Módulo 2

2025-09-14

Módulo 2

Introdução e objetivos

Objetivos

Entender o R e como ele interage com o computador. Saber consultar sua documentação. Há mais de 18 mil pacotes oficiais no CRAN, multiplataforma, inpumeros recursos gráficos. Download e pipipipopopó. IDEs/Editores disponíveis para R: Rstudio, Rcode, Tinn-R; VIM, Visual Studio Code (VS Code), GNU Emacs, Spacemacs ou Doom-Emacs usando ESS.

Primeiros passos

- CLi - command line interface
- REPL - Read, eval, print and loop

O básico é ter uma linha/cadeia de comando, coisa que já sabemos. Basicamente, é a linha de comandos redigidos.

Modos de uso

- Modo REPL: script com instruções, que são avaliadas no console, supervisionadas pelo analista, e salvas, duplicadas ou modificadas de acordo com a necessidade.
- Modo Batch: script pode ser executado sem supervisão; é usado em ambientes de produção ou simulação computacional.

As instruções ou comentários são marcados pelas #. Após elas, tudo no código é entendido como comentário. Linhas sem # são executadas no terminal. Instruções podem ser dadas em uma linha ou em várias. Para mais de uma instrução em uma mesma linha, separar cm ponto e vírgula (;). Ex:

```
# soma simples
x <- 57 + 19
print(x)
```

```
## [1] 76
```

```
# Meu IMC
82/1.70^2
```

```
## [1] 28.3737
```

```r
# Instrução em uma única linha
2 + 5 + 8 - 15 + 9
```

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

```r
# Instrução em várias linhas
2 +
  9 -
  7 +
  2
```

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

Recomendações:

1. Evitar passar de 72 ou 80 caracteres;
2. Manter o código devidamente indentado (CTRL + i);
3. Evitar muitas instruções em uma mesma linha.

Ordem de execução dos comandos

É a mesma ordem de execução matemática usual

```r
2 + 2 * 4
```

```
## [1] 10
```

```r
2 + (2 * 4)
```

```
## [1] 10
```

```r
2^2 * 5^3
```

```
## [1] 500
```

```r
(2^2) * (5^3)
```

```
## [1] 500
```

Área e espaço de trabalho

Criação de objetos

Ao fazer atribuições, são criados objetos na área de trabalho. Usa-se <- para atribuir algo a um objeto (alt + i). Objetos podem ser reusados para criar outros (é a intenção). Ex:

```r
# Criação de objetos para fazer operações
peso <- 82
altura <- 1.70
IMC <- peso/altura^2
IMC
```

```
## [1] 28.3737
```

```
# Listar objetos da área de trabalho
ls()
```

```
## [1] "altura" "IMC"    "peso"   "x"
```

```
# Apagar objetos
rm(IMC)
ls() # mostra o conteúdo do .GlobalEnv, a área de trabalho
```

```
## [1] "altura" "peso"   "x"
```

```
# peso  altura
```

O comando ls() mostra o conteúdo do .GlobalEnv, a área de trabalho, mas também há objetos em outros ambientes e espaços. Esses são os espaços dos pacotes. Quando não encontra um objeto em .GlobalEnv, ele parte para o próximo ambiente de trabalho. search() retorna a lista de espaços de trabalho. Cada pacote tem o seu próprio. Ex:

```
# Espaços de trabalho
women # já existente dentro do R
```

```
##    height weight
## 1      58    115
## 2      59    117
## 3      60    120
## 4      61    123
## 5      62    126
## 6      63    129
## 7      64    132
## 8      65    135
## 9      66    139
## 10     67    142
## 11     68    146
## 12     69    150
## 13     70    154
## 14     71    159
## 15     72    164
```

```
# Para listar o conteúdo de um determinado espaço
ls("package:datasets") # lista todos os datasets integrados ao R
```

```
##   [1] "ability.cov"          "airmiles"             "AirPassengers"
##   [4] "airquality"           "anscombe"             "attenu"
##   [7] "attitude"             "austres"              "beaver1"
##  [10] "beaver2"              "BJsales"              "BJsales.lead"
##  [13] "BOD"                  "cars"                 "ChickWeight"
##  [16] "chickwts"             "co2"                  "CO2"
##  [19] "crimtab"              "discoveries"          "DNase"
##  [22] "esoph"                "euro"                 "euro.cross"
```

```
##  [25] "eurodist"              "EuStockMarkets"        "faithful"
##  [28] "fdeaths"               "Formaldehyde"          "freeny"
##  [31] "freeny.x"              "freeny.y"              "gait"
##  [34] "HairEyeColor"          "Harman23.cor"          "Harman74.cor"
##  [37] "Indometh"              "infert"                "InsectSprays"
##  [40] "iris"                  "iris3"                 "islands"
##  [43] "JohnsonJohnson"        "LakeHuron"             "ldeaths"
##  [46] "lh"                    "LifeCycleSavings"      "Loblolly"
##  [49] "longley"               "lynx"                  "mdeaths"
##  [52] "morley"                "mtcars"                "nhtemp"
##  [55] "Nile"                  "nottem"                "npk"
##  [58] "occupationalStatus"    "Orange"                "OrchardSprays"
##  [61] "penguins"              "penguins_raw"          "PlantGrowth"
##  [64] "precip"                "presidents"            "pressure"
##  [67] "Puromycin"             "quakes"                "randu"
##  [70] "rivers"                "rock"                  "Seatbelts"
##  [73] "sleep"                 "stack.loss"            "stack.x"
##  [76] "stackloss"             "state.abb"             "state.area"
##  [79] "state.center"          "state.division"        "state.name"
##  [82] "state.region"          "state.x77"             "sunspot.m2014"
##  [85] "sunspot.month"         "sunspot.year"          "sunspots"
##  [88] "swiss"                 "Theoph"                "Titanic"
##  [91] "ToothGrowth"           "treering"              "trees"
##  [94] "UCBAdmissions"         "UKDriverDeaths"        "UKgas"
##  [97] "USAccDeaths"           "USArrests"             "UScitiesD"
## [100] "USJudgeRatings"        "USPersonalExpenditure" "uspop"
## [103] "VADeaths"              "volcano"               "warpbreaks"
## [106] "women"                 "WorldPhones"           "WWWusage"
```

```
# Caso crie-se um objeto com um nome de um dataset do R, é possível acessá-lo por meio de
datasets::women
```

```
##    height weight
## 1      58    115
## 2      59    117
## 3      60    120
## 4      61    123
## 5      62    126
## 6      63    129
## 7      64    132
## 8      65    135
## 9      66    139
## 10     67    142
## 11     68    146
## 12     69    150
## 13     70    154
## 14     71    159
## 15     72    164
```

Diretório de Trabalho

O diretório de trabalho é o local no sistema operacional para onde o R está apontando. Isto é, de onde ele lê e escreve arquivos de modo padrão. Pode-se definir comandos (recomendável) ou usando *RStudio IDE > Session > Setting Work Directory*. Ex:

```r
# Diretório atual de trabalho
# getwd()

# Trocar diretório de trabalho
# setwd("~/Downloads") # moveria para os downloads

# Lista o conteúdo do diretório atual
# dir()

# Instalação de pacotes
# install.packages("tidyverse")
```

Arquivos, pacotes e documentação

Arquivos da linguagem R

Há inúmeros arquivos para melhorar a experiência dos usuário:

- .Rhistory: arquivo de texto que salva o histórico de instruções executadas.
- .RData: arquivo binário que salva objetos na área de trabalho. Serve para restaurá-los e é útil quando o processamento é demorado.
- .Rproj: arquivo que define configurações do projeto. O uso também pode ser configurado por meio do caminho *RStudio IDE > Tools > Global Options*
- .Rprofile: Arquivos de configurações lido no início das seções. Carrega pacotes muito usados e configura opções, mensagem de boas vindas e diagnóstico do sistema, pode ser definido por projeto ou por usuário.

Instalação de pacotes

Pacotes São coleções de funções e conjuntos de dados organizados e documentados. Um pacote contém código R e eventualmente códigos de outras linguagens. Podem depender de *libs* do sistema operacional.

Formas de instalação Pacotes podem ser instalados de repositórios: CRAN, Biocondutor, MRAN, etc; de arquivos de instalação: *.tar.gz; De repositórios GIT: Github, Gitlab, etc.

```r
# Para instalar um pacote do repositório.
# install.packages("tidyverse")

# Para carregar o pacote e usá-lo.
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   4.0.0     v tibble    3.3.0
## v lubridate 1.9.4     v tidyr     1.3.1
## v purrr     1.1.0
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```r
# Para ver o conteúdo dele.
ls("package:tidyverse")
```

```
## [1] "tidyverse_conflicts" "tidyverse_deps"      "tidyverse_logo"
## [4] "tidyverse_packages"  "tidyverse_sitrep"    "tidyverse_update"
```

```r
# Documentação do pacote.
help(package = "tidyverse")

# Para ver onde foi instalado.
system.file(package = "tidyverse")
```

```
## [1] "C:/Users/anton/AppData/Local/R/win-library/4.5/tidyverse"
```

```r
# Os caminhos para endereços de instalação.
.libPaths()
```

```
## [1] "C:/Users/anton/AppData/Local/R/win-library/4.5"
## [2] "C:/Program Files/R/R-4.5.1/library"
```

```r
# Para remover o pacote da sessão.
detach("package:tidyverse", unload = TRUE)

# Funções relacionadas a pacotes.
apropos("package")
```

```
##  [1] "$.package_version"              ".packages"
##  [3] "as.package_version"             "aspell_package_C_files"
##  [5] "aspell_package_R_files"         "aspell_package_Rd_files"
##  [7] "aspell_package_vignettes"       "available.packages"
##  [9] "download.packages"              "find.package"
## [11] "findPackageEnv"                 "format.packageInfo"
## [13] "getPackageName"                 "install.packages"
## [15] "installed.packages"             "is.package_version"
## [17] "make.packages.html"             "methodsPackageMetaName"
## [19] "new.packages"                   "old.packages"
## [21] "package.skeleton"               "package_version"
## [23] "packageDate"                    "packageDescription"
## [25] "packageEvent"                   "packageHasNamespace"
## [27] "packageName"                    "packageNotFoundError"
## [29] "packageSlot"                    "packageSlot<-"
## [31] "packageStartupMessage"          "packageStatus"
## [33] "packageVersion"                 "path.package"
## [35] "print.packageInfo"              "promptPackage"
## [37] "remove.packages"                "setPackageName"
## [39] "suppressPackageStartupMessages" "update.packages"
```

Documentação interna

Consiste de documentação de objetos e funções. Tutoriais chamados de vinhetas (vignettes). Existem funções especícas para a consulta destes. Pode-se procurar na web também.

```r
# Duas formas iguais de chamar
# a documentação.
?women
```

```
## inicializando servidor httpd de ajuda ... concluído
```

```r
help(women)

# Procura por ocorrêncians de `women`.
help.search("women")

# Objetos que batem com o termo.
apropos("tukey")
```

```
## [1] "ptukey"   "qtukey"   "TukeyHSD"
```

```r
# Exibe as vinhetas de um pacote.
browseVignettes(package = "survival") # é outro tipo de documentação, mas não são padronizadas como a d

# Procura pelo termo no
# r-project.org.
RSiteSearch("spider plot")
```

```
## Uma busca foi submetida para https://search.r-project.org
## A página de resultados deverá abrir no seu navegador em breve
```

Campos oficiais da documentação:

- **Cabeçalho**: Indica o pacote.
- **Título**: Resumo do que são os objetos documentados
- **Descrição**: do que o objeto é/faz
- **Usage**: Como usar ou formas de montar as instruções
- **Arguments**: Quais os argumentos formais da função
- **Value**: O que a função retorna

Guia de sobrevivência do R

4 Principais funções para o iniciante

1. str()
2. ls()
3. apropos()
4. help()

Outras funções úteis

1. help.search()
2. help.start()

3. RSiteSearch()
4. browseVignettes()
5. vignette()
6. args()
7. class()
8. methods()
9. find()
10. example()
11. demo()

Operações aritméticas e lógicas

Operações básicas

```
2 + 3 # Soma.
```

```
## [1] 5
```

```
2 - 3 # Subtração.
```

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

```
2 * 3 # Multiplicação.
```

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

```
2/3 # Divisão.
```

```
## [1] 0.6666667
```

```
2^3 # Potenciação.
```

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

```
2^(1/3) # Radiciação.
```

```
## [1] 1.259921
```

```
10 %% 3 # Resto.
```

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

```r
10 %/% 3 # Parte inteira.
```

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

Logarítmo

```r
exp(2) # Exponencial neperiano.
```

```
## [1] 7.389056
```

```r
log(10) # Neperiano.
```

```
## [1] 2.302585
```

```r
log10(10) # Base 10.
```

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

```r
log2(10) # Base 2.
```

```
## [1] 3.321928
```

```r
log(10, base = 5) # Base qualquer.
```

```
## [1] 1.430677
```

Trigonométricas

```r
sin(3) # Seno.
```

```
## [1] 0.14112
```

```r
cos(3) # Cosseno.
```

```
## [1] -0.9899925
```

```r
tan(3) # Tangente.
```

```
## [1] -0.1425465
```

```r
asin(0.5) # Arco seno.
```

```
## [1] 0.5235988
```

```r
acos(0.5) # Arco cosseno.
```

```
## [1] 1.047198
```

```r
atan(0.5) # Arco tangente.
```

```
## [1] 0.4636476
```

Arredondamento

```r
round(pi, digits = 5)
```

```
## [1] 3.14159
```

```r
floor(pi) # Inteiro logo baixo.
```

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

```r
ceiling(pi) # Inteiro logo acima.
```

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

```r
trunc(pi) # Trunca em relação ao zero.
```

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

Operações lógicas

```r

x <- 3
y <- 4

2 == 2 # Igualdade.
```

Comparações de valor

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

```r
2 != 2 # Desigualdade.
```

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

```r
x <= y  # Outros operadores: "<", ">", and ">=".
```

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

```r
(2 < 5) & (7 >= 3) # Operador AND.
```

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

```r
(2 < 5) | (7 >= 3) # Operador OR.
```

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

```r
!(2 < 5) # Operador NOT.
```

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

```r
"a" == "b" # Compara strings.
```

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

```r
"a" < "b" # Ordem alfanumérica.
```

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

Tipos especiais **NA**: para valores ausentes. **NULL**: para objetos vazios. **Inf** e **-Inf**: para innitos. **NaN**: para resultados não razoavelmente denidos.

```r
5 + NA          # O resultado é NA.
```

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

```r
is.na(5 + NA)   # Verifica se é NA.
```

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

```r
10 + NULL       # Retorna objeto vazio.
```

```
## numeric(0)
```

```r
is.null(NULL)   # Verifica se é nulo.
```

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

```r
5/0             # Infinito.
```

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

```r
is.finite(5/0)  # Verifica se é finito.
```

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

```r
0/0             # Valor indeterminado.
```

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

```r
is.nan(0/0)     # Verifica se é not a number.
```

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

Estrutura de dados

Tipos de objetos

Praticamente tudo em R são objetos. Objetos têm duas características: tipo(type) e classe(class). O tipo de um objeto diz o que é possível fazer com ele. A classe diz quais métodos podemos aplicar em um objeto. Precisamos conhecer as principais estruturas e suas características. Principais tipos de objetos em R:

- Caracteres (character);
- Números reais (double e numeric);
- Números inteiros (integer);
- Lógico (logical);
- Complexo (complex);
- Raw (cru/bruto).

```r
inteiro <- 10L # O L força a interpretar o 10 como inteiro, e não real
real <- 10
is.numeric(inteiro)
```

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

```r
is.numeric(real)
```

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

```r
is.double(inteiro)
```

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

```r
is.double(real)
```

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

```r
is.integer(inteiro)
```

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

```r
is.integer(real)
```

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

Estruturas de dados

Conjuntos de dados são combinações dos de objetos. Como organizar tais dados é o que chamamos de estruturas de dados. O R vem equipado com diversas estruturas para trabalhar com dados. Essas estruturas são chamadas de classes, cujo comando é class(). Estruturas mais importantes em R:

- atomic vector (vetores atômicos).
- matrix (matrizes).
- list (listas).
- data.frame (tabelas).

Exemplos de criação de objetos:

```r
vetor_atomico <- c(1,2,3,4)

matriz <- matrix(c(1,0,0,1),2, 2) # dá dimensão (linhas e colunas) a um vetor

lista <- list("a" = 5, "b" = 5L,
"c" = "Letra") # diferentes tipos de objetos

tab <- data.frame("Nome" = c("Bent", "Jon"),
                  "Idade" = c(3, 4))
```

Vetores atômicos

Quando combinamos objetos de um único tipo temos os vetores **atômicos.**. Exemplo: `vetor_numerico <- c(1, 5, 11, 33)`. Como nomear objetos em R?

- Um nome pode consistir de letras, números, e _;
- Não pode conter palavras reservadas: TRUE, FALSE, NULL, if entre outras;
- Veja todas as palavras reservadas usando ? Reserved.

1. Vetor de caracteres. Ex: `vetor_caracter <- c("hello", "world")`.
2. Vetor Lógico. Ex: `vetor_logico <- c(TRUE, TRUE, FALSE)`.
3. Vetor combinado. Ex:

```r
vetor_numerico <- c(1, 5, 11, 33)
vetor_caracter <- c("hello", "world")
vetor_logico <- c(TRUE, TRUE, FALSE)

vetor_combinado <- c(vetor_numerico,
                     vetor_caracter,
                     vetor_logico,
                     "boo")

vetor_combinado
```

```
##  [1] "1"     "5"     "11"    "33"    "hello" "world" "TRUE"  "TRUE"  "FALSE"
## [10] "boo"
```

4. Coerção entre objetos. Ex: `class(vetor_combinado)`.

Como consultar o tipo de um objeto?

Usa-se `is.*` para consultar o tipo de um objeto. Ex:

```r
# Funções que começam com is.
# apropos("^is\\.")
is.integer(1)
```

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

```r
is.numeric(1)
```

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

```r
is.integer(1L)
```

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

```r
is.numeric(1L)
```

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

```r
is.character("Curitiba")
```

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

```r
y <- factor(c("Solteiro", "Casado"))
is.factor(y)
```

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

```r
is.character(y)
```

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

```r
is.logical(c(TRUE, FALSE))
```

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

Outra opção seria usar `typeof()`.

```r
x <- "caracter"
typeof(x) ## [1] "character"
```

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

```r
x <- 10
typeof(x) ## [1] "double"
```

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

```r
x <- 10L
typeof(x) ## [1] "integer"
```

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

```r
x <- TRUE
typeof(x) ## [1] "logical"
```

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

Conversão e classe de objetos

O R é uma linguagem automaticamente tipicada. Pode ser necessário converter um objeto para diferentes tipos. Exemplo:

```r
x <- 10
is.numeric(x)
```

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

```r
x <- as.logical(x)
is.numeric(x)
```

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

A classe `(class())` de um objeto orienta que tipo de métodos podemos usar. Métodos são funções genéricas do R que atuam de forma diferente de acordo com o tipo do objeto. Exemplo:

```r
# Um vetor numérico.
x <- c(1, 2, 3)
class(x)
```

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

```r
methods(class = "numeric")
```

```
##  [1] -             /             +             all.equal     as.data.frame
##  [6] as.Date       as.duration   as.interval   as.period     as.POSIXct
## [11] as.POSIXlt    as.raster     as_date       as_datetime   as_factor
## [16] as_mapper     coerce        Compare       full_seq      glyphJust
## [21] month         months        Ops           recode        scale_type
## [26] wday
## see '?methods' for accessing help and source code
```

É possível, ainda, criar objetos com atributos. Exemplos:

```r
# Numérico mas com valores nomeados.
notas <- c("João" = 7.8,
"Bianca" = 10,
"Eduarda" = 8.5)
class(notas)
```

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

```r
attributes(notas)
```

```
## $names
## [1] "João"    "Bianca"  "Eduarda"
```

```r
names(notas) # acessando os atributos do vetor
```

```
## [1] "João"    "Bianca"  "Eduarda"
```

```r
length(notas) # acessando o tamanho do vetor
```

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

Criando sequências estruturadas

Função `seq()` cria sequências estruturadas. Exemplo:

```r
# Sequencia de 1 a 7
1:7
```

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

```r
# Sequencia de 1 a 10 de 2 em 2
seq(from = 1, to = 10, by = 2)
```

```
## [1] 1 3 5 7 9
```

```r
# Sequencia de 1 a 20 de tamanho 7
seq(from = 1, to = 20, length.out = 7)
```

```
## [1]  1.000000  4.166667  7.333333 10.500000 13.666667 16.833333 20.000000
```

```r
# Sequencia começando em 1 de tamanho sete de 2 em 2.
seq(from = 1, by = 2, length.out = 7)
```

```
## [1]  1  3  5  7  9 11 13
```

Criando repetições estruturadas

Função `rep()` cria repetições estruturadas. Exemplo:

```r
# Repita o 0 5 vezes
rep(0, 5)
```

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

```r
# Repita a sequencia 1 a 3, 2 vezes
rep(1:3, times = 2)
```

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

```r
# Repita a sequencia 1 a 3, cada número 2 vezes
rep(1:3, each = 2)
```

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

Gerando valores aleatórios

Dado um vetor de elementos podemos usar a função sample() para obter uma amostra aleatória. Exemplo:

```r
# Retire uma amostra aleatório de tamanho 10 de uma sequencia de 1 a 20 sem reposição.
sample(1:20, size = 10, replace = FALSE)
```

```
##  [1]  7 16 14  2 20 10 11 18  9 19
```

```r
# Retire uma amostra de tamanho 10 da sequencia "a", "b", "c" de tamanho 10 com reposição.
sample(c("a", "b", "c"), size = 10, replace = TRUE)
```

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

Ou ainda, usando distribuições de probabilidade:

```r
# 10 amostras da distribuição uniforme entre 0 e 1
runif(n = 5, min = 0, max = 1)
```

```
## [1] 0.9723809 0.5558853 0.1753428 0.1181895 0.7706595
```

```r
# 10 amostras da distribuição Normal com média 1.80 e desvio-padrão 0.1
rnorm(n = 5, mean = 1.80, sd = 0.1)
```

```
## [1] 1.676658 1.674509 1.767732 1.954583 1.627998
```

Selecionando elementos de um vetor

Podemos selecionar os elementos de um vetor usando: alguma característica, posição ou nome. Exemplo:

```r
## Criando vetor numérico nomeado.
notas <- c("João" = 7.8,
"Bianca" = 10,
"Eduarda" = 8.5,
"Felipe" = 7.0,
"Márcia" = 6.5)

notas[1] # A posição 1.
```

```
## João
##  7.8
```

```r
notas[5] # A posição 5.
```

```
## Márcia
##    6.5
```

```r
notas[1:2] # Um intervalo.
```

```
##   João Bianca
##    7.8   10.0
```

```r
notas[c(1, 3)] # Um conjunto.
```

```
##    João Eduarda
##     7.8     8.5
```

```r
notas[-1] # Remove.
```

```
##  Bianca Eduarda  Felipe  Márcia
##    10.0     8.5     7.0     6.5
```

Seleção com máscara lógica

```r
# Alunos com nota maior que 7.0
mask <- notas > 7.0
mask
```

```
##    João  Bianca Eduarda  Felipe  Márcia
##    TRUE    TRUE    TRUE   FALSE   FALSE
```

```r
notas[mask]
```

```
##    João  Bianca Eduarda
##     7.8    10.0     8.5
```

```r
# Alunos com nota maior que 9.0
notas[notas > 9.0]
```

```
## Bianca
##     10
```

Modificando vetores

```r
# Atribui nota para um aluno.
notas["João"] <- 0
notas
```

```
##    João  Bianca Eduarda  Felipe  Márcia
##     0.0    10.0     8.5     7.0     6.5
```

```r
# Atribui nota "desconhecida" para aluno.
notas["Felipe"] <- NA
notas
```

```
##    João  Bianca Eduarda  Felipe  Márcia
##     0.0    10.0     8.5      NA     6.5
```

Removendo e adicionando componentes de um vetor

```r

# Remove elemento do vetor.
notas <- notas[-4]
notas
```

Removendo

```
##    João  Bianca Eduarda  Márcia
##     0.0    10.0     8.5     6.5
```

```
append(notas, value = c("Carlos" = 9.0))
```

Adicionando

```
##    João  Bianca Eduarda  Márcia  Carlos
##     0.0    10.0     8.5     6.5     9.0
```

```
append(notas, value = c("Simone" = 7.2),
       after = 0)
```

```
##  Simone    João  Bianca Eduarda  Márcia
##     7.2     0.0    10.0     8.5     6.5
```

```
novas_notas <- c(notas,
                 c("Pedro" = 8.0,
                   "Luana" = 8.3))
novas_notas
```

```
##    João  Bianca Eduarda  Márcia   Pedro   Luana
##     0.0    10.0     8.5     6.5     8.0     8.3
```

Matrizes, listas e tabelas

Matrizes

Matrizes são um simples extensão dos vetores. São vetores com dimensão (número de linhas e colunas). Todos os elementos de uma matriz devem ser do mesmo tipo. Definindo uma matriz:

```
matriz <- matrix(data = c(1, 0, 0, 1),
                 nrow = 2, ncol = 2)
matriz
```

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

```
dim(matriz)
```

Acessando a dimensão de uma matriz

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

Matrizes são por default preenchidas por colunas.

```r
matriz <- matrix(c(1,2,3,4,5,6,7,8),
                 nrow = 4, ncol = 2)
matriz
```

```
##      [,1] [,2]
## [1,]    1    5
## [2,]    2    6
## [3,]    3    7
## [4,]    4    8
```

Mas pode-se preencher por linhas

```r
matriz <- matrix(c(1,2,3,4,5,6,7,8),
                 nrow = 4, ncol = 2,
                 byrow = TRUE)
matriz
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
## [3,]    5    6
## [4,]    7    8
```

Acessando elementos de uma matriz Acesso por posição (linha e coluna).

```r
# Linha 1 Coluna 2
matriz[1,2]
```

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

```r
# Linha 3 Coluna 1
matriz[3,1]
```

```
## [1] 5
```

```r
# Linhas 1 e 2 Coluna 2
matriz[1:2, 2]
```

```
## [1] 2 4
```

```r
# Linhas 2 e 3 e Colunas 1 e 2
matriz[2:3, 1:2]
```

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

```r
## Elementos maiores do que 4
matriz[matriz > 4]
```

Seleção condicional por máscara

```
## [1] 5 7 6 8
```

```r
## Máscara lógica
matriz > 4
```

```
##       [,1]  [,2]
## [1,] FALSE FALSE
## [2,] FALSE FALSE
## [3,]  TRUE  TRUE
## [4,]  TRUE  TRUE
```

Listas

A estrutura `list` é a mais exível em R. Pode armazenar diferentes tipos de objetos.

```r
minha_lista <- list(10, "Dez", TRUE, 1+10i)
minha_lista
```

```
## [[1]]
## [1] 10
##
## [[2]]
## [1] "Dez"
##
## [[3]]
## [1] TRUE
##
## [[4]]
## [1] 1+10i
```

Acessando elementos de uma lista.

```r
# Primeiro elemento
minha_lista[[1]]
```

```
## [1] 10
```

```r
# Quarto elemento
minha_lista[[4]]
```

```
## [1] 1+10i
```

```r
# Primeiro e quarto elemento
minha_lista[c(1,4)]
```

```
## [[1]]
## [1] 10
##
## [[2]]
## [1] 1+10i
```

Listas nomeadas Os elementos da lista podem ser nomeados.

```r
minha_lista <-  list("Números" = c(10, 100),
                     "Caracter" = c("Dez", "Cem",
                                    "Logico" = TRUE,
                                    "Complexo" = 1+10i))

minha_lista
```

```
## $Números
## [1]  10 100
##
## $Caracter
##                      Logico Complexo
##    "Dez"    "Cem"   "TRUE"  "1+10i"
```

Acesso pelo nome

```r
minha_lista$Logico
```

```
## NULL
```

```r
minha_lista$Números
```

```
## [1]  10 100
```

Acessando os atributos de uma lista

```r
attributes(minha_lista)
```

```
## $names
## [1] "Números"  "Caracter"
```

Tabelas

- Classe mais utilizada para representar conjuntos de dados.
- Similar a uma planilha eletrônica.
- Um `data.frame` tem diversos atributos: `rownames()`, `colnames()`, `names()` e `dim()`.
- É aconselhável usar a convenção: cada linha representa uma observação (registro) e cada coluna uma variável.

- R permite ler conjuntos de dados de diversos formatos.
- O R já vem equipado com conjuntos de dados para ns didáticos.
- Conjunto de dados iris.

```
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

Classe factor A classe factor é similar a character no entanto permite denir ordem. Por exemplo:

```
fator <- factor(c("alta","baixa","baixa","media",
                  "alta","media","baixa","media","media"),
                levels = c("baixa", "media", "alta"),
                ordered = TRUE)
```

Se zermos uma tabela, por exemplo, a ordem será respeitada

```
table(fator)
```

```
## fator
## baixa media  alta
##     3     4     2
```

Criando e inspecionando um `data.frame` Podemos criar um `data.frame` diretamente em R.

```
dados <- data.frame(Letras = letters[1:6],
                    Numeros = 1:6,
                    Logico = rep(c(TRUE, FALSE),
                                 each = 3))
dados
```

```
##   Letras Numeros Logico
## 1      a       1   TRUE
## 2      b       2   TRUE
## 3      c       3   TRUE
## 4      d       4  FALSE
## 5      e       5  FALSE
## 6      f       6  FALSE
```

Diversas opções de inspeção:

- `head()` - mostra as seis primeiras linhas.
- `tail()` - mostra as seis últimas linhas.
- `dim()` - número de linhas e de colunas.
- `nrow()` - número de linhas.
- `ncol()` - número de colunas.
- `str()` - estrutura do `data.frame`.
- `names()`, `colnames()` e `rownames()` nome das linhas e colunas.

Acessando elementos de um `data.frame` O acesso é parecido com objetos da classe `matrix`.

```
# Primeira linha todas as colunas
iris[1,]
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
```

```
# Segunda linha colunas de 1 a 3
iris[2,1:3]
```

```
##   Sepal.Length Sepal.Width Petal.Length
## 2          4.9           3          1.4
```

```
# Linhas 1 e 5 todas as colunas
iris[c(1, 5),]
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
```

```
# Acesso pelo nome da coluna
iris$Sepal.Length
```

```
##   [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1
##  [19] 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0
##  [37] 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0 6.4 6.9 5.5
##  [55] 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1
##  [73] 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6 5.5
##  [91] 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3
## [109] 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0 6.9 5.6 7.7 6.3 6.7 7.2
## [127] 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8
## [145] 6.7 6.7 6.3 6.5 6.2 5.9
```

Resumo

- Os principais tipos de dados são: `character`, `numeric`, `integer`, `logical`, `complex` e `raw`.
- As principais estruturas de dados são: `vector`, `matrix`, `list` e `data.frame`.
- Lembre-se:
  - Vetores: conjunto de elementos unidimensional, todos do mesmo tipo.
  - Matrizes: conjunto de elementos bidimensional (linha e coluna), todos do mesmo tipo.
  - Listas: caso especial de vetor em que cada elemento pode ser uma estrutura diferente.
  - Data frames: tabela. Cada coluna é um vetor, portanto os elementos dentro da coluna são de mesmo tipo mas os tipos entre colunas podem ser diferentes.