazemos:

```
rm(mtcars)
```

Vamos fazer isso de forma menos entediante. Vamos montar um banco de dados de notícias.

Escolha 2 jonais ou portais de notícias diferentes. Vá em cada um deles e colete 3 notícias. Em casa notícia, colete as seguintes informações:

- Nome do jornal ou portal
- Data da notícia (não precisa coletar a hora)
- Título
- Autor(a)
- Número de palavras no texto (use o MS Word ou Libre Office se precisar - chute se tiver preguiça)
- Marque 1 se a notícia for sobre política e 0 caso contrário
- Marque 1 se a notícia for sobre esporte e 0 caso contrário
- Marque TRUE se a notícia contiver vídeo e FALSE caso contrário

Insira as informações nos vetores em ordem de coleta das notícias

Com cada informação, vamos construir um vetor. Vejam meus exemplos. Começando col o nome do jornal ou portal:

```
jornal <- c("The Guardian", "The Guardian", "The Guardian", "Folha de São Paulo",
            "Folha de São Paulo", "Folha de São Paulo")
```

Opa, calma! Temos um monte de coisas novas aqui!

A primeira delas é: se você criou corretamente o vetor *jornal*, então nada aconteceu na sua tela, ou seja, nenhum output foi impresso. Isso se deve ao fato de que criamos um novo **objeto**, chamado *jornal* e **atribuímos** a ele os valores coletados sobre os nomes dos veículos nos quais coletamos as notícias. O símbolo de **atribuição** em R é **<-**. Note que o símbolo lembra uma seta para a esquerda, indicando que o conteúdo do vetor será armazenado no objeto *jornal*.

Objetos não são nada mais do que um nome usado para armazenar dados na memória RAM (temporária) do seu computador. No exemplo acima, *jornal* é o objeto e o vetor é a informação armazenada. Uma vez criado, o objeto está disponível para ser usado novamente, pois ele ficará disponível no **workspace**. Veremos adiante que podemos criar um *data frame* a partir de vários vetores armazenados na memória e como examinar o conteúdo do *workspace*. Especificamente no RStudio, os objetos ficam disponíveis na no painel *environment* (que provavelmente está no canto esquerdo superior da sua tela).

Posso usar o símbolo = no lugar de **<-**? Sim. Funciona. Mas nem sempre e é uma fonte grande de confusão. Quando entendermos um pouco mais da sintaxe da linguagem R ficará claro.

Se o objetivo fosse criar o vetor sem "guardá-lo" em um objeto, bastaria repetir a parte do código acima que começa após o símbolo de aribuição. *c* "concatenate" é a função do R que combina valores de texto, número ou lógicos (ainda não falamos destes últimos) em um vetor. É um função muito utilizada ao programar em R.

```
c("The Guardian", "The Guardian", "The Guardian", "Folha de São Paulo",
  "Folha de São Paulo", "Folha de São Paulo")
```

```
## [1] "The Guardian"       "The Guardian"       "The Guardian"
## [4] "Folha de São Paulo" "Folha de São Paulo" "Folha de São Paulo"
```

Note que há uma quebra de linha no código. Não há problema algum. Uma vez que o parêntese que indica o fim do vetor não foi encontrado, o R entende o que estiver na próxima como continuidade do código (e, portanto, do vetor). Dica: quebras de linha ajudam a vizualizar o código e com o tempo você também usará.

Vamos seguir com nossa tarefa de criar vetores. Já temos o vetor jornal (que você pode observar no workspace). Os demais, na minha coleta de dados, estão a seguir:

```
# Data
# Obs: ha uma funcao em R com nome "data".
# Evite dar a objetos nome de funcoes
date <- c("10/03/2017", "10/03/2017", "10/03/2017", "10/03/2017", "10/03/2017", "10/03/2017")

# Titulo
titulo <- c("'Trump lies all the time': Bernie Sanders indicts president's assault on democracy",
            "Bruno, still guilty of murder but bafflingly welcome to walk back into football",
```

```
            "BBC interviewee interrupted by his children live on air - video",
            "Bolsista negra é hostilizada em atividade no campus da FGV de SP",
            "Meninas podem ser o que quiserem, inclusive matemáticas",
            "Favela de Paraisópolis tem novo incêndio em menos de dez dias")

# Autor
autor <- c("Ed Pilkington ", "Barney Ronay", NA, "Joana Cunha; Jairo Marques", "Marcelo Viana", NA)

# Numero de caracteres
caracteres <- c(5873, 6301, 358, 3644, 4086, 3454)

# Conteudo sobre politica
politica <- c(1, 0, 0, 0, 0, 0)

# Conteudo sobre esporte
esporte <- c(0, 1, 0, 0, 0, 0)

# Contem video
video <- c(FALSE, FALSE, TRUE, FALSE, FALSE, TRUE)
```

Para onde vão os objetos de R criados? Para o workspace. Se quisermos uma fotografia do nosso workspace, usamos a função *ls*, com parêntese vazio (ou seja, sem argumentos além dos pré-estabelecidos):

```
ls()
```

```
## [1] "autor"      "caracteres" "date"       "esporte"    "jornal"
## [6] "politica"   "titulo"     "video"
```

Detalhes importantes nos vetores acima

Mais alguns detalhes importantes a serem notados no exemplo acima:

- O formato da data foi arbitrariamente escolhido. Por enquanto, o R entende apenas como texto o que foi inserido. No datas são especialmente chatas de se trabalhar e há funções específicas para tanto.
- Os textos foram inseridos entre aspas. Os números, não. Se números forem inseridos com aspas o R os entenderá como número também.
- Além de textos e números, temos no vetor *vídeo* valores lógicos, TRUE e FALSE. *logical* é um tipo de dado do R (e é particularmente importante).
- O texto do primeiro título que coletei contém aspas. Como colocar aspas dentro de aspas sem fazer confusão? Se você delimitou o texto com aspas duplas, use aspas simples no texto e vice-versa.
- O que são os *NA* no meio do vetor *autor*? Quando coletei as notícias, não consegui identificar o autor(a) de algumas (eram notícias da redação). *NA* é o símbolo do R para **missing values** e lidar com eles é uma das grandes chatices em R.

Criando um data frame com vetores:

Como vimos acima, *data frames* são um conjunto de vetores horizontais. Se você introduziu os valores em cada vetor na ordem correta de coleta dos dados, então eles podem ser **pareados** e **combinados**. No meu exemplo, a primeira posição de cada vetor contém as informações sobre a primeira notícia, a segunda posição sobre a segunda notícia e assim por diante.

Obviamente, se estiverem pareados, os vetores devem ter o mesmo comprimento. Há uma função bastante útil para checar o comprimento:

```
length(jornal)
```

```
## [1] 6
```

Vamos criar com os vetores que construímos um *data frame* com o nome *dados*. Vamos produzí-lo, discutir a função *data.frame* e depois examiná-lo:

```
dados <- data.frame(jornal, date, titulo, autor, caracteres, politica, esporte, video)
```

A função *data.frame* cria um *data frame* a partir de vetores ou matrizes (que ainda não vimos). Quando criamos a partir de vetores, automaticamente os nomes das variáveis (colunas) no novo objeto serão os nomes dos vetores. Mas poderíamos querer nomes novos (por exemplo, em inglês). Bastaria fazer:

```
dados <- data.frame(newspaper = jornal,
                    date = date,
                    title = titulo,
                    author = autor,
                    n_char = caracteres, politica, esporte, video)
```

Usando as funções que aprendemos nessa aula:

```
# 6 primeiras (e unicas, no caso) linhas
head(dados)
```

```
##            newspaper       date
## 1       The Guardian 10/03/2017
## 2       The Guardian 10/03/2017
## 3       The Guardian 10/03/2017
## 4 Folha de São Paulo 10/03/2017
## 5 Folha de São Paulo 10/03/2017
## 6 Folha de São Paulo 10/03/2017
##                                                                                 title
## 1 'Trump lies all the time': Bernie Sanders indicts president's assault on democracy
## 2    Bruno, still guilty of murder but bafflingly welcome to walk back into football
## 3                    BBC interviewee interrupted by his children live on air - video
## 4                   Bolsista negra é hostilizada em atividade no campus da FGV de SP
## 5                            Meninas podem ser o que quiserem, inclusive matemáticas
## 6                      Favela de Paraisópolis tem novo incêndio em menos de dez dias
##                       author n_char politica esporte video
## 1             Ed Pilkington    5873        1       0 FALSE
## 2               Barney Ronay   6301        0       1 FALSE
## 3                       <NA>    358        0       0  TRUE
## 4 Joana Cunha; Jairo Marques   3644        0       0 FALSE
## 5              Marcelo Viana   4086        0       0 FALSE
## 6                       <NA>   3454        0       0  TRUE
```

```
# Estrutura do data frame
str(dados)
```

```
## 'data.frame':    6 obs. of  8 variables:
##  $ newspaper: Factor w/ 2 levels "Folha de São Paulo",..: 2 2 2 1 1 1
##  $ date     : Factor w/ 1 level "10/03/2017": 1 1 1 1 1 1
##  $ title    : Factor w/ 6 levels "BBC interviewee interrupted by his children live on air - video",.
##  $ author   : Factor w/ 4 levels "Barney Ronay",..: 2 1 NA 3 4 NA
##  $ n_char   : num  5873 6301 358 3644 4086 ...
##  $ politica : num  1 0 0 0 0 0
##  $ esporte  : num  0 1 0 0 0 0
```

```
##  $ video    : logi  FALSE FALSE TRUE FALSE FALSE TRUE
```

```
# Nome das variaveis
names(dados)
```

```
## [1] "newspaper" "date"      "title"     "author"    "n_char"    "politica"
## [7] "esporte"   "video"
```

```
# Dimensoes do data frame
dim(dados)
```

```
## [1] 6 8
```

Tipos de dados em R e vetores

Usando o que aprendemos sobre vetores, vamos examinar com cuidado os tipos de dados que podem ser armazenados em vetores: **doubles**, **integers**, **characters**, **logicals**, **complex**, e **raw**. Neste tutorial, vamos examinar os 3 mais comumente usados na análise de dados: *doubles*, *characters*, *logicals*.

Doubles

doubles são utilizados para guardar números. Por exemplo, o vetor *caracteres*, que indica o número de caracteres em cada texto, é do tipo *double* (ops, desculpe por criar um vetor com este nome para números!). Vamos repetir o comando que cria este vetor:

```
caracteres <- c(5873, 6301, 358, 3644, 4086, 3454)
```

Com a função *typeof* você consegue descobrir o tipo de cada vetor:

```
typeof(caracteres)
```

```
## [1] "double"
```

É possível fazer operações com vetores númericos (*integers* também são vetores numéricos, mas vamos esquecer deles por um segundo e fazer *double* sinônimo de numérico). Por exemplo, podemos somar 1 a todos os seus elementos, dobrar o valor de cada elemento ou somar todos:

```
caracteres + 1
```

```
## [1] 5874 6302  359 3645 4087 3455
```

```
caracteres * 2
```

```
## [1] 11746 12602   716  7288  8172  6908
```

```
sum(caracteres)
```

```
## [1] 23716
```

Note que as duas primeiras funções retornam vetores de tamanho igual ao original, enquanto a aplicação da função *sum* a um vetor retorna apenas um número, ou melhor, um **vetor atômico** (que contém um único elemento). Tal como aplicamos as operações matemáticas e a função *sum*, podemos aplicar diversas outras operações matemáticas e funções que sumarizam os dados (média, desvio padrão, etc) a vetores numéricos. Veremos operações e funções com calma num futuro breve.

Logicals

Os vetores *politica* e *esporte* também são do tipo *double*. Mesmo registrando apenas a presença e ausência de uma característica, os valores inseridos são números. Mas e o vetor *video*? Vejamos:

```
typeof(politica)
```

```
## [1] "double"
```

```
typeof(esporte)
```

```
## [1] "double"
```

```
typeof(video)
```

```
## [1] "logical"
```

Em vez de armazenarmos a sim/não, presença/ausência, etc com os números 0 e 1, podemos em R usar o tipo *logical*, cujos valores são TRUE e FALSE (ou T e F maiúsculos, para poupar tempo e dígitos). Diversas operações e resultados no R usam vetores lógicos (por exemplo, "filtrar uma base dados") e os utilizaremos com frequência.

O que acontece se fizermos operações matemáticas com vetores lógicos?

```
video + 1
```

```
## [1] 1 1 2 1 1 2
```

```
sum(video)
```

```
## [1] 2
```

Automaticamente, o R transforma FALSE em 0 e TRUE em 1. Por exemplo, ao somar 4 FALSE e 2 TRUE obtemos o valor 2, que é o total de notícias que contêm vídeos.

Character

Finalmente, vetores que contêm texto são do tipo *character*. O nome dos jornais, data, título e autor são informações que foram inseridas como texto (lembre-se, o R não sabe por enquanto que no vetor *date* há uma data). Operações matemáticas não valem pare estes vetores.

Tipo e classe

Veremos em momento oportuno que os objetos em R também tem um atributo denominado **class**. A classe diz respeito às características dos objetos enquanto tipo diz respeito ao tipo de dado armazenado. No futuro ignoraremos o tipo e daremos mais atenção à classe, mas é sempre bom saber distinguir os tipos de dados que podemos inserir na memória do computador usando R.

Abrindo dados de fontes externas

R é uma linguagem extremamente flexível quanto ao formato de dados que podem ser abertos. Comumente, utilizaremos dados provenientes de arquivos de texto (.txt, .csv, .tab, etc) ou de arquivos de excel transformados em arquivos de texto. Mas estas estão longe de serem as únicas possibilidades.