Introdução ao R para Análise de Dados

Minicurso

Paulo Alexandrino

Março 2023

Sumário

Capítulo 1

Conceitos Fundamentais

1.1 Operadores

1.1.1 Operadores matemáticos

A forma mais elementar de utilizar o R é como uma calculadora que nos permite realizar todo tipo de operação matemática desejada.

```
# Soma (+)
5 + 7
```

```
## [1] 12
```

```
# Subtração (-)
10 - 16
```

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

```
# Multiplicação (*)
2 * 7
```

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

```
# Exponenciação (^)
2^4
```

```
## [1] 16
```

```r
# Divisão (/)
5 / 2
```

```
## [1] 2.5
```

```r
# Divisão Inteira (%/%)
5 %/% 2
```

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

```r
# Módulo - Resto da divisão inteira (%%)
5 %% 2
```

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

1.1.2 Operadores lógicos

Operadores lógicos nos permitem estabelecer relações entre objetos e julgar proposições (orações declarativas) como verdadeiras (`TRUE`) ou falsas (`FALSE`). Serão muito úteis a construção de filtros.

```r
# Menor que (<)
5 < 3
```

```
## [1] FALSE
```

```r
# Menor ou igual que (<=)
2.5 <= 7
```

```
## [1] TRUE
```

```r
# Maior que (>)
10 > 15
```

```
## [1] FALSE
```

```r
# Maior ou igual que (>=)
11 >= 11
```

```
## [1] TRUE
```

```r
# Igual a (==)
2 == 3
```

```
## [1] FALSE
```

```r
# Diferente de (!=)
5 != 4
```

```
## [1] TRUE
```

A negação de uma proposição é seu inverso lógico.

```r
# Negação (!)
2 > 5
```

```
## [1] FALSE
```

```r
!(2 > 5)
```

```
## [1] TRUE
```

Os conectivos de conjunção ($\wedge$, lê-se `E`) e disjunção ($\vee$, lê-se `OU`) são operadores lógicos que permitem criar proposições compostas. Sejam p e q proposições quaisquer:

- A conjunção $p \wedge q$ é verdadeira se p e q são ambas verdadeiras; se ao menos uma delas for falsa, $p \wedge q$ é falsa.

p	q	p $\wedge$ q
V	V	V
V	F	F
F	V	F
F	F	F

```r
# E (&)
(2 > 1) & (3 != 0)
```

```
## [1] TRUE
```

```r
(3 < 2) & (5 < 11)
```

```
## [1] FALSE
```

- A disjunção $p \vee q$ é verdadeira se p e q se ao menos uma das proposições p ou q é verdadeira; se p e q são ambas falsas, então $p \vee q$ é falsa.

p	q	p V q
V	V	V
V	F	V
F	V	V
F	F	F

```r
# OU (|)
(3 < 2) | (5 < 11)
```

```
## [1] TRUE
```

```r
(5 == 6) | (12 < 8)
```

```
## [1] FALSE
```

1.2 Objetos, Tipos de Dados e Variáveis

Em R praticamente tudo é um objeto: um número, uma palavra, um vetor contendo diversos números, uma tabela contendo diferentes vetores, uma lista contendo várias tabelas...as coisas podem ficar tão complicadas quanto se queira.

1.2.1 Vetores

Vetores são objetos formados por dados de um mesmo tipo. Um vetor pode conter nenhum, um ou mais valores. Alguns dos principais tipos de dados são:

- `logical`: `TRUE` ou `FALSE`
- `integer`: números inteiros
- `numeric` ou `double`: números reais
- `character`: texto

```r
# Para atribuir um valor a uma variável utilizados "<-" ou então "=".
# Não confundiro o = de atribuição com o == operador lógico.
# Vetores podem ser criados, ou concatenados como veremos mais adiante,
# utilizando a função c().

# Criando a variável vetor_vazio e deixando ela vazia
vetor_vazio <- c()
# Imprimindo a variável vetor_vazio
vetor_vazio
```

```
## NULL
```

```r
# Podemos atribuir apenas um valor a uma variável
var_texto <- "Palavra"
var_texto
```

```
## [1] "Palavra"
```

```r
# Podemos atribuir qualquer valor a uma variável
var_numerica <- 3*(5 + 2)
var_numerica
```

```
## [1] 21
```

```r
# Criando um vetores com três valores
var_texto <- c("Palavra","Word","Mot")
var_texto
```

```
## [1] "Palavra" "Word"    "Mot"
```

```r
# Criando um vetor lógico
var_logico = c(3 + 4 < 5, 2 == 1 + 1)
var_logico
```

```
## [1] FALSE  TRUE
```

1.2.2 Data frames

Podemos agrupar vetores em um data frame. Em R, data frames são tabelas bidimensionais, semelhantes às planilhas do Excel. Este tipo de objeto será o mais utilizado em nosso dia-a-dia.

Vamos construir um data frame dos estados da região sudeste do Brasil, com as informações de nome, sigla, nome da capital, código DDD da capital e se a capital possui acesso para o mar.

```
# Criando um data frame:
estados_sudeste <- data.frame(
  estado = c("Rio de Janeiro", "Sao Paulo", "Minas Gerais","Espirito Santo"),
  sigla = c("RJ", "SP", "MG", "ES"),
  capital = c("Rio de Janeiro", "Sao Paulo", "Belo Horizonte", "Vitoria"),
  DDD_capital = c(21, 11, 31, 27),
  mar_na_capital = c(TRUE, FALSE, FALSE, TRUE)
)

estados_sudeste
```

```
##           estado sigla        capital DDD_capital mar_na_capital
## 1 Rio de Janeiro    RJ Rio de Janeiro          21           TRUE
## 2      Sao Paulo    SP      Sao Paulo          11          FALSE
## 3   Minas Gerais    MG Belo Horizonte          31          FALSE
## 4 Espirito Santo    ES        Vitoria          27           TRUE
```

Esse procedimento funciona para data frames pequenos, mas não é difícil notar que se torna uma tarefa trabalhosa conforme aumenta-se o número de observações e variáveis. Felizmente, o R permite importar dados de outras fontes, o que aprenderemos a fazer em breve.

1.3 Funções

Funções são ferramentas, porções de código reutilizáveis. Elas nos permitem realizar os mais variados tipos de operações, seja criar ou modificar objetos, realizar comandos no computador, e muito mais.

Funções quase sempre recebem argumentos, de forma a ajustar seu comportamento de acordo com algum objetivo específico. Alguns argumentos são obrigatórios, outros opicionais, ou seja, possuem valores pré-definidos.

Caso queira obter mais informações sobre uma função, você pode acessar sua documentação. A documentação contém tudo, ou a maior parte, do que se precisa saber sore uma função: quais seus argumentos, como eles se comportam, quais os valores pré-definidos. Para acessar a documentação basta digitar um ponto de interrogação seguido do nome da função.

Para aprendermos a usar algumas funções essenciais, usaremos o dataset `USArrests`, que contém dados sobre criminalidade nos 50 estados dos EUA no ano de 1973, e já vem pré-carregado no R. Assim, não precisaremos importar nada.

1.3.1 Explorando os dados

```
# Vendo a estrutura dos dados:
str(USArrests)
```

```
## 'data.frame':    50 obs. of  4 variables:
##  $ Murder  : num  13.2 10 8.1 8.8 9 7.9 3.3 5.9 15.4 17.4 ...
##  $ Assault : int  236 263 294 190 276 204 110 238 335 211 ...
##  $ UrbanPop: int  58 48 80 50 91 78 77 72 80 60 ...
##  $ Rape    : num  21.2 44.5 31 19.5 40.6 38.7 11.1 15.8 31.9 25.8 ...
```

```
# Vendo um resumo dos dados:
summary(USArrests)
```

```
##      Murder          Assault         UrbanPop          Rape
##  Min.   : 0.800   Min.   : 45.0   Min.   :32.00   Min.   : 7.30
##  1st Qu.: 4.075   1st Qu.:109.0   1st Qu.:54.50   1st Qu.:15.07
##  Median : 7.250   Median :159.0   Median :66.00   Median :20.10
##  Mean   : 7.788   Mean   :170.8   Mean   :65.54   Mean   :21.23
##  3rd Qu.:11.250   3rd Qu.:249.0   3rd Qu.:77.75   3rd Qu.:26.18
##  Max.   :17.400   Max.   :337.0   Max.   :91.00   Max.   :46.00
```

```
# Vendo as primeiras observações:
head(USArrests)
```

```
##            Murder Assault UrbanPop Rape
## Alabama      13.2     236       58 21.2
## Alaska       10.0     263       48 44.5
## Arizona       8.1     294       80 31.0
## Arkansas      8.8     190       50 19.5
## California    9.0     276       91 40.6
## Colorado      7.9     204       78 38.7
```

```
# Vendo as últimas observações:
tail(USArrests)
```

```
##               Murder Assault UrbanPop Rape
## Vermont          2.2      48       32 11.2
## Virginia         8.5     156       63 20.7
## Washington       4.0     145       73 26.2
## West Virginia    5.7      81       39  9.3
## Wisconsin        2.6      53       66 10.8
## Wyoming          6.8     161       60 15.6
```

```r
# Número de linhas:
nrow(USArrests)
```

```
## [1] 50
```

```r
# Número de colunas:
ncol(USArrests)
```

```
## [1] 4
```

Como vimos anteriormente, um `data frame` é uma representação tabular, onde cada linha corresponde a uma observação e cada coluna uma variável, ou atributo. Para obtermos uma coluna específica de um `data frame`, podemos utilizar o símbolo `$`.

```r
# Selecionando apenas a coluna Murder:
USArrests$Murder
```

```
##  [1] 13.2 10.0  8.1  8.8  9.0  7.9  3.3  5.9 15.4 17.4  5.3  2.6 10.4  7.2  2.2
## [16]  6.0  9.7 15.4  2.1 11.3  4.4 12.1  2.7 16.1  9.0  6.0  4.3 12.2  2.1  7.4
## [31] 11.4 11.1 13.0  0.8  7.3  6.6  4.9  6.3  3.4 14.4  3.8 13.2 12.7  3.2  2.2
## [46]  8.5  4.0  5.7  2.6  6.8
```

```r
# Classe de um objeto:
class(USArrests$Murder)
```

```
## [1] "numeric"
```

```r
# Mínimo:
min(USArrests$Murder)
```

```
## [1] 0.8
```

```r
# Máximo:
max(USArrests$Murder)
```

```
## [1] 17.4
```

```r
# Somatório:
sum(USArrests$Murder)
```

```
## [1] 389.4
```

```r
# Comprimento de um vetor:
length(USArrests$Murder)
```

```
## [1] 50
```

```r
# Ordenar um vetor em ordem crescente:
sort(USArrests$Murder)
```

```
##  [1]  0.8  2.1  2.1  2.2  2.2  2.6  2.6  2.7  3.2  3.3  3.4  3.8  4.0  4.3  4.4
## [16]  4.9  5.3  5.7  5.9  6.0  6.0  6.3  6.6  6.8  7.2  7.3  7.4  7.9  8.1  8.5
## [31]  8.8  9.0  9.0  9.7 10.0 10.4 11.1 11.3 11.4 12.1 12.2 12.7 13.0 13.2 13.2
## [46] 14.4 15.4 15.4 16.1 17.4
```

```r
# A função sort possui o argumento opcional decreasing = FALSE.
# Para ordenar a lista de forma decrescente, podemos alterá-lo.
sort(USArrests$Murder, decreasing = TRUE)
```

```
##  [1] 17.4 16.1 15.4 15.4 14.4 13.2 13.2 13.0 12.7 12.2 12.1 11.4 11.3 11.1 10.4
## [16] 10.0  9.7  9.0  9.0  8.8  8.5  8.1  7.9  7.4  7.3  7.2  6.8  6.6  6.3  6.0
## [31]  6.0  5.9  5.7  5.3  4.9  4.4  4.3  4.0  3.8  3.4  3.3  3.2  2.7  2.6  2.6
## [46]  2.2  2.2  2.1  2.1  0.8
```

1.3.2 Estatísticas Descritivas

Vamos usar agora outras colunas para calcularmos estatísticas descritivas utilizando funções do R.

```r
# Média:
mean(USArrests$Assault)
```

```
## [1] 170.76
```

```r
# Mediana:
median(USArrests$Assault)
```

```
## [1] 159
```

```r
# Quantis:
quantile(USArrests$Assault)
```

```
##   0%  25%  50%  75% 100%
##   45  109  159  249  337
```

```r
# Variância:
var(USArrests$Assault)
```

```
## [1] 6945.166
```

```r
# Desvio padrão:
sd(USArrests$Assault)
```

```
## [1] 83.33766
```

```r
# Covariância:
cov(USArrests$Assault, USArrests$Murder)
```

```
## [1] 291.0624
```

```r
# Correlação:
cor(USArrests$Assault, USArrests$Murder)
```

```
## [1] 0.8018733
```

Ao resultado de uma função pode ser aplicada outra função, o que é chamado de aninhamento. Vamos supor, por exemplo que queiremos arredondar, com três casas decimais, o valor calculado para a correlação entre as variáveis `Assault` e `Murder`.

```r
# A função utilizada para arredondar um número é round().
# Para que tenha 2 casa decimais, precisaremos ajustar o parâmetro digits.
round(cor(USArrests$Assault, USArrests$Murder), digits = 3)
```

```
## [1] 0.802
```

1.3.3 Criação de vetores

Vamos conhecer algumas funções que podem ser usadas para criar vetores numéricos de forma fácil.

```r
# Criar um vetor de com o valor x repetido n vezes:
rep(2, 3)
```

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

```r
# Criar uma sequencia de números: seq()
seq(from = 1, to = 50, by = 3)
```

```
##  [1]  1  4  7 10 13 16 19 22 25 28 31 34 37 40 43 46 49
```

```r
# Podemos substituir, por exemplo seq(from = x, to = y, by = 1)
3:15
```

```
##  [1]  3  4  5  6  7  8  9 10 11 12 13 14 15
```

```r
# Vamos atribuir os valores anteriormente encontrados a variáveis:
var1 <- rep(2, 3)
var2 <- seq(from = 1, to = 50, by = 3)

# Para concatenar vetores, utilizamos a função c()
meu_vetor <- c(var1, var2)
meu_vetor
```

```
##  [1]  2  2  2  1  4  7 10 13 16 19 22 25 28 31 34 37 40 43 46 49
```

1.3.4 Amostragem

O R foi uma linguagem criada por estatísticos e até hoje possui como um de seus pontos fortes sua capacidade de lidar com amostragem e simulações.

```r
# Permutação aleatórias dos números naturais menores que 7:
sample(7)
```

```
## [1] 1 5 7 6 2 3 4
```

```r
# Permutação aleatória dos números entre 2 e 5:
sample(2:5)
```

```
## [1] 3 2 5 4
```

```r
# Sorteio, sem reposição, de 5 números aleatórios
# da variável var2 que criamos anteriormente:
sample(var2, 5)
```

```
## [1] 19 10 25 13 46
```

```r
# Sorteio, com reposição, de 6 números aleatórios entre 1 e 4:
sample(1:4, size = 6, replace = TRUE)
```

```
## [1] 2 2 3 3 1 1
```

```r
# Sorteio, sem reposição, de uma palavra em uma lista:
sample(c("pera", "uva", "banana", "caqui"), 1)
```

```
## [1] "banana"
```

1.4 Baixando e excutando pacotes externos

Até agora utilizamos apenas funções pré-instaladas no R. Para ter acesso a novos pacotes, será necessário instalá-los primeiro. Esse procedimento só é necessário ser feito uma vez, a não ser que o R precise ser reinstalado. Sempre que iniciar uma sessão será necessário carregar os pacotes que serão utilizados.

```r
# Instalando o tidyverse
install.packages("tidyverse")

# Carregando o tidyverse
library("tidyverse")
```

O `tidyverse` é um conjunto de pacotes que juntos fornecem uma ampla gama de ferramentas para análise de dados. Algumas dessas ferramentas apresentam versões otimizadas de funções já existentes no R.

O operador pipe (`%>%`) permite substituir o aninhamento de funções por uma cadeia de operação, o que torna os códigos mais legíveis. Veremos aplicações do pipe e de algumas dos principais pacotes e funções do `tidyverse` nos capítulos posteriores.

Capítulo 2

Importação

O R nos permite importar e utilizar em nossas análises uma série de diferentes tipos de arquivos. Isso pode ser feito por meio de funções nativas ou através de ferramentas disponíveis em pacotes criados por outros usuários. Por exemplo, o `tidyverse` possui os pacotes `readr`, que permite a leitura de arquivos como .csv, .fwf e .txt, e o pacote `readxl`, para leitura de arquivos Excel.

2.1 Arquivos separados por vírgula (.csv)

Comma-separated values é um formato de arquivo de texto onde as observações de uma tabela são representadas por linhas e as colunas são separadas por vírgula (no padrão americano) ou ponto-e-vírgula (no padrão europeu e brasileiro). No padrão separado por vírgula, o indicador decimal é o ponto; no padrãos separado por ponto-e-vírgula, a vírcula. É importante estar atento ao tipo de arquivo que estamos lidando.

O dataset `gapminder` contém dados econômicos e demográficos para diversos países a cada 5 anos de 1952 a 2007. O arquivo que usaremos como exemplo apresenta uma amostra contendo as informações disponíveis para o Brasil, Botsuana e Coréia do Sul. Ele foi salvo em .csv com padrão separado por ponto-e-vírgula.

```
# Primeira forma: ajustando os parâmetros da função read.csv
gapminderCountries <- read.csv("Aula1/gapminderCountries.csv",
                               sep = ";",
                               dec = ",")

# Segunda forma: utilizando a função read.csv2
gapminderCountries <- read.csv2("Aula1/gapminderCountries.csv")
```

```r
# Mostrando um pedaço do dataset
head(gapminderCountries)
```

```
##    country continent year lifeExp    pop gdpPercap
## 1 Botswana    Africa 1952  47.622 442308  851.2411
## 2 Botswana    Africa 1957  49.618 474639  918.2325
## 3 Botswana    Africa 1962  51.520 512764  983.6540
## 4 Botswana    Africa 1967  53.298 553541 1214.7093
## 5 Botswana    Africa 1972  56.024 619351 2263.6111
## 6 Botswana    Africa 1977  59.319 781472 3214.8578
```

Para maiores informações, leia a documentação das funções `read.csv` e `read.csv2`.

2.2 Arquivos Excel (.xls e .xlsx)

O Microsoft Excel é um dos *softwares* de planilha mais populares do mundo. São muito utilizados não só no meio corporativo, mas também no meio acadêmico. Apesar de limitações principalmente relacionadas a lidar com datas e tabelas muito grandes, é uma ferramenta poderosa e torna-se ainda melhor se aliada ao R.

O `tidyverse` não carrega automaticamente o pacote necessário para ler arquivos Excel, portanto, precisaremos carregar o `readxl`. Utilizaremos como exemplo uma amostra do dataset `gapminder`, mas desta vez contendo informações para todos os países no ano de 2007.

```r
# Carregando o pacote necessário
library(readxl)

# Importando o arquivo de interesse
gapminder2007 <- read_excel("Aula1/gapminder2007.xlsx")

# Mostrando um pedaço do dataset
head(gapminder2007)
```

```
## # A tibble: 6 x 6
##   country     continent  year lifeExp      pop gdpPercap
##   <chr>       <chr>     <dbl>   <dbl>    <dbl>     <dbl>
## 1 Afghanistan Asia       2007    43.8 31889923      975.
## 2 Albania     Europe     2007    76.4  3600523     5937.
## 3 Algeria     Africa     2007    72.3 33333216     6223.
```

```
## 4 Angola      Africa     2007    42.7 12420476     4797.
## 5 Argentina   Americas   2007    75.3 40301927    12779.
## 6 Australia   Oceania    2007    81.2 20434176    34435.
```

Capítulo 3

Visualização

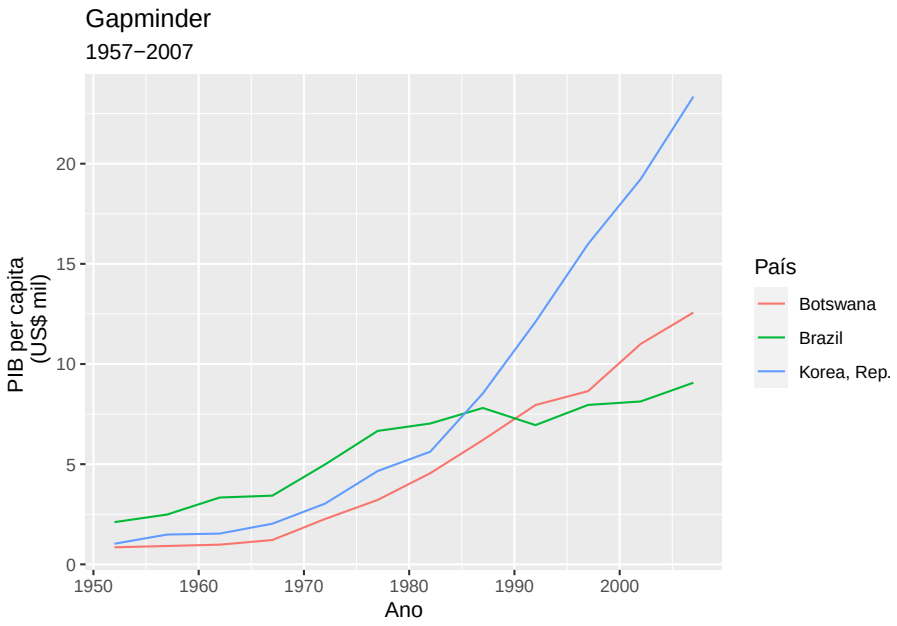
A visualização pode ser utilizada tanto para conhecer os dados com que iremos trabalhar, quanto para comunicar com ao público aquilo que descobrimos. Para tanto, o `tidyverse` fornece o pacote `ggplot2`, uma ferramenta poderossíma!

Ele permite construir desde gráficos simples até visualizações extremamente elaboradas, onde diversas variáveis são mapeadas em diferentes atributos de um gráfico. Somado a outras ferramentas, o `ggplot2` também pode ser utilizado para a construção de mapas.

3.1 Gráfico de linha

Gráficos de linhas são muito utilizados para mostrar tendências ao longo do tempo. Utilizando o dataset `gapminderCountries` que contruímos anteriormente, poderemos ver de que forma o PIB per capita de Brasil, Botsuana e Coreia do Sul se comportaram ao longo do tempo.

```
gapminderCountries %>%
  ggplot() +
  geom_line(aes(x = year,
                y = gdpPercap/10^3,
                color = country)) +
  labs(title = "Gapminder",
       subtitle = "1957-2007",
       color = "País") +
  xlab("Ano") +
  ylab("PIB per capita \n (US$ mil)")
```

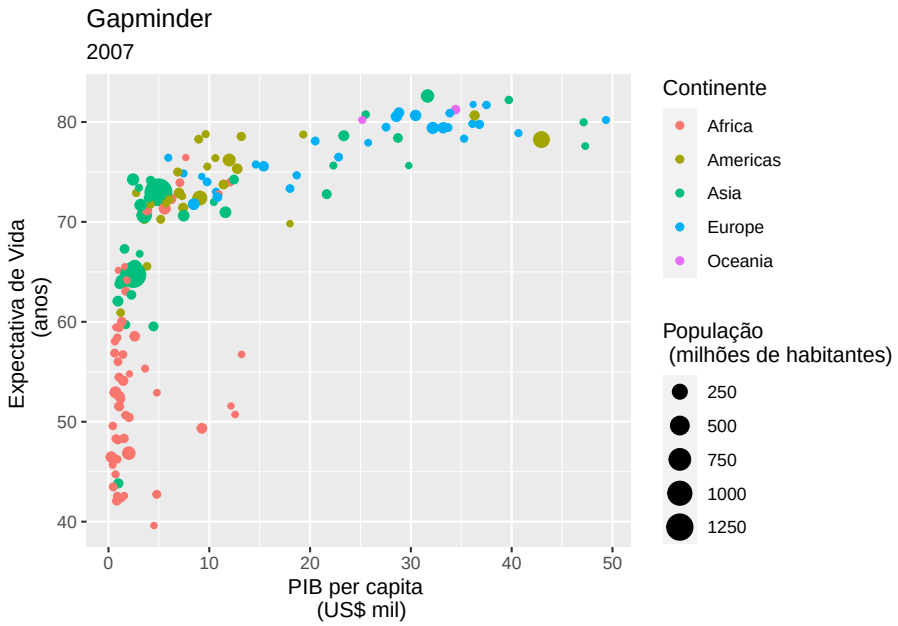


3.2 Gráfico de dispersão

Gráficos de dispersão permitem visualizar a relação entre duas variáveis numéricas, o que nos permite visualizar possíveis correlações. Utilizaremos o dataset `gapminder2007` para ver de que forma o PIB per capita está relacionado à expectativa de vida nos diferentes países do mundo em 2007.

Desta vez poderemos mostrar de que forma o `ggplot2` pode mapear diferentes atributos das observações em um mesmo gráfico. Em nosso exemplo, cada ponto representa um país, cada cor um continente, e o tamanho dos pontos é dado por sua população.

```
gapminder2007 %>%
  ggplot() +
  geom_point(aes(x = gdpPercap/10^3,
                 y = lifeExp,
                 color = continent,
                 size = pop/10^6)) +
  labs(title = "Gapminder",
       subtitle = "2007",
       color = "Continente",
       size = "População \n (milhões de habitantes)") +
  xlab("PIB per capita \n (US$ mil)") +
  ylab("Expectativa de Vida \n (anos)")
```


Gapminder
2007
Expectativa de Vida
(anos)
80
70
60
50
40
0
10
20
30
40
50
PIB per capita
(US$ mil)
Continente
Africa
Americas
Asia
Europe
Oceania
População
(milhões de habitantes)
250
500
750
1000
1250