Aula 2: Introdução à Linguagem R

Introdução à Ciência dos Dados 2023 Mestrado Profissional em Administração

Seu nome

## Configurações Globais e Pacotes Necessários

knitr::opts\_chunk$set(digits = 4, scipen = 999, warning = FALSE, message = FALSE)  
  
# pacotes utilizados  
library(MASS)  
library(dplyr)  
library(skimr)  
library(gapminder)

## Usando R como uma calculadora

2 + 18 # adição

## [1] 20

2 - 18 # subtração

## [1] -16

50821/6 # divisão

## [1] 8470.167

21\*4 # multiplicação

## [1] 84

exp(10) # função e^10

## [1] 22026.47

log(10) # log natural de 10

## [1] 2.302585

choose(10, 8) # comb(n,k)

## [1] 45

factorial(100) # !100

## [1] 9.332622e+157

cos(pi) # cosseno

## [1] -1

sin(pi) # tangente

## [1] 1.224647e-16

round(pi, digits=2) # arredondamento

## [1] 3.14

### Precedência de operações

1/200\*30 # (1/200)\*30 - divisao tem precedencia

## [1] 0.15

1/(200\*30) # deve-se usar parenteses para definir a precedencia

## [1] 0.0001666667

definindo a precedencia com parênteses:

(59 + 73 + 2)/3

## [1] 44.66667

## Variaveis e Atribuição de valores

Variáveis não precisam ser declaradas previamente, são definidas e alteradas com as operações realizadas para criá-las (tipagem dinâmica).

É uma boa prática atribuir valores às variáveis criadas usando o operador <-.

No Windows, um atalho para inseri-lo é dado pela seguinte combinação de teclas: Alt + (-):

x <- 3\*4 # boa pratica  
x

## [1] 12

Não é uma boa prática atribuir valores a uma variável utilizando =:

x = 5\*9 # pratica ruim  
x

## [1] 45

## Sintáxe da Linguagem

(r\_rocks <- 2^3)

## [1] 8

Qual o problema com os códigos abaixo?

r\_rock

R\_rocks

A linguagem R é sensível ao caso, ou seja, letras minúsculas e maiúsculas representam objetos diferentes.

Usando o nome correto do objeto:

r\_rocks

## [1] 8

## Operadores Lógicos

A seguir, são apresentados exemplos de operadores lógicos, ou seja, operadores que retornam TRUE ou FALSE.

Podemos criar e atribuir valores a diferentes variáveis em uma mesma linha de usando ; para separar as operações

a <- 5; b <- 7

Se quisermos ver os valores atribuídos às variáveis, basta envolvermos cada uma das expressões com parênteses:

(a <- 5); (b <- 7)

## [1] 5

## [1] 7

Vejamos alguns exemplos de testes lógicos envolvendo variáveis numéricas:

Exemplo 1: a é menor que b?

a < b

## [1] TRUE

Exemplo 2: a é menor ou igual a b?

a <= b

## [1] TRUE

Exemplo 3: a é maior que b?

a > b

## [1] FALSE

Exemplo 4: a é maior ou igual a b?

a >= b

## [1] FALSE

Exemplo 5: a é exatamente igual a b?

a == b # exatamente igual

## [1] FALSE

Exemplo 6: a é diferente de b?

a != b # não igual a

## [1] TRUE

Agora, vamos ver alguns exemplos de testes lógicos utilizando variáveis booleanas, isto é, variáveis que assume apenas dois valores TRUE ou FALSE

Inicialmente, vamos definir duas variáveis lógicas x e y:

x <- TRUE  
x

## [1] TRUE

y <- FALSE  
y

## [1] FALSE

Qual o complemento lógico ou negação de x?

!x

## [1] FALSE

O operador & significa a conjunção “e”:

x & y # x e y

## [1] FALSE

O operador | significa a conjunção “ou”:

x | y # x ou y

## [1] TRUE

A seguir um exemplo de uso do operador & com uma variável numérica:

z <- 12  
z > 5 & z < 15

## [1] TRUE

## Tipos Atômicos de Dados

### double

a <- 1.23

typeof(a) # fornece o tipo atomico do objeto

## [1] "double"

### integer

b <- 2L  
b

## [1] 2

typeof(b)

## [1] "integer"

### character

d <- "João"   
d

## [1] "João"

typeof(d)

## [1] "character"

### logical

e <- TRUE  
e

## [1] TRUE

typeof(e)

## [1] "logical"

Fazendo a coerção de logical para numeric

v <- as.numeric(e)  
v

## [1] 1

### complex

c <- 1 + 3i  
c

## [1] 1+3i

typeof(c)

## [1] "complex"

## Tipos Especiais

### NA = Not Available

Em geral, o símbolo NA é reservado para representar dados faltantes:

g <- c(0, NA, 4, 7, NA)  
g

## [1] 0 NA 4 7 NA

### NaN = Not a Number

h <- c(0/0, 2, 100)  
h

## [1] NaN 2 100

### Inf = infinity

i <- c(1, 100/0, -10/0)  
i

## [1] 1 Inf -Inf

## Estruturas de Dados

### Vector

Vetores são estruturas de dados que podem armazenar somente elementos do mesmo tipo atômico.

Podemos criar vetores usando a função c(), sendo c uma abreviação de *concatenate*.

Vamos criar um vetor numérico:

vec1 <- c(0.5, 0.6, 0.1, 0.8, 2, 1.5) # numeric  
print(vec1)

## [1] 0.5 0.6 0.1 0.8 2.0 1.5

um lógico:

vec2 <- c(TRUE, FALSE) # logical  
print(vec2)

## [1] TRUE FALSE

um contendo caracteres:

vec3 <- c("a", "b", "c") # character  
print(vec3)

## [1] "a" "b" "c"

contendo números inteiros:

vec4 <- 9:29 # integer  
vec4

## [1] 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29

contendo números complexos:

vec5 <- c(1+0i, 2+4i) # complex  
vec5

## [1] 1+0i 2+4i

Podemos verificar a classe de cada um dos vetores criados usando a função class():

class(vec1)

## [1] "numeric"

class(vec2)

## [1] "logical"

class(vec3)

## [1] "character"

class(vec4)

## [1] "integer"

class(vec5)

## [1] "complex"

## Vetorização

Na linguagem R, um conceito fundamental é o de “vetorização”.

A vetorização refere-se à capacidade de realizar operações em vetores de dados de forma conveniente, sem a necessidade de loops explícitos.

Isso é essencial para a eficiência e a simplicidade do código em R.

A vetorização signfica que podemos realizar operações diretamente em vetores (adição, subtração, multiplicação e divisão).

Ou seja, temos dois vetores, podemos simplesmente efetuar operações aritméticas sem a necessidade de loops ou construções condicionais:

x <- c(1, 2, 3, 4, 5)  
y <- c(2, 3, 4, 5, 6)  
resultado <- mean((x - y)^2)  
resultado

## [1] 1

## Data Frame

### Criando uma data frame I

df1 <- data.frame(x1 = c(7.1, 2.5, 8.4, 3.2, 3.8, 7.3),   
 x2 = c("H", "M", "M", "H", "M", "H"),  
 x3 = c(0, 0, 1, 1, 0, 1)  
 )  
df1

## x1 x2 x3  
## 1 7.1 H 0  
## 2 2.5 M 0  
## 3 8.4 M 1  
## 4 3.2 H 1  
## 5 3.8 M 0  
## 6 7.3 H 1

### criando uma data frame II

custo = c(120, 180, 348, 125, 290)  
preco = c(152, 239, 487, 145, 345)  
produto = c("A","B","C","D","E")  
dataf = data.frame(produto, custo, preco)  
class(dataf)

## [1] "data.frame"

glimpse(dataf)

## Rows: 5  
## Columns: 3  
## $ produto <chr> "A", "B", "C", "D", "E"  
## $ custo <dbl> 120, 180, 348, 125, 290  
## $ preco <dbl> 152, 239, 487, 145, 345

## Factor

Considere uma variável que registra meses:

x1 <- c("Dez", "Abr", "Jan", "Mar")

Usar um vetor de caractres para registrar essa variável tem dois problemas:

1. Existem apenas doze meses possíveis e possibilidade de **typos** (erros de digitação):

x2 <- c("Dez", "Abr", "Jam", "Mar")  
x2

## [1] "Dez" "Abr" "Jam" "Mar"

1. O vetor não é ordenado de forma útil:

sort(x1)

## [1] "Abr" "Dez" "Jan" "Mar"

Podemos fixar ambos os problemas usando a estrutura de dados factor:

1. definindo os niveis do fator

month\_levels <- c(  
 "Jan", "Fev", "Mar", "Abr", "Mai", "Jun",   
 "Jul", "Ago", "Set", "Out", "Nov", "Dez"  
)

1. criando o fator

y1 <- factor(x1, levels = month\_levels)  
y1

## [1] Dez Abr Jan Mar  
## Levels: Jan Fev Mar Abr Mai Jun Jul Ago Set Out Nov Dez

1. Ordenação:

sort(y1)

## [1] Jan Mar Abr Dez  
## Levels: Jan Fev Mar Abr Mai Jun Jul Ago Set Out Nov Dez

y2 <- factor(x2, levels = month\_levels)  
y2

## [1] Dez Abr <NA> Mar   
## Levels: Jan Fev Mar Abr Mai Jun Jul Ago Set Out Nov Dez

sort(y2)

## [1] Mar Abr Dez  
## Levels: Jan Fev Mar Abr Mai Jun Jul Ago Set Out Nov Dez

acessando os níveis:

levels(y1)

## [1] "Jan" "Fev" "Mar" "Abr" "Mai" "Jun" "Jul" "Ago" "Set" "Out" "Nov" "Dez"

Quaisquer valores que não estejam no vetor de dados serão silenciosamente convertidos para NA:

x2 <- c("Dez", "Abr", "Jam", "Mar")  
y2 <- factor(x2, levels = month\_levels)  
y2

## [1] Dez Abr <NA> Mar   
## Levels: Jan Fev Mar Abr Mai Jun Jul Ago Set Out Nov Dez

Se omitirmos os níveis, eles serão retirados dos dados em ordem alfabética:

x1 <- c("Dez", "Abr", "Jan", "Mar")  
factor(x1)

## [1] Dez Abr Jan Mar  
## Levels: Abr Dez Jan Mar

## Matrix

### Criando uma Matriz

m <- matrix(c(0, 2, 1, 0), nrow = 2, ncol = 2, byrow = TRUE)  
m

## [,1] [,2]  
## [1,] 0 2  
## [2,] 1 0

dim(m) # dimensoes da matriz

## [1] 2 2

### Algebra Matricial

Adição:

m + m

## [,1] [,2]  
## [1,] 0 4  
## [2,] 2 0

Subtração:

m - m

## [,1] [,2]  
## [1,] 0 0  
## [2,] 0 0

Multiplicação por escalar:

2\*m

## [,1] [,2]  
## [1,] 0 4  
## [2,] 2 0

Multiplicação:

m %\*% m # multiplicacao de matrizes

## [,1] [,2]  
## [1,] 2 0  
## [2,] 0 2

Multiplicação elemento por elemento:

m \* m # multiplicacao elemento x elemento

## [,1] [,2]  
## [1,] 0 4  
## [2,] 1 0

Matriz transposta:

t(m) # transposta

## [,1] [,2]  
## [1,] 0 1  
## [2,] 2 0

Matriz inversa:

solve(m) # inversa de m1 (se existir)

## [,1] [,2]  
## [1,] 0.0 1  
## [2,] 0.5 0

verificando a validade da matriz inversa:

solve(m) %\*% m == diag(nrow = nrow(m), ncol = ncol(m))

## [,1] [,2]  
## [1,] TRUE TRUE  
## [2,] TRUE TRUE

### Autovalores e Autovetores

eigen(m)

## eigen() decomposition  
## $values  
## [1] 1.414214 -1.414214  
##   
## $vectors  
## [,1] [,2]  
## [1,] 0.8164966 -0.8164966  
## [2,] 0.5773503 0.5773503

## List

### criando uma lista

lista\_1 <- list(vec1, df1, m)  
lista\_1

## [[1]]  
## [1] 0.5 0.6 0.1 0.8 2.0 1.5  
##   
## [[2]]  
## x1 x2 x3  
## 1 7.1 H 0  
## 2 2.5 M 0  
## 3 8.4 M 1  
## 4 3.2 H 1  
## 5 3.8 M 0  
## 6 7.3 H 1  
##   
## [[3]]  
## [,1] [,2]  
## [1,] 0 2  
## [2,] 1 0

# Manipulacao de Dados

## Vetores

ls() # lista os objetos ativos na secao

## [1] "a" "b" "c" "custo" "d"   
## [6] "dataf" "df1" "e" "g" "h"   
## [11] "i" "lista\_1" "m" "month\_levels" "preco"   
## [16] "produto" "r\_rocks" "resultado" "v" "vec1"   
## [21] "vec2" "vec3" "vec4" "vec5" "x"   
## [26] "x1" "x2" "y" "y1" "y2"   
## [31] "z"

### Extracao de elementos de Vetores

Exibindo o vetor vec1

print(vec1)

## [1] 0.5 0.6 0.1 0.8 2.0 1.5

eleciona o primeiro elemento:

vec1[1]

## [1] 0.5

seleciona o sexto elemento:

vec1[6]

## [1] 1.5

seleciona todos, exceto o primeiro elemento:

vec1[-1]

## [1] 0.6 0.1 0.8 2.0 1.5

seleciona todos menos o primeiro e o segundo elementos:

vec1[c(-1,-2)]

## [1] 0.1 0.8 2.0 1.5

seleciona o segundo e quarto elementos:

vec1[c(2,4)]

## [1] 0.6 0.8

seleciona o segundo até o quarto elementos:

vec1[c(2:4)]

## [1] 0.6 0.1 0.8

### Substituindo um elemento de um vetor

O terceiro elemento passa a ser 500:

vec1[3] <- 500   
vec1

## [1] 0.5 0.6 500.0 0.8 2.0 1.5

### Funções Matemáticas e Estatísticas para vetores

Calcula o tamanho/numero de elementos do vetor:

length(vec1)

## [1] 6

Há quantos elementos únicos no vetor:

unique(vec1) #

## [1] 0.5 0.6 500.0 0.8 2.0 1.5

Ordena os elementos em ordem ascendente:

sort(vec1)

## [1] 0.5 0.6 0.8 1.5 2.0 500.0

Ordena os elementos em ordem decrescente:

sort(vec1, decreasing = TRUE)

## [1] 500.0 2.0 1.5 0.8 0.6 0.5

Calcula a soma dos elementos:

sum(vec1)

## [1] 505.4

Calcula o produto dos elementos do vetor:

prod(vec1)

## [1] 360

Fornece o mínimo dos elementos do vetor:

min(vec1)

## [1] 0.5

Fornece o máximo dos elementos do vetor

max(vec1) # máximo dos elementos do vetor

## [1] 500

Calcula a média dos elementos:

sum(vec1)/length(vec1)

## [1] 84.23333

mean(vec1) # média dos elementos

## [1] 84.23333

Calcula a mediana dos elementos:

median(vec1) # mediana dos elementos

## [1] 1.15

Fornece os valores mínimo e máximo:

range(vec1)

## [1] 0.5 500.0

Calcula a variância dos dados:

sum((vec1 - mean(vec1))^2)/(length(vec1) - 1)

## [1] 41487.19

var(vec1)

## [1] 41487.19

Calcula o desvio-padrão:

sqrt(var(vec1))

## [1] 203.6841

sd(vec1)

## [1] 203.6841

Calcula a covariância entre as duas variáveis:

cov(vec1,vec1) # covariância (cov(x,y))

## [1] 41487.19

Calcula a correlação entre as duas variáveis:

cor(vec1,vec1) # correlação (cor(x,y))

## [1] 1

Fornece um resumo de estatísticas descritvas:

summary(vec1) # estatíticas descritivas

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 0.500 0.650 1.150 84.233 1.875 500.000

### Lidando com valores faltantes em estatísticas descritivas

Vamos criar um vetor contendo valores faltantes (missing data):

vetor\_na <- c(1, 2, 0, 2, NA, 5, 10, NA)

Se calcularmos a média de vetor\_na com a função mean() sem remover os valores ausentes, obtemos a seguinte resultado:

mean(vetor\_na)

## [1] NA

se houver valores faltantes em uma variável e R não for instruída a considerar sua presença ao executar uma função, então a saída dessa função será NA.

Portanto, precisamos informar a linguagem para ignorar as observações que são NA.

Fazemos isso inserindo a opção na.rm=TRUE dentro da função, o que significa que é verdadeiro remover os dados faltantes:

mean(vetor\_na, na.rm = TRUE)

## [1] 3.333333

Há dados faltantes há no vetor vetor\_na?

is.na(g)

## [1] FALSE TRUE FALSE FALSE TRUE

Quantos dados faltantes há no vetor vetor\_na?:

sum(is.na(g))

## [1] 2

Quantos dados completos há no vetor vetor\_na?

## Análise Exploratória de Dados

Dados utilizados

data("gapminder")  
head(gapminder) # exibe as primeiras 6 linhas da data frame

## # A tibble: 6 × 6  
## country continent year lifeExp pop gdpPercap  
## <fct> <fct> <int> <dbl> <int> <dbl>  
## 1 Afghanistan Asia 1952 28.8 8425333 779.  
## 2 Afghanistan Asia 1957 30.3 9240934 821.  
## 3 Afghanistan Asia 1962 32.0 10267083 853.  
## 4 Afghanistan Asia 1967 34.0 11537966 836.  
## 5 Afghanistan Asia 1972 36.1 13079460 740.  
## 6 Afghanistan Asia 1977 38.4 14880372 786.

Podemos ter uma visão rápida e geral da estrutura de dados de gapminder com a função glimpse() do dplyr:

dplyr::glimpse(gapminder)

## Rows: 1,704  
## Columns: 6  
## $ country <fct> "Afghanistan", "Afghanistan", "Afghanistan", "Afghanistan", …  
## $ continent <fct> Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, …  
## $ year <int> 1952, 1957, 1962, 1967, 1972, 1977, 1982, 1987, 1992, 1997, …  
## $ lifeExp <dbl> 28.801, 30.332, 31.997, 34.020, 36.088, 38.438, 39.854, 40.8…  
## $ pop <int> 8425333, 9240934, 10267083, 11537966, 13079460, 14880372, 12…  
## $ gdpPercap <dbl> 779.4453, 820.8530, 853.1007, 836.1971, 739.9811, 786.1134, …

A função skim do pacote skimr fornece diversas estatísticas descritivas de uma data.frame ou tibble (versão modernizada de uma data.frame):

skimr::skim(gapminder)

Data summary

|  |  |
| --- | --- |
| Name | gapminder |
| Number of rows | 1704 |
| Number of columns | 6 |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  |
| Column type frequency: |  |
| factor | 2 |
| numeric | 4 |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  |
| Group variables | None |

**Variable type: factor**

| skim\_variable | n\_missing | complete\_rate | ordered | n\_unique | top\_counts |
| --- | --- | --- | --- | --- | --- |
| country | 0 | 1 | FALSE | 142 | Afg: 12, Alb: 12, Alg: 12, Ang: 12 |
| continent | 0 | 1 | FALSE | 5 | Afr: 624, Asi: 396, Eur: 360, Ame: 300 |

**Variable type: numeric**

| skim\_variable | n\_missing | complete\_rate | mean | sd | p0 | p25 | p50 | p75 | p100 | hist |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| year | 0 | 1 | 1979.50 | 17.27 | 1952.00 | 1965.75 | 1979.50 | 1993.25 | 2007.0 | ▇▅▅▅▇ |
| lifeExp | 0 | 1 | 59.47 | 12.92 | 23.60 | 48.20 | 60.71 | 70.85 | 82.6 | ▁▆▇▇▇ |
| pop | 0 | 1 | 29601212.32 | 106157896.74 | 60011.00 | 2793664.00 | 7023595.50 | 19585221.75 | 1318683096.0 | ▇▁▁▁▁ |
| gdpPercap | 0 | 1 | 7215.33 | 9857.45 | 241.17 | 1202.06 | 3531.85 | 9325.46 | 113523.1 | ▇▁▁▁▁ |

### Pacote dplyr: select()

select(df, a, b,...): seleciona apenas as colunas/variáveis que desejamos.

seleção por inclusão

dados <- select(gapminder, year, country, gdpPercap)

seleção por exclusão

smaller\_gapminder\_data <- select(gapminder, -continent)

Boa pratica com dplyr

dados <- gapminder %>% select(year, country, gdpPercap)

**Dica:** atalho do RStudio: Ctrl + Alt + I insere o operador pipe %>%

### Pacote dplyr: filter()

Se agora quisermos avançar analisando apenas países europeus, podemos combinar select(), que seleciona colunas/variáveis, e filter() que seleciona linhas.

dados <- gapminder %>%  
 filter(continent == "Europe") %>%  
 select(year, country, gdpPercap)

Se quisermos analisar a expectativa de vida dos países europeus, mas apenas para um ano específico (por exemplo, 2007), podemos fazer:

dados\_lifexp2007 <- gapminder %>%  
 filter(continent == "Europe", year == 2007) %>%  
 select(country, lifeExp)

### Pacote dplyr: group\_by() e summarise()

Se quisermos estimar a renda média per capita por continente em todos os anos?

Usando a função group\_by(), dividimos a data frame original em várias partes, então podemos executar funções como mean() dentro de summarise()

rmp\_continente <- gapminder %>%  
 group\_by(continent) %>%  
 summarize(gdpPercap\_media = mean(gdpPercap))  
rmp\_continente

## # A tibble: 5 × 2  
## continent gdpPercap\_media  
## <fct> <dbl>  
## 1 Africa 2194.  
## 2 Americas 7136.  
## 3 Asia 7902.  
## 4 Europe 14469.  
## 5 Oceania 18622.

### agrupando por mais de uma coluna/variavel

A função group\_by() nos permite agrupar os dados por múltiplas variáveis.

rmp\_continente\_ano <- gapminder %>%  
 group\_by(continent, year) %>%  
 summarize(gdpPercap\_media = mean(gdpPercap))  
rmp\_continente\_ano

## # A tibble: 60 × 3  
## # Groups: continent [5]  
## continent year gdpPercap\_media  
## <fct> <int> <dbl>  
## 1 Africa 1952 1253.  
## 2 Africa 1957 1385.  
## 3 Africa 1962 1598.  
## 4 Africa 1967 2050.  
## 5 Africa 1972 2340.  
## 6 Africa 1977 2586.  
## 7 Africa 1982 2482.  
## 8 Africa 1987 2283.  
## 9 Africa 1992 2282.  
## 10 Africa 1997 2379.  
## # ℹ 50 more rows

### Sumarizando dados por mais de uma estatistica

Isso já é bastante poderoso, mas fica ainda melhor! Você não está limitado a definir apenas uma estatística em summarise()

rmp\_continente\_pop\_ano <- gapminder %>%  
 group\_by(continent, year) %>%  
 summarize(gdpPercap\_media = mean(gdpPercap),  
 gdpPercap\_dp = sd(gdpPercap),  
 pop\_media = mean(pop),  
 pop\_dp = sd(pop))  
rmp\_continente\_pop\_ano

## # A tibble: 60 × 6  
## # Groups: continent [5]  
## continent year gdpPercap\_media gdpPercap\_dp pop\_media pop\_dp  
## <fct> <int> <dbl> <dbl> <dbl> <dbl>  
## 1 Africa 1952 1253. 983. 4570010. 6317450.  
## 2 Africa 1957 1385. 1135. 5093033. 7076042.  
## 3 Africa 1962 1598. 1462. 5702247. 7957545.  
## 4 Africa 1967 2050. 2848. 6447875. 8985505.  
## 5 Africa 1972 2340. 3287. 7305376. 10130833.  
## 6 Africa 1977 2586. 4142. 8328097. 11585184.  
## 7 Africa 1982 2482. 3243. 9602857. 13456243.  
## 8 Africa 1987 2283. 2567. 11054502. 15277484.  
## 9 Africa 1992 2282. 2644. 12674645. 17562719.  
## 10 Africa 1997 2379. 2821. 14304480. 19873013.  
## # ℹ 50 more rows

### Pacote dplyr: count() e n()

* Uma operação muito comum é contar o número de observações para cada grupo.
* O pacote dplyr possui duas funções relacionadas que ajudam nisso.
* count(): permite contar os valores únicos de uma ou mais variáveis
* n(): fornece o tamanho do grupo

Por exemplo, se quisermos verificar o número de países, por continente, incluídos no conjunto de dados para o ano de 2002, podemos fazer:

gapminder %>%  
 filter(year == 2002) %>%  
 count(continent, sort = TRUE)

## # A tibble: 5 × 2  
## continent n  
## <fct> <int>  
## 1 Africa 52  
## 2 Asia 33  
## 3 Europe 30  
## 4 Americas 25  
## 5 Oceania 2

### Pacote dplyr: mutate()

Também podemos criar novas variáveis antes (ou mesmo depois) de resumir  
informações usando mutate():

pib\_pop\_continente\_ano <- gapminder %>%  
 mutate(pib\_bilhoes = gdpPercap\*pop/10^9) %>%  
 group\_by(continent,year) %>%  
 summarize(gdpPercap\_media = mean(gdpPercap),  
 gdpPercap\_dp = sd(gdpPercap),  
 pop\_media = mean(pop),  
 pop\_dp = sd(pop),  
 pib\_bilhoes\_media = mean(pib\_bilhoes),  
 pib\_bilhoes\_dp = sd(pib\_bilhoes))  
  
glimpse(pib\_pop\_continente\_ano)

## Rows: 60  
## Columns: 8  
## Groups: continent [5]  
## $ continent <fct> Africa, Africa, Africa, Africa, Africa, Africa, Afri…  
## $ year <int> 1952, 1957, 1962, 1967, 1972, 1977, 1982, 1987, 1992…  
## $ gdpPercap\_media <dbl> 1252.572, 1385.236, 1598.079, 2050.364, 2339.616, 25…  
## $ gdpPercap\_dp <dbl> 982.9521, 1134.5089, 1461.8392, 2847.7176, 3286.8539…  
## $ pop\_media <dbl> 4570010, 5093033, 5702247, 6447875, 7305376, 8328097…  
## $ pop\_dp <dbl> 6317450, 7076042, 7957545, 8985505, 10130833, 115851…  
## $ pib\_bilhoes\_media <dbl> 5.992295, 7.359189, 8.784877, 11.443994, 15.072242, …  
## $ pib\_bilhoes\_dp <dbl> 11.43635, 14.50029, 17.17966, 23.18867, 30.39608, 38…

### Pacote dplyr: arrange()

* arrange() ordena as linhas/observações/casos de uma data.frame (ou tibble) pelos valores das colunas/variáveis selecionadas.

rmp\_continente <- gapminder %>%  
 group\_by(continent) %>%  
 summarize(gdpPercap\_media = mean(gdpPercap)) %>%  
 arrange(gdpPercap\_media)   
  
rmp\_continente

## # A tibble: 5 × 2  
## continent gdpPercap\_media  
## <fct> <dbl>  
## 1 Africa 2194.  
## 2 Americas 7136.  
## 3 Asia 7902.  
## 4 Europe 14469.  
## 5 Oceania 18622.

* Ordenando em ordem decrescente:

rmp\_continente <- gapminder %>%  
 group\_by(continent) %>%  
 summarize(gdpPercap\_media = mean(gdpPercap)) %>%  
 arrange(desc(gdpPercap\_media))  
  
rmp\_continente

## # A tibble: 5 × 2  
## continent gdpPercap\_media  
## <fct> <dbl>  
## 1 Oceania 18622.  
## 2 Europe 14469.  
## 3 Asia 7902.  
## 4 Americas 7136.  
## 5 Africa 2194.

# INCLUIR GRÁFICOS