R básico

meus primeiros passos no R

R-Ladies Vitória

nov/2019

Programação

- R-Ladies
- Introdução ao R
- R básico
- Importação de dados
- Tratamento de dados
- Medidas descritivas

- R-Ladies é uma organização mundial que **promove a diversidade de gênero** na comunidade R.
- Capacitar pessoas de gêneros sub-representados, criando e fortalecendo redes colaborativas dentro da comunidade R para que elas alcancem todas e quaisquer funções e áreas de participação no mundo da tecnologia.

Como

- Promovendo meetups (encontros) e mentorias.
- Garantindo espaço amigável e seguro.

O Capítulo da cidade de Vitória foi criado em 29 de setembro de 2019.

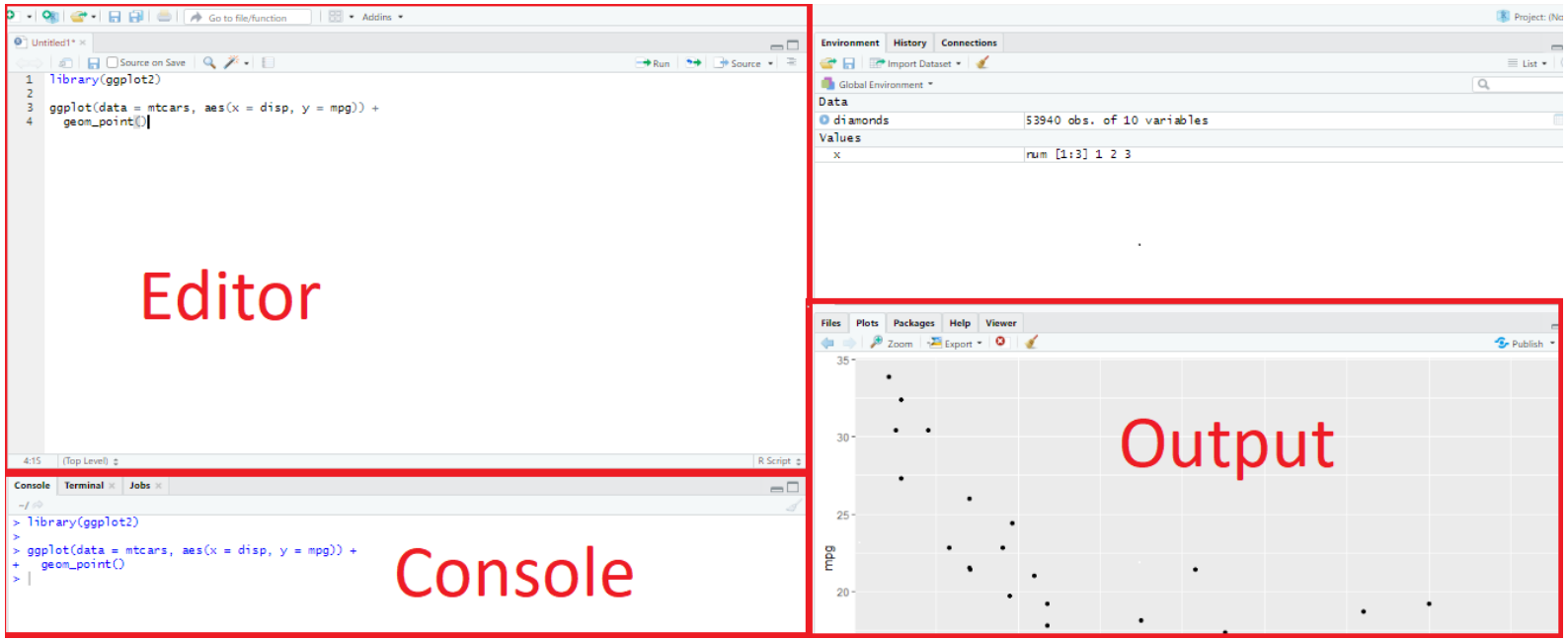
- Código de conduta - R-Ladies

- Saiba mais:
  - RLadies Global: https://rladies.org/
  - MeetUp: https://www.meetup.com/pt-BR/R-Ladies-Vitoria
  - Twitter: @RLadiesGlobal, @rladiesvix
  - Instagram: @RLadiesVix
  - Github: https://github.com/rladies/meetup-presentations_vitoria

- O R é uma linguagem de programação, além de um ambiente de software gratuito.
- oferece um vasto leque de funcionalidades acessíveis via instalação de bibliotecas.
- o R possui uma comunidade extremamente ativa, engajada desde o aprimoramento da ferramenta e desenvolvimento de novas bibliotecas, até o suporte aos usuários.
- Saiba mais em: r-project.org

- Optar por programar em R também implica na escolha de uma IDE (Integrated Development Environment) que, na grande maioria dos casos, será o RStudio.
- O RStudio é um conjunto de ferramentas integradas projetadas (IDE - Integrated Development Environment) da linguagem R para editar e executar os códigos em R.
- O R, em combinação com o RStudio, possui um conjunto de funcionalidades cuja intenção é ajudar no processo de desenvolvimento.
- Saiba mais em: https://rstudio.com/

RStudio


Editor
Console
Output
library(ggplot2)
ggplot(data = mtcars, aes(x = disp, y = mpg)) +
geom_point()
> library(ggplot2)
> ggplot(data = mtcars, aes(x = disp, y = mpg)) +
+ geom_point()
Environment
History
Connections
Data
diamonds
53940 obs. of 10 variables
Values
x
num [1:3] 1 2 3
Files
Plots
Packages
Help
Viewer
mpg

RStudio

- Editor/Scripts: É onde escrever os códigos. Arquivos do tipo .R.
- Console: Executar os comandos e ver os resultados.
- Enviroment: Painel com todos os objetos criados.
- History: História dos comandos executados.
- Files: Navegar em pastas e arquivos.
- Plots: Onde os gráficos serão apresentados.
- Packages: Pacotes instalados (sem ticar) e habilitados (ticados).
- Help: Retorna o tutorial de ajuda do comando solicitado com help() ou ?comando. Veremos melhor como pedir ajuda no R ainda nessa aula.

Rproj e diretórios

- Organizar arquivos é uma parte integral do processo de programação.
- Denominados “projetos”, eles não passam de pastas comuns com um arquivo .Rproj.

Uma funcionalidade importante é a criação de projetos, permitindo dividir o trabalho em múltiplos ambientes, cada um com o seu diretório, documentos e workspace.

Para criar um projeto, os seguintes passos podem ser seguidos:

1) Clique na opção “File” do menu, e então em “New Project”.

2) Clique em “New Directory”.

3) Clique em “New Project”.

4) Escreva o nome do diretório (pasta) onde deseja manter seu projeto, ex “my_project”.

5) Clique no botão “Create Project”.

Para criar um novo script para escrever os códigos, vá em File -> New File -> R Script

Boas práticas

- Comente bem o seu código: É possível fazer comentários usando o símbolo '#'. É sempre bom explicar o que uma variável armazena, o que uma função faz, porque alguns parâmetros são passados para uma determinada função, qual é o objetivo de um trecho de código, etc.
- Evite linhas de código muito longas: Usar linhas de código mais curtas ajuda na leitura do código.
- Escreva um código organizado: Por exemplo, adote um padrão no uso de minúsculas e maiúsculas, uma lógica única na organização de pastas e arquivos, pode ser adotada uma breve descrição (como comentário) indicando o que um determinado script faz.
- Carregue todos os pacotes que irá usar sempre no início do arquivo: Quando alguém abrir o seu código será fácil identificar quais são os pacotes que devem ser instalados e quais dependências podem existir.
- Evite referência de caminho que considere seu computador ou usuário: Faça referência ao caminho do projeto.

Shall we?

Ilustração por Allison Horst - Twitter: @allison_horst

R como calculadora

```
#adição
10+15
  #> [1] 25
#subtração
10-2
  #> [1] 8
#multiplicação
2*10
  #> [1] 20
#divisão
30/2
  #> [1] 15
#raiz quadrada
sqrt(4)
  #> [1] 2
#potência
2^2
  #> [1] 4
```

Se você digitar um comando incompleto, como 10 *, o R mostrará um +. Isso não tem a ver com a soma e apenas que o

Atribuição

Para atribuir a um objeto, o sinal de atribuição é = ou <-. Exemplos:

```
x <- 10/2
x
  #> [1] 5
X
  #> Error in eval(expr, envir, enclos): objeto 'X' não encontrado
```

Por que tivemos um erro acima?

O R é case sensitive, isto é, faz a diferenciação entre as letras minúsculas e maiúsculas. Portanto, x é diferente de X.

Objetos em R

Existem cinco classes básicas no R:

- character: "UAH!"
- numeric: 0.95 (números reais)
- integer: 100515 (inteiros)
- complex: 2 + 5i (números complexos, a + bi)
- logical: TRUE (booleanos, TRUE/FALSE)

Vamos atribuir a x a string banana.

```
x <- banana
  #> Error in eval(expr, envir, enclos): objeto 'banana' não encontrado
x <- "banana"
x
  #> [1] "banana"
```

O primeiro caso (x<-banana) não deu certo, pois ele entendeu que estamos atribuindo a x outro objeto banana, que não foi declarado. Para atribuir o string banana à x, precisamos colocar entre aspas ou aspas simples. Uma string sem aspas é entendido como um objeto, veja abaixo:

```
banana <- 30
x <- banana
x
  #> [1] 30
```

Função class().

```
y <- "ola"
class(y)
  #> [1] "character"

x <- 2.5
class(x)
  #> [1] "numeric"
```

Apagar objetos

```
x <- 20
x
  #> [1] 20
remove(x)
x
  #> Error in eval(expr, envir, enclos): objeto 'x' não encontrado
```

E se eu quiser limpar o console - apaga todos os objetos atribuidos até aqui:

```
rm(list=ls())
```

Data Strutures

- atomic vector
- matrix
- factors
- data frame
- list

Vetor

```
x <- c(2,3,4)
x
  #> [1] 2 3 4
y <- seq(1:10)
y
  #>  [1]  1  2  3  4  5  6  7  8  9 10
z <- rep(1,10)
z
  #>  [1] 1 1 1 1 1 1 1 1 1 1
a <- 1:10
a
  #>  [1]  1  2  3  4  5  6  7  8  9 10
bicho <-c("macaco","pato","galinha","porco")
bicho
  #> [1] "macaco"  "pato"    "galinha" "porco"
```

E se quisermos visualizar o conteúdo da posição 2 no vetor bicho?

```
bicho[2]
  #> [1] "pato"
```

Operações vetoriais

```
k <- x*2
y <- c(x,k)
y
  #> [1] 2 3 4 4 6 8
```

Como calcularia o IMC de 6 pessoas?

```
peso <- c(62, 70, 52, 98, 90, 70)
altura <- c(1.70, 1.82, 1.75, 1.94, 1.84, 1.61)
imc <- peso/(altura^2)
imc
  #> [1] 21.45329 21.13271 16.97959 26.03890 26.58318 27.00513
```

Função length().

```
length(imc)
  #> [1] 6
```

Matrizes

```r
x <- matrix(seq(1:16), nrow=4,ncol=4)
x
  #>      [,1] [,2] [,3] [,4]
  #> [1,]    1    5    9   13
  #> [2,]    2    6   10   14
  #> [3,]    3    7   11   15
  #> [4,]    4    8   12   16
x[2,3] #retorna o elemento na segunda linha e terceira coluna da matriz
  #> [1] 10
x[3,  ]   # seleciona a 3ª linha
  #> [1]  3  7 11 15
x[ , 2]   # seleciona a 2ª coluna
  #> [1] 5 6 7 8
x[1,] <- c(13,15,19,30)  #substituir a primeira linha por (13,15,19,30)

x
  #>      [,1] [,2] [,3] [,4]
  #> [1,]   13   15   19   30
  #> [2,]    2    6   10   14
  #> [3,]    3    7   11   15
  #> [4,]    4    8   12   16
```

Concatenar linhas em uma matriz

```
vet <- c(2,20,12,34)
x2 <- rbind(x,vet)
x2
  #>     [,1] [,2] [,3] [,4]
  #>       13   15   19   30
  #>        2    6   10   14
  #>        3    7   11   15
  #>        4    8   12   16
  #> vet    2   20   12   34
```

Concatenar colunas em uma matriz

```
v2 <- c(25,10,15,4)
x3 <- cbind(x,v2)
x3
  #>                 v2
  #> [1,] 13 15 19 30 25
  #> [2,]  2  6 10 14 10
  #> [3,]  3  7 11 15 15
  #> [4,]  4  8 12 16  4
```

Operações matriciais

```
xa <- x2[1:2,1:2]
xb <- matrix(rnorm(4),2,2)
xa*xb #multiplicacao ponto a ponto
  #>           [,1]       [,2]
  #>  -21.28309198 -0.2643236
  #>    0.06628563  5.8511509
xa%*%xb #multiplicacao matricial
  #>         [,1]      [,2]
  #>  -20.785950 14.398797
  #>   -3.075465  5.815908
solve(xa) #inversa de xa
  #>
  #> [1,]  0.12500000 -0.3125000
  #> [2,] -0.04166667  0.2708333
diag(xa) #matriz diagonal
  #> [1] 13  6
```

Data frame

Trata-se de uma "tabela de dados" onde as colunas são as variáveis e as linhas são os registros. Essas colunas podem ser de classes diferentes. Essa é a grande diferença entre data.frame's e matrizes (matriz é só numerica).

Posso criar um data frame no R com os vetores, por exemplo:

```
ID <- seq(1:6)
pes <- c(62, 70, 52, 98, 90, 70)
alt <- c(1.70, 1.82, 1.75, 1.94, 1.84, 1.61)
imc <- pes/(alt^2)
dados <- data.frame(ID=ID,peso=pes,altura=alt, imc=imc)
dados
  #>   ID peso altura      imc
  #> 1  1   62   1.70 21.45329
  #> 2  2   70   1.82 21.13271
  #> 3  3   52   1.75 16.97959
  #> 4  4   98   1.94 26.03890
  #> 5  5   90   1.84 26.58318
  #> 6  6   70   1.61 27.00513
```

Selecionar a variavel de interesse:

```
dados$altura
  #> [1] 1.70 1.82 1.75 1.94 1.84 1.61
```

Putz, esqueci de colocar a varivel de grupo no data frame. Tenho que criar tudo de novo? Não:

```
gr <- c(rep(1,3),rep(2,3))
dados$grupo <- gr

dados
  #>   ID peso altura      imc grupo
  #> 1  1   62   1.70 21.45329     1
  #> 2  2   70   1.82 21.13271     1
  #> 3  3   52   1.75 16.97959     1
  #> 4  4   98   1.94 26.03890     2
  #> 5  5   90   1.84 26.58318     2
  #> 6  6   70   1.61 27.00513     2
```

Funcoes uteis para data.frame:

- head() - Mostra as primeiras 6 linhas.
- tail() - Mostra as últimas 6 linhas.
- dim() - Número de linhas e de colunas.
- names() - Os nomes das colunas (variáveis).
- str() - Estrutura do data.frame. Mostra, entre outras coisas, as classes de cada coluna.

```
names(dados)
  #> [1] "ID"     "peso"   "altura" "imc"    "grupo"
str(dados)
  #> 'data.frame':    6 obs. of  5 variables:
  #>  $ ID    : int  1 2 3 4 5 6
  #>  $ peso  : num  62 70 52 98 90 70
  #>  $ altura: num  1.7 1.82 1.75 1.94 1.84 1.61
  #>  $ imc   : num  21.5 21.1 17 26 26.6 ...
  #>  $ grupo : num  1 1 1 2 2 2
```

```
mean(dados$imc)
  #> [1] 23.1988
sd(dados$imc)
  #> [1] 4.00006
summary(dados$imc)
  #>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  #>   16.98   21.21   23.75   23.20   26.45   27.01
```

```
table(dados$grupo)
  #>
  #> 1 2
  #> 3 3
```

Fator

```
sexo <- c("M", "H", "H", "H", "M", "M", "H")
sex <- as.factor(sexo)
sex
  #> [1] M H H H M M H
  #> Levels: H M
levels(sex)
  #> [1] "H" "M"
```

Arrays

```
dim3 <- array(rnorm(18),dim = c(3,3,2))
dim3
  #> , , 1
  #>
  #>           [,1]       [,2]      [,3]
  #> [1,] -1.463941  0.1454567 -1.071631
  #> [2,]  1.401086 -0.9751037  1.432764
  #> [3,] -1.264694 -0.1558900  1.163923
  #>
  #> , , 2
  #>
  #>           [,1]       [,2]        [,3]
  #> [1,]  0.285915 -0.2138941 -0.08158459
  #> [2,] -3.396967 -0.2406444  1.51794199
  #> [3,] -1.864337 -1.9072540 -1.20950424
```

Lists

```
ls <- list(ls1 = 'a',ls2 = c(1,2,3),ls3 = array(rnorm(6),dim = c(3,1,2,1)))
ls
  #> $ls1
  #> [1] "a"
  #>
  #> $ls2
  #> [1] 1 2 3
  #>
  #> $ls3
  #> , , 1, 1
  #>
  #>            [,1]
  #> [1,]  0.5329555
  #> [2,] -1.6518211
  #> [3,]  0.8040427
  #>
  #> , , 2, 1
  #>
  #>            [,1]
  #> [1,] -0.8405715
  #> [2,]  0.5840419
```

Operadores Relacionais

Igual a: ==

```
10==11
  #> [1] FALSE
```

Diferente de: !=

```
10!=11
  #> [1] TRUE
```

- Maior que: >
- Maior ou igual: >=
- Menor que: <
- Menor ou igual: <=

Operadores Lógicos

- E: & - será verdadeiro se os dois forem TRUE

```
x <- 15
x > 10 & x < 30
  #> [1] TRUE

x < 10 & x < 30
  #> [1] FALSE
```

- OU: | - será verdadeiro se um dos dois for TRUE

```
x > 10 | x > 30
  #> [1] TRUE
```

- Negação: !

```
x <- 15
!x<30

  #> [1] FALSE
```

If e else

```
a <- 224
b <- 225
if (a==b) { v=10
} else {v=15}
v
  #> [1] 15
```

```
a <- 224
b <- 225
if (a==b) { v=10
} else if (a > b) {v=15
} else {v=25}
v
  #> [1] 25
```

Note que a condição de igualdade é representada por dois iguais ==. Como dito anteriormente, apenas um igual = é símbolo de atribuição.

For

```
m <- c(1,20,50,60,100)
```

Quero criar um novo vetor, p digamos, que seja formado por cada elemento de m dividido por sua posição.

```
p <- rep(0,length(m))
for (i in 1: length(m)){
  p[i] <- m[i]/i
}
p
  #> [1]  1.00000 10.00000 16.66667 15.00000 20.00000
```

Note que primeiro definimos o objeto p.

Funções

```
f.soma <- function(x,y) {
  out <- x+y
  return(out)
}
```

- o nome: f.soma;
- os argumentos: x e y;
- o corpo out <- x+y e
- o que retorna return(out).

Vamos agora chamar a função:

```
f.soma(x=10,y=20)
  #> [1] 30

f.soma(10,20)
  #> [1] 30
```

Dados faltantes, infinitos e indefinições matemáticas

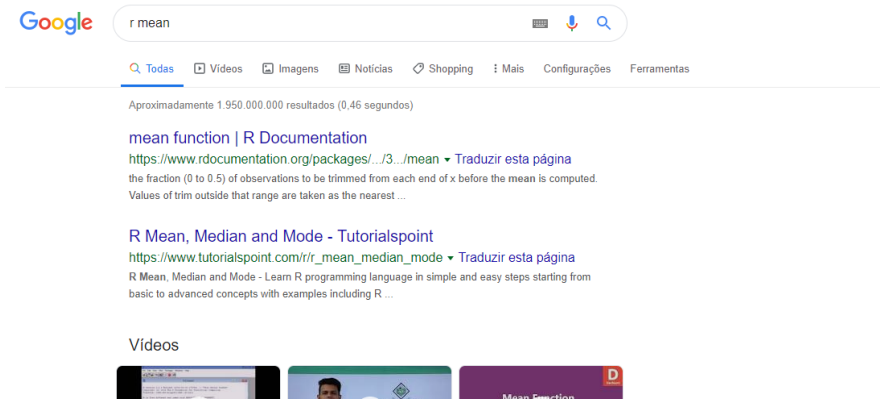
- NA (Not Available): dado faltante/indisponível.
- NaN (Not a Number): indefinições matemáticas. Como 0/0 e log(-1).
- Inf (Infinito): número muito grande ou o limite matemático. Aceita sinal negativo -Inf.

Como obter ajuda no R

- Help/documentação do R

```
help(mean) #ou
?mean
```

- Google.



- Comunidade.

Stack Overflow e o Stack Overflow em Português.

Pacotes

Instalação

- Via CRAN: install.packages("nome-do-pacote").

install.packages("tidyverse")

Note que o nome do pacote está entre aspas.

- Via Github: devtools::install_github("nome-do-repo/nome-do-pacote").

devtools::install_github("tidyverse/dplyr")

Carregar pacotes:

library(nome-do-pacote)

Para carregar o pacote, não usar aspas.

Só é necessário instalar o pacote uma vez, mas precisa carregá-lo toda vez que começar uma nova sessão.

Importação de dados

Extensão .txt ou .csv

Opção com o pacote **readr**.

```
library(readr) #pacote readr
dados_csv <- read_csv(file = "dados1.csv")
dados_txt <- read_delim(file = "caminho-para-o-arquivo/dados1.txt", delim = " ")
```

Outra opção:

```
dados_txt2 <- read.table(file="dados1.txt",header=T)
```

Vale ressaltar que para cada função read*, existe uma respectiva função write* para salvar o arquivo no formato de interesse. Como exemplo, queremos salvar a base de dados cars.

```
write_csv(x = mtcars, path = "cars.csv")
write_delim(x = mtcars, delim = " ", path = "cars.txt"))
```

Arquivos em Excel

O pacote **readxl** pode ser utilizado para leiturade arquivos do Excel, como .xls e xlsx.

```
library(readxl)
dados_excel <- read_xls(path = "dados1.xls") #Leitura do arquivo .xls
dados_excelx <- read_xlsx(path = "dados1.xlsx") #Leitura do arquivo .xlsx
```

Uma maneira mais simples é a utilização da função read_excel(), pois ela auto detecta a extensão do arquivo.

```
library(readxl)
dados_excel1 <- read_excel(path = "dados1.xls")
dados_excelx1 <- read_excel(path = "dados1.xlsx")
```

Arquivos de outros softwares

```
library(haven)
dados_stata <- read_stata("dados1.dta")
dados_spss <- read_spss("dados1.sav")

dados_sas <- read_sas("dados1.sas7bdat")
```

Tratamento de dados

A análise de consistência consiste em realizar uma primeira análise dos dados com o intuito de encontrar inconsistências.

- boas práticas para nome das variáveis.
- identificar erros de digitação;
- indivíduos imputados mais de uma vez na planilha de dados de maneira errada;
- identificar casos missings e avaliar se a observação está ausente de maneira correta ou não;
- identificar as categorias de variáveis qualitativas.

Consideramos como exemplo os dados fictícios de $n = 30$ gestações gemelares. Vamos considerar duas bases de dados. Na primeira, chamada de "Base_CTG_caracterizacao" estão contidas as informações de caracterização das gestantes e gestações.

Para importar a base de dados:

```
library(readxl)
dados <- read_excel(path = "Base_CTG_caracterizacao.xls",na="NA")
dados
  #> # A tibble: 34 x 13
  #>       ID CORION `Data aval`         `Data nascimento`   `COR BRANCO`

  #>    <dbl> <chr>  <dttm>             <dttm>                     <dbl>
```

Exercício

Na base em excel, retire os NA's, deixando em branco, e rode o seguinte comando:

```
library(readxl)
dados <- read_excel(path = "Base_CTG_caracterizacao.xls")
```

O default do missing é o espaço em branco. Acesse o help em ?read_excel e veja na = "".

Nome das variáveis

Utilizaremos as funções tidyverse e janitor para a arrumação da base de dados.

```
library(tidyverse)
library(janitor)

names(dados)
  #>  [1] "ID"              "CORION"          "Data aval"
  #>  [4] "Data nascimento" "COR BRANCO"      "Peso Pré"
  #>  [7] "ALT"             "Gesta"           "Para"
  #> [10] "Aborto"          "IGP semana"      "IGP dia"
  #> [13] "oi"
dados <- clean_names(dados) # a função clean_names() para primeiro ajuste dos nomes das variaveis
names(dados)
  #>  [1] "id"              "corion"          "data_aval"
  #>  [4] "data_nascimento" "cor_branco"      "peso_pre"
  #>  [7] "alt"             "gesta"           "para"
  #> [10] "aborto"          "igp_semana"      "igp_dia"
  #> [13] "oi"
```

Linhas e colunas vazias

Na base de dados em questão, não há linhas vazias, como pode ser visto na saída abaixo.

```
dados <- remove_empty(dados,"rows")
```

Propositalmente, inclui a coluna "oi" vazia para podermos eliminá-la com o comando abaixo:

```
dados <- remove_empty(dados,"cols")
names(dados)
  #>  [1] "id"              "corion"          "data_aval"
  #>  [4] "data_nascimento" "cor_branco"      "peso_pre"
  #>  [7] "alt"             "gesta"           "para"
  #> [10] "aborto"          "igp_semana"      "igp_dia"
```

Identificação de casos duplicados

```
get_dupes(dados, id)
  #> # A tibble: 8 x 13
  #>      id dupe_count corion data_aval           data_nascimento
  #>   <dbl>      <int> <chr>  <dttm>              <dttm>
  #> 1     2          2 Mono   2016-03-21 00:00:00 1982-03-30 00:00:00
  #> 2     2          2 Mono   2016-03-21 00:00:00 1982-03-30 00:00:00
  #> 3    11          2 Di     2016-02-17 00:00:00 1981-02-25 00:00:00
  #> 4    11          2 Di     2016-02-17 00:00:00 1981-02-25 00:00:00
  #> 5    17          2 Di     2017-04-23 00:00:00 1993-04-29 00:00:00
  #> 6    17          2 Di     2017-04-23 00:00:00 1993-04-29 00:00:00
  #> 7    23          2 Di     2016-02-17 00:00:00 1997-02-21 00:00:00
  #> 8    23          2 Di     2016-02-17 00:00:00 1997-02-21 00:00:00
  #> # ... with 8 more variables: cor_branco <dbl>, peso_pre <dbl>, alt <dbl>,
  #> #   gesta <dbl>, para <dbl>, aborto <dbl>, igp_semana <dbl>, igp_dia <dbl>
```

```
library(dplyr)
dados1 <-  distinct(dados,id, .keep_all = TRUE)
dados1
  #> # A tibble: 30 x 12
  #>       id corion data_aval           data_nascimento     cor_branco peso_pre
```

Identificar tipo e classe de todas as variáveis da base

Para identificar a classe de todas as variáveis que o R está interpretando, usamos a função str(.).

```
# Ver a estrutura dos dados
str(dados1)
  #> Classes 'tbl_df', 'tbl' and 'data.frame':    30 obs. of  12 variables:
  #>  $ id              : num  1 2 3 4 5 6 7 8 9 10 ...
  #>  $ corion          : chr  "Di" "Mono" "Di" "Di" ...
  #>  $ data_aval       : POSIXct, format: "2017-04-23" "2016-03-21" ...
  #>  $ data_nascimento: POSIXct, format: "1988-04-30" "1982-03-30" ...
  #>  $ cor_branco      : num  1 1 0 1 0 0 0 1 3 1 ...
  #>  $ peso_pre        : num  93 59 87 52 78 62 54 72 72 102 ...
  #>  $ alt             : num  1.63 1.45 1.69 1.55 1.59 1.64 NA 1.64 1.65 1.68 ...
  #>  $ gesta           : num  3 3 2 2 7 7 2 2 8 3 ...
  #>  $ para            : num  1 1 1 0 2 4 1 1 4 2 ...
  #>  $ aborto          : num  1 1 0 1 4 2 0 0 3 0 ...
  #>  $ igp_semana      : num  37 37 35 38 38 29 38 34 36 37 ...
  #>  $ igp_dia         : num  4 3 0 1 3 3 3 6 3 5 ...
```

```
dados1$data_aval  <- as.Date(dados1$data_aval)

dados1$data_nascimento  <- as.Date(dados1$data_nascimento)
```

Identificar tipo e classe de todas as variáveis da base

Utilizamos a função as.Date(.) porque queríamos mudar para tipo data. Abaixo está a lista das funções para mudança de tipo.

- as.character - converte para variável texto.
- as.numeric - converte para variável número.
- as.factor - converte para variável categórica.
- as.integer - converte para variável inteiro.
- as.Date - converte para variável data.
- as.POSIXct - converte para variável data e hora completa.

Identificar erros

Para variáveis qualitativas: tabela de frequências da variável corion.

```
#do pacote janitor
tabyl(dados1,corion)
  #>  corion  n    percent
  #>      Di 25 0.83333333
  #>      DI  1 0.03333333
  #>    Mono  3 0.10000000
  #>    MONO  1 0.03333333
```

Para lidar com variáveis de texto, vamos utilizar a função str_to_lower do pacote stringr.

```
library(stringr)
dados1$corion <- str_to_lower(dados1$corion)
tabyl(dados1,corion)
  #>  corion  n   percent
  #>      di 26 0.8666667
  #>    mono  4 0.1333333
```

A variável indicadora de cor branca (cor_branco) está categorizada como 0 para não e 1 para sim.

```
tabyl(dados1,cor_branco)
  #>  cor_branco  n    percent
  #>           0 12 0.40000000
  #>           1 17 0.56666667
  #>           3  1 0.03333333
```

```
dados1$cor_branco <- ifelse(dados1$id==9,1,dados1$cor_branco)
```

No R tem um pacote só para manipular fatores: o **forcats** (for categorial variables).

Primeiro, precisamos informar o R que a variável é fator, com o comando as.factor(.).

```
library(forcats)
dados1$cor_branco  <- as.factor(dados1$cor_branco)

dados1$cor_branco <- fct_recode(dados1$cor_branco,
           branco = "1",
           nbranco = "0")

tabyl(dados1,cor_branco)
```

Para fazer análise geral de todas as variáveis da base de dados, usamos a função skim(.) do pacote **skimr**.

```
library(skimr)
kable(skim(dados1))
  #> Skim summary statistics
  #>  n obs: 30
  #>  n variables: 12
  #>
  #> Variable type: character
  #>
  #>  variable    missing    complete    n     min    max    empty    n_unique
  #> ----------  ---------  ----------  ----  -----  -----  -------  ----------
  #>   corion         0          30       30     2      4       0         2
  #>
  #> Variable type: Date
  #>
  #>     variable        missing    complete    n        min           max           median       n_uniqu
  #> -----------------  ---------  ----------  ----  ------------  ------------  ------------  --------
  #>     data_aval          0          30       30    2016-02-17    2017-12-14    2016-10-06        4
  #>  data_nascimento       0          30       30    1980-02-26    1998-12-19    1988-04-30        23
  #>
  #> Variable type: factor
  #>
```

Transformação de variáveis quantitativas

Calcular IMC (índice de massa corpórea) - peso (em km) dividido pela altura (em metros) ao quadrado.

```
dados1 <- mutate(dados1,imc = peso_pre/(alt^2))
kable(skim(dados1,imc))
  #> Skim summary statistics
  #>  n obs: 30
  #>  n variables: 13
  #>
  #> Variable type: numeric
  #>
  #>  variable    missing    complete    n     mean     sd      p0       p25      p50      p75     p100
  #> ----------  ---------  ----------  ----  ------  ------  -------  -------  -------  -------  -----
  #>    imc          2          28       30    26.6    5.95    16.36    22.12    25.97    29.32    40.6
str(dados1)
  #> Classes 'tbl_df', 'tbl' and 'data.frame':    30 obs. of  13 variables:
  #>  $ id             : num  1 2 3 4 5 6 7 8 9 10 ...
  #>  $ corion         : chr  "di" "mono" "di" "di" ...
  #>  $ data_aval      : Date, format: "2017-04-23" "2016-03-21" ...
  #>  $ data_nascimento: Date, format: "1988-04-30" "1982-03-30" ...
  #>  $ cor_branco     : Factor w/ 2 levels "nbranco","branco": 2 2 1 2 1 1 1 2 2 2 ...
  #>  $ peso_pre       : num  93 59 87 52 78 62 54 72 72 102 ...
```

Exercício

Crie a variável igp (idade gestacional do parto) em semanas - obtida ao somar igp_semana e igp_dia/7.

Transformação de variáveis qualitativas

A variável "gesta" indica o número de gestações, contando com a atual. Logo, uma gestante com gesta=1 está em sua primeira gestação, ou seja, é primigesta. Queremos criar uma nova variável indicadora de gestação primigesta. Há diferentes forma de fazer isso. Vamos usar o comando ifelse já utilizado anteriormente.

```
dados1$primigesta <- ifelse(dados1$para==1,1,0)
tabyl(dados1,primigesta)
  #>  primigesta  n percent
  #>           0 21     0.7
  #>           1  9     0.3
```

Agora vamos recodificar primigesta com o nome de cada categoria:

```
dados1$primigesta <- as.factor(dados1$primigesta)
dados1$primigesta <- fct_recode(dados1$primigesta,
           nao = "0",
           sim = "1")
tabyl(dados1,primigesta)
  #>  primigesta  n percent
  #>         nao 21     0.7
  #>         sim  9     0.3
```

Exercício:

1) Crie a variável indicador_aborto (sim e nao) - sim se aborto >=1 e nao se aborto=0.

2) Crie a variável primipara (sim e nao) - sim se para >=1 e nao se para=0.

Diferença de datas

Vamos calcular a idade das pacientes (data da avaliação e data do nascimento). Para realizar operações com data, usaremos o pacote **lubridate**.

A data está salva no formato ano-mês-dia e por isso usamos a função ymd(.) para as variáveis de data. Para calcular a diferença entre as data, usamos a função , atribuindo ao objeto intervalo. Por fim, obtemos a idade ao dividir o intervalo por ano.

```
library(lubridate)
intervalo <- ymd(dados1$data_nascimento) %--%  ymd(dados1$data_aval)

dados1$idade <- intervalo / dyears(1)  #número de anos

kable(skim(dados1,idade))
  #> Skim summary statistics
  #>  n obs: 30
  #>  n variables: 15
  #>
  #> Variable type: numeric
  #>
  #>  variable    missing    complete    n     mean     sd     p0     p25     p50     p75     p100
  #> ----------  ---------  ----------  ----  ------  ------  ----  -------  -----  -------  ------  --
  #>   idade         0          30       30    27.9    6.05    18    23.25    29     33.75     37     <
```

Combinação de bases de dados

Agora vamos considerar a segunda base de dados. Essa base de dados contém novas variáveis para os mesmos $n = 30$ gestantes, identificadas pela variável chave "id". Vamos então ler a base de dados, atribuindo para o objeto "dados.ctg".

```
dados.ctg <- read_excel(path = "Base_CTG_NumContraeColo.xls")

str(dados.ctg)
  #> Classes 'tbl_df', 'tbl' and 'data.frame':    30 obs. of  5 variables:
  #>  $ ID            : num  1 2 3 4 5 6 7 8 9 10 ...
  #>  $ Grupo         : num  2 1 1 1 2 1 2 1 1 2 ...
  #>  $ IG_Aval       : num  33.7 33.4 30.9 33 32.9 ...
  #>  $ MedidaColo    : num  33 34.6 25 27 32.8 20.6 33.1 28 21.8 46.3 ...
  #>  $ Num_contra_CTG: num  9 4 5 6 5 2 1 3 2 8 ...
```

EXERCÍCIO:

Realize o tratamento da base de dados "dados.ctg".

Há algumas funções de combinação de duas bases de dados no pacote **dplyr**. Aqui estão as mais úteis:

- inner_join () - retorna valores de ambas as tabelas somente onde há uma correspondência.
- left_join () - retorna todos os valores da primeira tabela mencionada, mais os da segunda tabela correspondente.
- semi_join () - filtra a primeira tabela mencionada para incluir apenas os valores que possuem correspondências na segunda tabela.
- anti_join () - filtra a primeira tabela mencionada para incluir apenas valores que não possuem correspondências na segunda tabela.

```r
dados.todos <- inner_join(dados1, dados.ctg, by=c("id" = "ID"))

dados.todos
```

```
#> # A tibble: 30 x 19
#>       id corion data_aval  data_nascimento cor_branco peso_pre   alt gesta
#>    <dbl> <chr>  <date>     <date>          <fct>         <dbl> <dbl> <dbl>
#>  1     1 di     2017-04-23 1988-04-30      branco           93  1.63     3
#>  2     2 mono   2016-03-21 1982-03-30      branco           59  1.45     3
#>  3     3 di     2016-02-17 1991-02-23      nbranco          87  1.69     2
#>  4     4 di     2017-12-14 1983-12-23      branco           52  1.55     2
#>  5     5 di     2017-04-23 1988-04-30      nbranco          78  1.59     7
#>  6     6 di     2016-03-21 1989-03-28      nbranco          62  1.64     7
#>  7     7 di     2016-02-17 1985-02-24      nbranco          54 NA        2
#>  8     8 di     2017-12-14 1988-12-21      branco           72  1.64     2
#>  9     9 di     2017-04-23 1980-05-02      branco           72  1.65     8
#> 10    10 di     2016-03-21 1984-03-29      branco          102  1.68     3
#> # ... with 20 more rows, and 11 more variables: para <dbl>, aborto <dbl>,
#> #   igp_semana <dbl>, igp_dia <dbl>, imc <dbl>, primigesta <fct>,
#> #   idade <dbl>, Grupo <dbl>, IG_Aval <dbl>, MedidaColo <dbl>,
#> #   Num_contra_CTG <dbl>
```

Imagem de Curso-R

Obrigada!

- Contato:
  - Website RLadies Global: https://rladies.org/
  - MeetUp: https://www.meetup.com/pt-BR/R-Ladies-Vitoria
  - Twitter: @RLadiesGlobal, @rladiesvix
  - Instagram: @RLadiesVix
  - Github: https://github.com/rladies/meetup-presentations_vitoria
  - Email: vitoria@rladies.org