Oficina de Estatistica Tutorial 1

Virgilio Mendes

15/07/2019

Tutorial 1

Breve introdução ao R

1. Introdução à linguagem R

Nesta parte inicial do curso, vamos apresentar alguns elementos fundamentais da linguagem e, simultaneamente, discutir algumas convenções que facilitam a legibilidade do código e evitam erros.

Este tutorial deve ser bastante útil para os dias desse curso. **Volte a ele sempre que necessário**.

1.1 Objetos, Vetores e Funções

Começaremos pela compreensão do que são objetos, vetores e funções, três elementos fundamentais para a utilização do R.

1.1.1 Objetos

Objetos são nomes que guardam informações (como números, textos,bancos de dados, etc) que podem ser acessados a qualquer momento da sua sessão.

Podemos criar um objeto com os operadores `<-` ou `=`. Apesar de ambas as possibilidades, recomendo a utilização do `<-` uma vez que o `=` será utilizado para outros fins, evitando confusões.

```
x <- 1
y = 4
```

Se não colocarmos nada após o operador, o que acontece?

```
#x <-
```

Caso o objetivo seja criar um objeto vazio, isso é indicado com `NULL`.

```
x <- NULL
```

Existem algumas recomendações para se nomear objetos:

- O que o R não permite?

Não é possível criar nomes que começam por (1) números, (2) caracteres especiais ou (3) com espaços:

```r
# Exemplos

# 1x <- 1 # (1)
#
# _a <- 1 # (2)
# .1 <- 1

#meu objeto <- 1 # (3)
```

- O que devemos evitar?

Devemos evitar (1) o uso de letras maiúsculas, pois o R é *case sensitive*, (2) o uso de nomes iguais aos de funções, (3) evitar acentos ou caracteres especiais, pois eles podem não abrir em outras sessões do R, a depender do *encoding* adotado pelo usuário, (4) usar o nome de objetos já existentes no seu ambiente, pois o R sobrescreverá o objeto existente.

```r
# Exemplos

MAIUSCULA <- 5 # (1)
maiuscula <- 5

rm <- 1 # (2)

ação <- 1 # (3)
```

- Quais são boas práticas de nomeação de objetos?

Além de evitar as práticas descritas acimas, uma importante dica para auxiliar o usuário e futuros leitores do código é a utilização do _ para espaçamento em nomes de objetos.

```r
# Exemplos

meuobjeto <- 1 # Ruim
meu_objeto <- 1 # Bom
```

Para verificar quais os nomes de objetos estão sendo utilizados, você pode checar seu *global environment*, ou usar a funcao `ls`, que imprimirá esses nomes no *console* do R.

```r
ls()
```

```
## [1] "ação"       "maiuscula"  "MAIUSCULA"  "meu_objeto" "meuobjeto"
## [6] "rm"         "x"          "y"
```

1.1.2 Vetores e introdução de classe de objetos

Objetos comportam uma série de informações. Dentre elas, uma daquelas que usaremos com grande frequência são os **vetores**. Vetores são um conjunto de informações atribuídas em uma única linha.

As informações guardadas em objetos podem ser de diferentes *classes*. Começaremos pelas classes numérica (`numeric`) e texto (`character`).

Vetores numéricos podem ser criados de diversas formas, e podem conter valores positivos, negativos, inteiros ou decimais:

```r
vetor_1 <- 1:10
vetor_2 <- c(0.1, -3.5, 2.556, 5.1)
```

Vetores `character` contém textos. Textos são definidos pelo uso de `" "`. Assim como em informações da classe `numeric`, podemos criar vetores com textos:

```r
exemplo <- "texto"
vetor_3 <- c("Meu", "exemplo", "de", "vetor")
```

Estas não são as únicas classes de objetos que existem e que utilizaremos. Outras serão apresentados à medida que progredirmos no curso.

1.1.3 Apresentando funções e retornando às classes de objetos

Funções são as principais ferramentas do R para o desempenho de tarefas. O R apresenta algumas funções básicas, mas usuários podem criar suas próprias funções. Como exemplos para compreender a estrutura básica de uma função, veremos a função `class` que nos diz qual a classe de um objeto.

```r
class(vetor_1) # integer
```

```
## [1] "integer"
```

```r
class(vetor_2) # numeric
```

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

```r
class(vetor_3) # character
```

```
## [1] "character"
```

Além da função que nos diz a classe de um objeto, podemos "perguntar" se ele é de certa classe. O resultado dessas funções nos trazem um nova classe de objeto: `logical`. Objetos lógicos são resultados *verdadeiro/falso* (`TRUE` ou `FALSE`) de operadores lógicos.

```r
is.numeric(vetor_1) # TRUE
```

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

```r
is.numeric(vetor_3) # FALSE
```

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

```r
objeto_logico <- is.numeric(vetor_1)
class(objeto_logico) # logical
```

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

O R permite a criação de objetos/vetores formados por elementos de classes diferentes. Contudo, ao uní-los em um único vetor, a classe do objeto se torna aquela que permite conversão e manipulação dos dados.

```r
a_1 <- c(2, "character") # R converte para character
a_2 <- c(FALSE, 3) # R converte para numeric (TRUE vira 1, FALSE vira 0)
a_3 <- c("Texto", T)  # R converte para character
a_4 <- c(1.3, -2) # R converte para numeric
```

Podemos usar funções para forçar a conversão de uma classe para outra

```r
as.numeric(TRUE) # Transforma TRUE (que e' logical) para numeric
```

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

```r
as.character(1.2) # Transforma 1.2 em character
```

```
## [1] "1.2"
```

Isso nem sempre funciona. Se tentarmos converter texto para números, por exemplo, o R não saberá o que fazer (pois a operação não faz sentido) e retornará um aviso.

```r
as.numeric("Meu texto")
```

```
## Warning: NAs introduzidos por coerção
```

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

O resultado da última operação é `NA`: o *missing* do R (geralmente, missings são indicados em bases de dados como 99999999 ou semelhantes). Note, também, que `NA` é uma classe própria.

```r
is.numeric(NA) # NA != numerico
```

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

```r
is.character(NA) # NA != texto
```

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

```r
is.na(NA) # NA = NA!
```

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

1.2 Manipulando objetos

Vimos que objetos podem conter diferentes classes de informação e que podemos aplicar funções a eles. Veremos agora diferentes formas em que é possivel manipular objetos (e vetores, por extensão).

Com objetos numéricos, podemos fazer qualquer operação matemática. Vamos então criar um objeto numérico, acessar diferentes valores dentro dele e fazer operações.

```r
# Criar o objeto
meu_objeto <- c(0:50, 60, 70, 80, 90)
meu_objeto
```

```
##  [1]  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22
## [24] 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45
## [47] 46 47 48 49 50 60 70 80 90
```

```r
# "Acessar" o valor de determinado elemento do vetor
meu_objeto[1] # acessa o 1o elemento do vetor 'meu_objeto'
```

```
## [1] 0
```

```r
meu_objeto[55] # acessa 55o elemento do vetor 'meu_objeto'
```

```
## [1] 90
```

```r
# Por meio dessa selecao de valores, podemos fazer operacoes:
meu_objeto[1] + meu_objeto[10] + meu_objeto[55]
```

```
## [1] 99
```

```r
meu_objeto[1] + meu_objeto[10] * meu_objeto[55]
```

```
## [1] 810
```

```r
meu_objeto[1] - meu_objeto[10]
```

```
## [1] -9
```

```r
meu_objeto[10] ^ meu_objeto[3]
```

```
## [1] 81
```

Podemos ainda usar funções para calcular algumas estatísticas descritivas, guardá-las em objetos e unir todas essas informações em um vetor.

```r
# Calcular algumas estatisticas descritivas:
mean(meu_objeto) # media
```

```
## [1] 28.63636
```

```r
max(meu_objeto)  # maximo
```

```
## [1] 90
```

```r
min(meu_objeto)  # minimo
```

```
## [1] 0
```

```r
# Guardar essas informacoes em objetos
media  <- mean(meu_objeto)
maximo <- max(meu_objeto)
minimo <- min(meu_objeto)

# Unir em um vetor
descritivas <- c(media, maximo, minimo)
descritivas
```

```
## [1] 28.63636 90.00000  0.00000
```

1.2.1 Matrizes

Para explorar mais a manipulação de objetos, vamos trabalhar com um nova classe de objeto: as **matrizes**. Basicamente, matrizes são vetores com 2 dimensões: x linhas e y colunas, cuja *estrutura é semelhante àquela de bancos de dados.*

```r
matrix(nrow = 2, ncol = 2)
```

```
##      [,1] [,2]
## [1,]   NA   NA
## [2,]   NA   NA
```

Por exemplo, podemos criar uma matriz com duas colunas (ou "variáveis", caso fosse um banco de dados) e 100 linhas ("observações"). Até o momento, estudamos funções que exigiam somente um argumento: o nome dos objetos. Porém, certas funções exigem mais de um argumento para funcionarem, como a função `matrix`. No caso, o primeiro argumento (1) dá o valor a ser utilizado para preenchimento da matriz, o segundo (100) é o número de linhas, e o terceiro (2) o número de colunas.

```r
minha_matriz <- matrix(1, 100, 2)
```

Na ajuda de uma função podemos ver a sua estrutura básica e identificar quais os argumentos vinculados a ela. Além disso, a ajuda nos descreve o que a função faz e apresenta exemplos de utilização. Para acessar a ajuda utilizamos uma interrogação na frente do nome da função ou utilizamos a função `help`. A ajuda do R será útil em boa parte das vezes em que você recorrer a ela. Porém, ela não é infalível. Muitas vezes precisamos de informações que não se encontram na ajuda. Nesses casos, podemos recorrer ao Google e a uma série de fóruns e blogs, como o stackoverflow e o R-bloggers.

```r
?matrix
help(matrix)
```

Poderíamos reescrever a função que criou o objeto `minha_matriz` utilizando o nome dos argumentos. Ao usar esses nomes, os argumentos não precisam estar na ordem "correta", indicada na descrição da função.

```r
minha_matriz <- matrix(ncol = 2, data = 1:200, nrow = 100)
```

Como matrizes funcionam de forma semelhante a vetores, podemos acessar cada elemento dela do mesmo jeito que fazemos com vetores normais. Mas, se quisermos acessar uma coluna ou linha inteira, precisamos usar um indexador mais complexo: [x, y], onde **x** indica o número da linha que queremos acessar, e **y** o número da coluna. Embora pareça complexo, este sistema de indexação é muito útil para se trabalhar com o R, principalmente com bases de dados (data frames).

```r
# Acessando elementos individuais da matriz:
minha_matriz[1]
```

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

```r
# Seleciona a primeira linha inteira da matriz:
minha_matriz[1, ]
```

```
## [1]   1 101
```

```r
# Seleciona a segunda coluna da matriz, tambem em formato de vetor
minha_matriz[, 2] # retorna um vetor
```

```
##   [1] 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117
##  [18] 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134
##  [35] 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151
##  [52] 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168
##  [69] 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185
##  [86] 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200
```

```r
# Porem, ao usarmos o argumento drop = F, a selecao retorna uma matriz, com uma coluna
minha_matriz[, 2, drop = FALSE]
```

```
##        [,1]
##   [1,]  101
##   [2,]  102
##   [3,]  103
##   [4,]  104
##   [5,]  105
##   [6,]  106
##   [7,]  107
##   [8,]  108
##   [9,]  109
##  [10,]  110
##  [11,]  111
##  [12,]  112
##  [13,]  113
##  [14,]  114
##  [15,]  115
##  [16,]  116
##  [17,]  117
##  [18,]  118
##  [19,]  119
```

```
##  [20,]  120
##  [21,]  121
##  [22,]  122
##  [23,]  123
##  [24,]  124
##  [25,]  125
##  [26,]  126
##  [27,]  127
##  [28,]  128
##  [29,]  129
##  [30,]  130
##  [31,]  131
##  [32,]  132
##  [33,]  133
##  [34,]  134
##  [35,]  135
##  [36,]  136
##  [37,]  137
##  [38,]  138
##  [39,]  139
##  [40,]  140
##  [41,]  141
##  [42,]  142
##  [43,]  143
##  [44,]  144
##  [45,]  145
##  [46,]  146
##  [47,]  147
##  [48,]  148
##  [49,]  149
##  [50,]  150
##  [51,]  151
##  [52,]  152
##  [53,]  153
##  [54,]  154
##  [55,]  155
##  [56,]  156
##  [57,]  157
##  [58,]  158
##  [59,]  159
##  [60,]  160
##  [61,]  161
##  [62,]  162
##  [63,]  163
##  [64,]  164
##  [65,]  165
##  [66,]  166
##  [67,]  167
##  [68,]  168
##  [69,]  169
##  [70,]  170
##  [71,]  171
##  [72,]  172
##  [73,]  173
```

```
##  [74,]  174
##  [75,]  175
##  [76,]  176
##  [77,]  177
##  [78,]  178
##  [79,]  179
##  [80,]  180
##  [81,]  181
##  [82,]  182
##  [83,]  183
##  [84,]  184
##  [85,]  185
##  [86,]  186
##  [87,]  187
##  [88,]  188
##  [89,]  189
##  [90,]  190
##  [91,]  191
##  [92,]  192
##  [93,]  193
##  [94,]  194
##  [95,]  195
##  [96,]  196
##  [97,]  197
##  [98,]  198
##  [99,]  199
## [100,]  200
```

```
# Se quisermos acessar o elemento que se encontra na 1a linha da 2a coluna:
minha_matriz[1, 1]
```

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

```
# Podemos ainda guardar os elementos acessados em um objeto:
m1 <- minha_matriz[2, 1]
m2 <- minha_matriz[1, ]
```

1.3 Pacotes

Até o momento, trabalhamos somente com funções "embutidas" no R, o que é conhecido como *R Base*. Porém, uma das características mais atrativas do R é o extenso numero de **pacotes** desenvolvidos por usuários, que possuem as mais diversas utilidades. Para vocês terem uma pequena ideia de alguns pacotes úteis para pesquisas em ciências sociais (ou áreas afins), o pesquisador Denisson Silva apresenta alguns deles. Dentre eles, encontram-se pacotes para trabalhar com dados eleitorais, do IBGE ou do SUS, por exemplo.

Para carregar um pacote no R, usamos a função `library`. Como argumento, incluímos o nome do pacote que desejamos carregar. Caso o nome do pacote esteja errado, o R retornará um erro. Este procedimento deve ser repetido sempre que uma nova sessão do R for iniciada.

Vamos carregar um pacote que usaremos bastante daqui para frente: o dplyr. Caso ele não esteja instalado no seu computador, o R apresentará uma mensagem de erro. Nesse caso, é necessário instalar o pacote antes de utilizá-lo, por meio da função `install.packages`.

```r
# No caso da funcao install.packages, o nome do pacote deve vir entre aspas.
#install.packages("dplyr")
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

Uma boa prática de redação de scripts é sempre redigir, no topo deles, os pacotes que serão utilizados para conduzir os procedimentos programados. Assim, ao se executar o script desde o início, teremos todas as habilitado todas funções necessárias.

Pausa

Pare, respire, beba uma água, esclareça suas dúvidas e vamos seguir!

2. Carregar e visualizar bancos de dados

O R é um software extremamente flexível no que se refere aos formatos de arquivos com os quais pode trabalhar. Dessa forma, ele consegue abrir bancos de dados oriundos de diferentes programas, de maneira simples e (na maior parte das vezes) rápida.

Neste curso, vamos trabalhar somente com dados em formato .csv. Porém, se você tiver necessidade de abrir arquivos em outro formato, leia este capítulo do livro de Fernando Meireles e Denisson Silva.

2.1 Carregando um banco de dados

Uma funcão "coringa" para se abrir bancos é a `read.table`. Vamos abrir um banco de dados de exemplo direto do repositório do curso.

```r
#banco <- read.table("https://raw.githubusercontent.com/lgelape/Curso_2018_ENAP/master/bancos/AtlasBras
```

Aparentemente, funcionou. Mas será que ele realmente abriu o banco de dados corretamente? Vamos clicar na janelinha que está no final da linha determinada por "banco" no seu Environment.

Ao abrirmos o banco de dados, ele claramente não está correto. Aparentemente, ele só tem 2 variáveis, os nomes dos municípios não estão escritos corretamente e os valores da segunda variável não parecem fazer sentido.

O que pode ter dado errado?

A função `read.table` possui diversos argumentos que podem ser acrescentados para permitir a abertura correta de um banco de dados. Quais seriam eles?

Normalmente, trata-se do argumento `sep`, que indica qual o separador das células do banco. Outros argumentos importantes são:

- `dec` estabelece qual o símbolo dos decimais;
- `header` diz se a 1ª linha do banco traz os nomes das variáveis;
- `na.strings` estabelece como os missings estão escritos no banco original, e que serão traduzido para o R como `NA`; e
- `skip` indica o n. de linhas que deve ser pulado no inicio da leitura do banco.

Porém, como descobrimos isto? Se a pessoa que produziu o banco de dados seguiu os melhores procedimentos para tanto, existirá um arquivo "leia-me" com essas informações. Caso o banco não venha acompanhado de um arquivo desse tipo, podemos abrir o arquivo e verificar tais informações. Mas, o arquivo pode ser muito grande e de difícil manuseio. Nesses casos, a tentativa e erro (ou o pedido de ajuda para algum colega que já tenha trabalhado com o banco!) é uma opção válida.

Para este banco de dados, que é um recorte do Atlas do Desenvolvimento Humano no Brasil, que eu converti para o formato .csv, as células estão separadas por `;`.

```
#banco <- read.table("https://raw.githubusercontent.com/lgelape/Curso_2018_ENAP/master/bancos/AtlasBras
```

Incluir o argumento `sep = ";"` não foi o suficiente! Por quê?

O argumento `encoding` é utilizado para especificar a codificação segundo a qual os caracteres devem ser lidos. No nosso caso (e em vários casos de bancos de dados produzidos no Brasil), ele deve ser usado para permitir a leitura do banco sob a codificação `"latin1"`, que permite a leitura do alfabeto latino. Outras opções que podem ser tentadas são a `"windows-1252"` ou a `"utf-8"`.

Além deles, vamos também incluir os argumentos `header = T` e `dec = ","` para especificar que a primeira linha do banco traz os nomes das variáveis e que o marcador que define os decimais é a vírgula.

```
banco <- read.table("/home/virgilioam/Área de trabalho/Modus 2019/Oficina de Estatística/Oficina de Esta
                    sep = ";", encoding = "latin1", header = T, dec = ",")
```

2.2 Exploração preliminar de um banco de dados

Como vimos antes, ao guardarmos as informações de um banco de dados em um objeto, podemos visualizar esse banco ao clicar no ícone correspondente. Para realizar a mesma operação por meio de um código, podemos usar a função `View`.

```
View(banco) # nao se esquecam que o V e maiusculo!
```

Contudo, no R precisamos evitar trabalhar com um olhar constante para o banco, uma vez que uma das principais vantagens (a capacidade de se trabalhar com um alto volume de dados em relativamente pouco tempo) acabará se perdendo.

Sendo assim, para não precisarmos olhar constantemente para o banco, o R apresenta algumas funções que nos permitem "espiá-lo" e verificar, ao menos preliminarmente, se ele foi aberto de forma correta. Aqui veremos 2 delas.

A primeira dessas funções é a `head`, que apresenta os nomes das variaveis e algumas observações iniciais do banco em questão.

```
head(banco)
```

```
##                      munic gini_91 gini_00 gini_2010 pop_91 pop_00  pop10
## 1    Abadia de GoiÃ¡s (GO)    0.44    0.50      0.42   4227   4971   6876
```

```
## 2 Abadia dos Dourados (MG)    0.46    0.50      0.47   6492   6446   6704
## 3          AbadiÃ¢nia (GO)    0.52    0.54      0.43   9402  11452  15757
## 4             AbaetÃ© (MG)    0.52    0.58      0.54  20689  22360  22690
## 5          Abaetetuba (PA)    0.50    0.60      0.53  99989 119152 141100
## 6             Abaiara (CE)    0.44    0.60      0.48   7889   8385  10496
##   rpc_91 rpc_00 rpc_10 pobres_91 pobres_00 pobres_10
## 1 266.22 385.66 574.96     33.17     27.18      6.18
## 2 247.43 370.42 596.18     40.41     25.41      7.94
## 3 261.01 330.07 519.87     42.30     32.47      8.45
## 4 263.12 498.82 707.24     47.17     19.13      6.69
## 5 169.25 206.84 293.01     63.64     61.15     38.95
## 6  83.34 143.75 229.74     85.15     67.72     44.64
```

Além da `head`, temos a função `glimpse`, que pode ser utilizada quando carregamos o pacote `dplyr`.

```
library(dplyr)
glimpse(banco)
```

```
## Observations: 5,565
## Variables: 13
## $ munic     <fct> Abadia de GoiÃ¡s (GO), Abadia dos Dourados (MG), Aba...
## $ gini_91   <dbl> 0.44, 0.46, 0.52, 0.52, 0.50, 0.44, 0.52, 0.47, 0.53...
## $ gini_00   <dbl> 0.50, 0.50, 0.54, 0.58, 0.60, 0.60, 0.53, 0.55, 0.43...
## $ gini_2010 <dbl> 0.42, 0.47, 0.43, 0.54, 0.53, 0.48, 0.46, 0.55, 0.44...
## $ pop_91    <int> 4227, 6492, 9402, 20689, 99989, 7889, 8052, 10182, 1...
## $ pop_00    <int> 4971, 6446, 11452, 22360, 119152, 8385, 8322, 12275,...
## $ pop10     <int> 6876, 6704, 15757, 22690, 141100, 10496, 8316, 17064...
## $ rpc_91    <dbl> 266.22, 247.43, 261.01, 263.12, 169.25, 83.34, 136.5...
## $ rpc_00    <dbl> 385.66, 370.42, 330.07, 498.82, 206.84, 143.75, 204....
## $ rpc_10    <dbl> 574.96, 596.18, 519.87, 707.24, 293.01, 229.74, 290....
## $ pobres_91 <dbl> 33.17, 40.41, 42.30, 47.17, 63.64, 85.15, 74.15, 69....
## $ pobres_00 <dbl> 27.18, 25.41, 32.47, 19.13, 61.15, 67.72, 51.44, 62....
## $ pobres_10 <dbl> 6.18, 7.94, 8.45, 6.69, 38.95, 44.64, 31.74, 47.77, ...
```

2.3 Identificação de variáveis

Ao explorarmos o banco com o qual estamos trabalhando, vimos que é possivel identificar, de forma geral, o nome e tipos de variáveis.

Porém, por vezes é necessário identificá-los em contextos mais aplicados, que não do panorama do banco como um todo.

Para verificarmos o nome de todas as variáveis do nosso banco, podemos usar a função `names`, que nos retorna um vetor (lembra-se dele?) com os nomes das variáveis.

```
names(banco)
```

```
##  [1] "munic"     "gini_91"   "gini_00"   "gini_2010" "pop_91"
##  [6] "pop_00"    "pop10"     "rpc_91"    "rpc_00"    "rpc_10"
## [11] "pobres_91" "pobres_00" "pobres_10"
```

E se quisermos saber qual o tipo dessa variável (categórica x contínua)? Assim como para os vetores, utilizamos a funcao `class`.

A função `class` aplicada ao banco nos retorna a classe do objeto completo do banco de dados (um data.frame, portanto).

```
class(banco)
```

```
## [1] "data.frame"
```

Para acessarmos uma das variáveis desse banco, utilizamos o operador `$`.

```
class(banco$Índice.de.Gini.1991)
```

```
## [1] "NULL"
```

```
class(banco$Espacialidades)
```

```
## [1] "NULL"
```

Opa, a variavel 'Espacialidades' é de uma classe diferente `factor`. Esta classe é utilizada para se trabalhar com variaveis categóricas no R (assim como a `character`). Veremos um pouco mais sobre ela futuramente.

Além disso, vocês devem ter observado que os nomes dessas variáveis vão contra todas as dicas que colocamos sobre como criar nomes de objetos (e, por extensão, de variáveis) no R. Como corrigir isto? Ao longo do curso vou mostrar como isso pode ser feito, mas vocês não precisam se preocupar, pois todos os bancos de dados que usaremos estarão com nomes que nos evitarão problemas.

Resumo do conteúdo trabalhado no Tutorial 1:

- Nos familiarizamos com a interface do *R Studio*.
- Aprendemos a criar objetos e guardar informações neles (com '<-').
- Discutimos algumas dicas na criação de nomes de objetos.
- Vimos que os objetos podem ser de várias classes (character, numeric, NA...).
- Conhecemos as funções, os *verbos* do R, e sua estrutura.
- Exploramos vetores, realizando operações e seleções a partir deles.
- Aprendemos a pedir ajuda (help e ?) dentro do R e entendê-la.
- Trabalhamos o básico de matrizes, objetos com lógica de funcionamento semelhante à bases de dados.
- Conhecemos a ideia de pacotes, que aumentam substancialmente as potencialidades do R.
- Aprendemos a abrir um banco de dados no R, com alguns cuidados que podem ser necessários.
- Utilizamos algumas funções para fazer uma exploração inicial do banco: `View`, `head` e `glimpse`.
- Aprendemos como acessar os nomes e tipos das variáveis da base de dados.

Orientações Finais

Revise o material para o caso de dúvidas. Se estiver confortável, siga para o Tutorial 2, onde vamos calcular algumas variáveis do banco de dados gerado a partir do questionário respondido pelos colegas.