R para epidemiologia

Rodrigo Citton P. dos Reis
rodrigocpdosreis@gmail.com

Universidade Federal do Rio Grande do Sul
DEst, PPGEPI, ELSA-Brasil

Porto Alegre, 2018

Conceitos básicos do R

Objetos e funções

- Para entender como o R funciona, você precisa saber que o R gira em torno de "duas coisas": **objetos** e **funções**.
  - Quase tudo no R ou é um objeto ou uma função.
- **O que é um objeto?** Um objeto é uma "coisa" - como um número, um conjunto de dados, uma estatística resumo (uma média ou desvio padrão), ou um teste estatístico.
- Objetos vêm em muitas formas e tamanhos diferentes no R.
  - Há objetos simples, como **escalares** que representam números individuais, **vetores** que representam uma lista de números, objetos mais complexos como **dataframes** que representam tabelas de dados e objetos ainda mais complexos, como **testes de hipóteses** ou **regressão** que contêm todos os tipos de informação estatística.

Objetos e funções

- Diferentes tipos de objetos têm **atributos** diferentes.
  - Por exemplo, um vetor de dados tem um **atributo de comprimento** (ou seja, quantos números estão no vetor), enquanto que o teste de hipótese tem muitos atributos, tais como um **teste estatístico** e um **valor de p**.
  - Não se preocupe se isso é um pouco confuso agora - tudo ficará mais claro logo a seguir. **(Por enquanto, só é preciso saber que objetos em 'R' são "coisas", e diferentes objetos têm atributos diferentes.)**
- **O que é uma função?** Uma função é um **procedimento** que normalmente tem um ou mais objetos como argumentos **(inputs)**, faz alguma "coisa" com esses objetos, e retorna um novo objeto.
  - Por exemplo, a função `mean()` toma um vetor de dados numéricos como um argumento, calcula a média aritmética desses dados e retorna um número único (um escalar) como resultado **(output)**.
  - No R você pode facilmente **criar suas próprias funções** para fazer o que quiser **(veremos isto posteriormente no curso)**.

Objetos e funções

- Felizmente, o R tem centenas (milhares?) de funções próprias que realizam a maioria das tarefas de análise básicas que você pode pensar.

- Vamos ver uma função na prática.
  - No código abaixo, vamos definir um objeto vetor chamado **idades**.
  - Vamos utilizar a função `mean()` para calcular a idade média do vetor idades, definindo o argumento da função como o vetor idades.

```
# 1. Crie o objeto vetor chamado idades
idades <- c(32, 29, 60, 59, 22, 27, 98)

# 2. Aplique a função mean() no objeto idades
mean(idades)
```

```
## [1] 46.71429
```

Objetos e funções

- Como você pode ver, o resultado da função `mean()`, com o objeto vetor idades como input, retornou um número único (um objeto escalar) com um valor de 46.71429.
- Diferentes funções requerem diferentes tipos de objetos como inputs.
  - Por exemplo, a função `mean()` tem de ter um objeto numérico (tipicamente um vector) como um input.
  - Se você tentar usar um objeto não-numérico como um argumento para `mean()`, você obterá um erro porque a função não foi concebida para trabalhar com esse tipo de argumento. **(NÃO SE ASSUSTE COM ERROS NO R! É APENAS A MANEIRA DELE SE COMUNICAR COM O USUÁRIO!)**
- Conforme você aprende R, você aprenderá novos tipos de objetos (e seus atributos) e novas funções.
  - Para ver uma lista das funções mais comumente usadas em R, veja o **R Card**

Objetos e funções

Objetos e funções

- Esqueça o **R Card**!!!
- Veja o **R Reference Card:**

https://cran.r-project.org/doc/contrib/Short-refcard.pdf.

(Imprima o cartão e mantenha-o por perto quando você estiver aprendendo novas funções.)

O R como calculadora

O R pode ser usado como uma calculadora:

```
1 + 2
```

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

```
3 - 4
```

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

```
5 * 6
```

```
## [1] 30
```

O R como calculadora

```
7/8
```

```
## [1] 0.875
```

```
2^3
```

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

O R como calculadora

SUA VEZ!

1. Calcule as seguintes expressões no R:
   - **1.1** $1 + 1$.
   - **1.2** 10×1.
   - **1.3** $3 - 0$.
   - **1.4** $\left(\frac{10+5}{5}\right)^2$ (veja como o R trabalha com os `()`; experimente "diferentes" formas de escrever esta expressão).
2. Uitilize as funções de ajuda (`help()`, `help.search()`, `apropos()`) para calcular as seguintes expressões no R:
   - **2.1** $\log(1)$.
   - **2.2** $\sqrt{3}$.
   - **2.3** $\sqrt{3^2 + 4^2}$.

Escrevendo scripts: um breve guia de estilo

- Como já vimos anteriormente, a melhor forma de se trabalhar no R é através de **scripts**.
- Um script pode ser visto como uma coleção de funções aplicadas para realizar uma determinada tarefa.
- Para que nosso trabalho seja eficiente e possa ser reaproveitado é necessário um pouco de organização.

Escrevendo scripts: um breve guia de estilo

- Como todas as linguagens de programação, o R não é apenas feito para ser lido por um computador, também é feito para ser lido por outros seres humanos.
- Por esta razão, é importante que o código parece agradável e seja compreensível para outras pessoas e o você mesmo no futuro.
- Não iremos fornecer um guia de estilo completo - para tal veja o **guia de estilo R do Google**: https://google.github.io/styleguide/Rguide.xml.
- No entanto iremos ver os dois aspectos mais críticos do bom estilo: **comentando** e **espaçamento**.

Comente seu código com

- Os comentários são completamente ignorados pelo R e aparecem no código apenas para quem está lendo o mesmo.
- Você pode usar os comentários para explicar o que uma determinada linha de código está fazendo, ou apenas para separar visualmente pedaços significativos de código uns dos outros.
- Outra razão para se usar comentários é impedir que um pedaço de código seja executado no console.
- Comentários em R são designados por um sinal #.
  - Sempre que o R encontra um sinal #, ele irá ignorar todo o código após o sinal # nessa linha.
  - Além disso, na maioria dos editores de código (como RStudio), o editor irá exibir comentários em uma cor diferente do que o código em R padrão para lembrá-lo que é um comentário.

Comente seu código com

Exemplo:

Comente seu código com

```r
# Belo Horizonte, 4 de agosto de 2016
# Aula de R para epidemiologia
# Autor: Rodrigo

# Instalando pacotes do R
# install.packages(Epi)

# Carregando pacotes do R
library(Epi)

# Carregando os dados de BIRTHS
data(births)

# O nome das colunas deste conjunto de dados
names(births)

# Qual a média da variável bweight
mean(births$bweight)

# Quantos meninos nasceram com baixo peso?
sum(births$lowbw[births$sex == 1])
```

Espaços

Comovocêgostariadelerumlivrosenãohouvesseespaçosentreaspalavras? Euestousupondoquevocênãofariaisso. Assimcadavezquevocêescrever-códigosemespaçamentoadequadolembresedestafrase.

- Comentários não são única forma de tornar o seu código legível.
- É importante fazer uso apropriado de espaços e linhas em branco.
  - Por exemplo, inclua espaços entre operadores aritméticos (=, + e −) e após vírgulas.

Exemplo:

```
a<-(100+3)-2
mean(c(a/100,642564624.34))
t.test(formula=bweight~sex,data=births)
plot(x=births$gestwks,y=births$bweight,main="myplot")
```

Espaços

- A falta de espaço dificulta a leitura do código. Compare o mesmo código com o uso adequado do espaçamento:

```
# Alguns cálculos sem significado

a <- (100 + 3) - 2
mean(c(a / 100, 642564624.34))

# Teste t para comparar o peso
# ao nascer de meninos e meninas

t.test(formula = bweight ~ sex,
       data = births)

# Gráfico de dispersão de tempo
# de gestação (em semanas) e peso ao nascer
plot(x = births$gestwks,
     y = births$bweight,
     main = "myplot")
```

Criando novos objetos no R

- Até agora utilizamos o R para realizar simples cálculos.
- Para realmente tirarmos proveito do R precisamos saber como criar e manipular **objetos**.
- Todo dado, análise, e até mesmo gráfico, são ou podem ser salvos em objetos do R.
  - Por exemplo, o objeto `births` que utilizamos anteriormente é um objeto armazenado no pacote `Epi`.
  - Quando carregamos o pacote `Epi`, avisamos ao R nos dar acesso ao objeto `births`.
  - Uma vez que o objeto foi carregado, podemos calcular estatísticas descritivas, testes de hipóteses e criar gráficos.

Criando novos objetos no R

- Para criar um novo objeto no R, você precisará fazer **atribuição de objeto**.
  - A atribuição de objetos nos permite armazenar objetos de dados com nomes relevantes para posterior análise.
- Para fazer uma atribuição usamos o operador <- **(podemos chamá-lo de operador "atribuir")**.
- Para atribuir algo a um novo objeto **(ou atualizar um objeto existente)** use a notação `objeto <- []`, em que `objeto` é um novo **(ou atualizado)** objeto e `[]` é o que você quiser armazenar no `objeto`.
- Vamos começar criando um objeto muito simples chamado `a` e atribuindo o valor 535 a este:

```
a <- 535
```

Criando novos objetos no R

- Note que uma vez executado o código acima, o R não mostra nada.
- Se você quiser ver o valor armazenado no objeto a você terá que chamá-lo executando seu nome.

```
a
```

```
## [1] 535
```

- Se você tentar avaliar um objeto ainda não definido, o R retornará um erro:

```
b
```

```
## Error in eval(expr, envir, enclos): objeto 'b' não encontrado
```

Criando novos objetos no R

- Uma vez que você definiu um objeto você pode combiná-lo com outros objetos usando aritmética simples.

```
b <- 500

# O que é a + b?
a + b
```

```
## [1] 1035
```

```
# Atribua a + b a um novo objeto (c)
c <- a + b

# O que é c?
c
```

```
## [1] 1035
```

Criando novos objetos no R

```r
# Atualize o objeto a
a <- 11
a
```

```
## [1] 11
```

Para alterar um objeto, você deve atribuí-lo novamente

- Suponha que o objeto z armazene o valor 0.
- Gostaríamos de adicionar 1 a z para torná-lo 1.
- Isso é o que acontece se você **não atribuir** novamente:

```
z <- 0
z + 1
```

- Agora veja o valor de z

```
z
```

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

Para alterar um objeto, você deve atribuí-lo novamente

- z ainda é 0!
- Ao escrevermos z + 1 na segunda linha, o R pensou que queríamos apenas fazer um cálculo e imprimir este resultado, sem armazenar o valor em um novo objeto z.
- Se quisermos atualizar o valor de z devemos reatribuir o resultado a z:

```
z <- 0
z <- z + 1 # Agora de fato estamos mudando z
z
```

Como nomear objetos

- Você pode utilizar qualquer combinação de letras e alguns caracteres **(como . ou _)** especiais para criar nomes de objetos.
- Tente manter os nomes de objetos curtos e fáceis de lembrar!

```
# Nomes válidos
media.grupo <- 10.21
minha.idade <- 32
TimeFavorito <- "Grêmio"
soma.de.1.a.5 <- 1 + 2 + 3 + 4 + 5


# Nomes inválidos: possuem espaços ou começam com números
5experimento <- 50
5! <- 50
Pontuacao do Gremio <- 50
```

O R é case-sensitive

- Isto significa que o R trata de maneira diferente letras maiúsculas e minúsculas.

```
# Estes são objetos diferentes
Peso <- 78
peso <- 68
PESO <- 98
```

Sua vez!

1. Crie um novo script R. Usando comentários, escreva seu nome, data e "Sua vez do Conceitos básicos do R" no topo do script. Escreva a resposta dos próximos exercícios no script (copia e cole o enunciado usando comentários). Seu script deve conter apenas código R válido e comentários.
2. Qual deste (se houver algum) dos seguintes nomes de objetos são inválidos?

```
esteaqui <- 1
ESTEAQUI <- 2
este.aqui <- 3
Este.1 <- 4
EsTE.......AQU....I <- 5
Este!Aqui! <- 6
ldkflkdflsk <- 7
```

Sua vez!

3. Em 2016, a cidade de Belo Horizonte registrou até o mês de junho 42 casos de zika. Crie um objeto chamado zika.bh.2016 e atribua o correspondente valor a este.
4. Atualização: os dados de julho chegaram, e infelizmente mais 7 casos de zika foram registrados em Belo Horizonte. Acrescente estes casos no objeto zika.bh.2016.
5. Veja o código a seguir. O que o R retornará após a terceira linha? Faça uma predição e depois teste você mesmo.

```
a <- 10
a + 10
a
```

6. Crie um objeto com o seu peso; crie um segundo objeto com sua altura; e por fim, um objeto com seu IMC utilizando os dois objetos criados anteriormente para o cálculo.

Escalares e vetores

Escalares

- O objeto mais simples no R é um **escalar**.
- Um escalar é apenas um valor como um número ou um nome.

```
a <- 100
b <- 3 / 100
c <- (a + b) / c
```

- Escalares não precisam ser **numéricos**, eles também podem **caracteres** (strings).
  - No R denotamos caracteres usando aspas.

```
d <- "Epidemiologia"
e <- "Saúde Pública"
f <- "Campus Pampulha fechado por causa dos Olimpíadas!"
```

Escalares

- Como você pode imaginar, o R **trata escalares numéricos e caracteres diferentemente**:

```
a.num <- 1
b.num <- 2
a.num + b.num
```

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

```
a.cha <- "1"
b.cha <- "2"
a.cha + b.cha
```

```
## Error in a.cha + b.cha: argumento não-numérico para operador binário
```

```
# Se o conteúdo do objeto caracter for um número,
# podemos torná-lo um objeto numérico:
as.numeric(a.cha)
```

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

Vetores

- Um objeto **vetor** é apenas uma combinação de vários escalares armazenados em um único objeto.
- A forma mais simples de se criar um vetor no `R` é utilizando a função `c()` (concatenar).

`c(a, b, c, ...)`

`a, b, c, ...`
Um ou mais objetos a serem combinados em um vetor.

- Por exemplo, os números de 1 a 10 podem ser um vetor de comprimento 10.

Vetores

```
a <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
a
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

- Como os escalares, vetores podem ser numéricos ou caracteres **(mas não os dois!)**.

```
peso <- c(70, 97, 78)
peso
```

```
## [1] 70 97 78
```

Vetores

```
sexo <- c("Fem", "Masc", "Masc")
sexo
```

```
## [1] "Fem"  "Masc" "Masc"
```

- Posteriormente veremos outras **classes de objetos**.
- Também podemos combinar dois ou mais vetores:

```
a <- c(1, 2, 3, 4, 5)
b <- c(6, 7, 8, 9, 10)
c <- c(a, b)
c
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

Vetores

- **Coerção:** veja o que acontece se você tentar combinar escalares numéricos e caracteres.

```
vet.mescla <- c("R", 1, "para", 2, "Epidemiologia", 3)
vet.mescla
```

```
## [1] "R"             "1"             "para"          "2"
## [5] "Epidemiologia" "3"
```

Funções para gerar vetores numéricos

- Imagine que você precisa criar um vetor de identificação **(id)** de pacientes de 1 até 100.
  - Você pode utilizar a função `c()`, mas provavelmente levaria muito tempo. (Por que?)
- No entanto, o `R` já possui funções para gerar vetores numéricos: `a:b`, `seq()`, `rep()`.

Funções para gerar vetores numéricos

a:b

a
: O começo da sequência.

b
: O fim da sequência.

```
id <- 1:100
# id
10:1
```

```
##  [1] 10  9  8  7  6  5  4  3  2  1
```

```
2.5:8.5
```

```
## [1] 2.5 3.5 4.5 5.5 6.5 7.5 8.5
```

Funções para gerar vetores numéricos

`seq(from, to, by)`

`from`
: O começo da sequência.

`to`
: O fim da sequência.

`by`
: O tamanho do passo da sequência.

`length.out`
: O comprimento final da sequência (só usar se `by` não for especificado).

```
id <- seq(from = 1, to = 100, by = 1)
# id
seq(from = 0, to = 100, by = 10)
```

Funções para gerar vetores numéricos

```
##  [1]   0  10  20  30  40  50  60  70  80  90 100
```

```
seq(from = 0, to = 100, length.out = 5)
```

```
## [1]   0  25  50  75 100
```

Funções para gerar vetores numéricos

rep(x, times, each)

`x`
: Um escalar ou um vetor de valores a serem repetidos.

`times`
: O número de vezes que a sequência deve ser repetida.

`each`
: O número de vezes que cada elemento da sequência deve ser repetido.

```
rep(x = 3, times = 10)
```

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

```
rep(x = c(1, 2), times = 5)
```

Funções para gerar vetores numéricos

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

```
rep(x = c("fumante", "não-fumante"), each = 2)
```

```
## [1] "fumante"     "fumante"     "não-fumante" "não-fumante"
```

```
rep(x = 1:5, times = 3)
```

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

Sua vez!

1. Consulte o help das funções `a:b`, `seq()`, `rep()`. Experimente executar os exemplos do help.
2. Crie o vetor [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] de três formas: usando as funções `c()`, `a:b` e `seq()`.
3. Crie o vetor [2.1, 4.1, 6.1, 8.1] de duas formas: usando as funções `c()` e `seq()`.
4. Crie o vetor [0, 5, 10, 15] de três formas: usando `c()`, `seq()` com o argumento `by` e `seq()` com o argumento `length.out`. **(Na dúvida consulte o help)**
5. Crie o vetor [101, 102, 103, 200, 205, 210, 1000, 1100, 1200] usando uma combinação das funções `c()` e `seq()`. **(Neste exercício você poderá ver o poder reciclador do R: o output de hoje é o input de amanhã!)**

Sua vez!

6. Um estudo clínico estabeleceu que de 100 pacientes, 10 irão receber o placebo, 40 receberão o tratamento padrão e 50 pacientes receberão o novo tratamento. Crie um vetor com 1 placebo, 4 padrão e 5 novo a cada grupo de 10.

Funções de vetores

length()

- Uma vez que temos um vetor de dados, gostaríamos de saber o seu comprimento.
- Não é uma boa ideia imprimir os valores do vetor e contar o número de elementos.
- Ao invés disso, usaremos a função `length()`.

```
a <- 1:10
length(a)
```

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

```
b <- seq(from = 1, to = 100, length.out = 20)
length(b)
```

```
## [1] 20
```

length()

```
length(c("R", "para", "Epidemiologia"))
```

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

```
length(c("R para Epidemiologia"))
```

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

Operações aritméticas em vetores

- Operações com vetores são realizadas **elemento a elemento**.
- Por exemplo, suponha que temos dois vetores a e b:

```
a <- c(1, 3, 5, 7)
b <- c(1, 2, 4, 8)
```

- Então se multiplicarmos a por 5, teremos um vetor com **cada um dos elementos** de a multiplicado por 5:

```
5 * a
```

```
## [1]  5 15 25 35
```

Operações aritméticas em vetores

- E se adicionarmos a e b, a soma será um novo vetor, no qual os membros serão a soma dos correspondentes elementos de a e b:

```
a + b
```

```
## [1]  2  5  9 15
```

- De forma análoga para a subtração, multiplicação e divisão:

```
a - b
```

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

```
a * b
```

Operações aritméticas em vetores

```
## [1]  1  6 20 56
```

```
a / b
```

```
## [1] 1.000 1.500 1.250 0.875
```

- Atenção: vetores de comprimentos diferentes

```
u <- c(10,20,30)
v <- 1:9
u + v
```

```
## [1] 11 22 33 14 25 36 17 28 39
```

Sua vez!

1. Transforme o vetor de proporções [0.01, 0.2, 0.13, 0.47, 0.08] em um vetor de percentuais.
2. Crie o vetor com os seguintes valores de cintura (em cm) [102, 92, 104, 90, 97, 85, 100, 114, 83, 99]. Crie o vetor com os seguintes valores de quadril (em cm) [119, 104, 123, 102, 105, 109, 117, 102, 99, 112]. Crie um novo vetor com a relação cintura/quadril.

Funções de vetores

- Agora que já sabemos criar vetores, vamos aprender as funções básicas de estatísticas descritivas.
- Começaremos vendo as funções que se aplicam a vetores numéricos **(variáveis contínuas ou discretas)**.
  - Sua vez: dê exemplos de variáveis contínuas e discretas.

Funções de vetores

Funções estatísticas para vetores numéricos

`sum(x)`, `product(x)`
: A soma e o produto dos elementos do vetor x.

`min(x)`, `max(x)`
: Os valores mínimo e máximo de um vetor x.

`mean(x)`
: A média aritmética de um vetor x.

`median(x)`
: A mediana de um vetor x.

`sd(x)`, `var(x)`
: O desvio padrão e a variância de um vetor x.

`quantile(x, p)`
: p-ésimo quantil de um vetor x.

`summary(x)`
: Apresenta algumas estatísticas resumo de um vetor x.

Funções de vetores

- Vamos calcular algumas estatísticas descritivas para a variável colesterol total.

```
colesterol <- c(221.7, 195.0, 235.8, 234.4, 246.9,
                212.7, 204.2, 173.9, 227.9, 237.3)

min(colesterol)
```

```
## [1] 173.9
```

```
max(colesterol)
```

```
## [1] 246.9
```

```
mean(colesterol)
```

```
## [1] 218.98
```

Funções de vetores

```r
median(colesterol)
```

```
## [1] 224.8
```

```r
quantile(colesterol, p = 0.5) # Quem é este quantil?
```

```
##   50%
## 224.8
```

```r
quantile(colesterol, p = c(0.25, 0.5, 0.75))
```

```
##     25%     50%     75%
## 206.325 224.800 235.450
```

```r
summary(colesterol)
```

Funções de vetores

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   173.9   206.3   224.8   219.0   235.4   246.9
```

```
sd(colesterol)
```

```
## [1] 22.5781
```

Atenção com valores ausentes (NA)!

- No `R` dados ausentes são codificados como `NA`.
- Em conjuntos de dados reais, valores `NA` aparecem a toda hora.
- Existem algumas formas de "tratarmos" os valores ausentes em uma análise estatística.
- Talvez a forma mais comum de tratarmos valores ausentes seja descartando-os e analisando apenas os dados completos.
- No `R`, precisamos deixar claro a nossa opção por descartarmos os dados ausentes.
  - Para as funções que acabos de ver, isto se dá através do **argumento** `na.rm`.
  - Por **default** o `R` especifica este argumento como `FALSE`.
  - Quando este argumento é especificado como `TRUE`, os valores `NA` são removidos do vetor, e a função é computada para os valores restantes.

Atenção com valores ausentes (NA)!

```r
hdl <- c(68.1, 30.5, NA, 31.2, NA, 78.8, NA, 51.2, 62.2, 57.6)

mean(hdl)
```

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

```r
mean(hdl, na.rm = TRUE)
```

```
## [1] 54.22857
```

```r
# T como um "apelido" para TRUE
mean(hdl, na.rm = T)
```

```
## [1] 54.22857
```

```r
# F como um "apelido" para FALSE
sd(hdl, na.rm = F)
```

Atenção com valores ausentes (NA)!

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

```
# Veja a informação adicional que aparece
# quando utilizamos a função summary()
# em um vetor com valores NA
summary(hdl)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##   30.50   41.20   57.60   54.23   65.15   78.80       3
```

Funções para dados discretos e não-numéricos

- Sua vez: Quais as estatísticas resumo são mais apropriadas para **variáveis discretas** ou **categóricas**?

Funções de contagem para dados discretos

`unique(x)`

Retorna um vetor com todos os valores únicos de um vetor x.

`table(x)`

Retorna uma tabela com os valores únicos de um vetor x. assim como as contagens de cada ocorrência. Por default, a função `table()` NÃO conta valores `NA`. Para incluir a contagem de valores `NA`, inclua o argumento `exclude = NULL`.

Funções para dados discretos e não-numéricos

```
gest.anteriores <- c(1, 0, 2, 1, 4, 1, 0, 3, 2, 1)
sexo <- c("M", "M", "F", "F", "F", "M", "F", "M", "F", "F")
```

```
unique(gest.anteriores)
```

```
## [1] 1 0 2 4 3
```

```
unique(sexo)
```

```
## [1] "M" "F"
```

```
table(gest.anteriores)
```

Funções para dados discretos e não-numéricos

```
## gest.anteriores
## 0 1 2 3 4
## 2 4 2 1 1
```

```
table(sexo)
```

```
## sexo
## F M
## 6 4
```

- Se quisermos obter as **proporções** no lugar das contagens, podemos dividir o resultado da função `table()` pela soma dos resultados **(O R reciclando novamente)**:

```
table(gest.anteriores)/sum(table(gest.anteriores))
```

Funções para dados discretos e não-numéricos

```
## gest.anteriores
##   0   1   2   3   4
## 0.2 0.4 0.2 0.1 0.1
```

```r
table(sexo)/sum(table(sexo))
```

```
## sexo
##   F   M
## 0.6 0.4
```

- Sua vez: calcule os percentuais relativos as proporções calculadas acima.

Padronização (*z-score*)

- Uma tarefa muito comum na análise de dados é a **padronização de variáveis** (também conhecido como calcular o *z-score*).
- O propósito da padronização de uma variável é colocar esta variável em uma escala comum para que esta possa ser comparada com outras variáveis.
- Para padronizar uma variável, simplesmente substraímos seus valores pela média amostral e então didimos o resultado pelo desvio padrão amostral.

Padronização (*z-score*)

```r
notas <- c(7.5, 9.2, 6, 8.1, 8, 9.9, 8.3, 8.7, 8.5, 7.7)
mean(notas)
```

```
## [1] 8.19
```

```r
sd(notas)
```

```
## [1] 1.047165
```

```r
# Criando o vetor de notas padronizadas
notas.z <- (notas - mean(notas)) / sd(notas)
mean(notas.z)
```

```
## [1] 4.218847e-16
```

```r
sd(notas.z)
```

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

Sua vez!

1. Crie os vetores de peso ao nascer para crianças do sexo masculino [2597, 3009, 3040, NA, 3985, 3064, 3571, NA, 3678, 3187] e feminino [NA, 3092, 3323, 924, 3753, NA, 3316, 3404, 3007, 3919].
   - Calcule as principais estatísticas resumo para estes dois vetores.
   - Compare as médias destes dois vetores.
   - Obtenha os valores padronizados para cada um dos vetores. **(Consulte o help da função scale())**
   - Combine estes dois vetores em um único vetor e calcule as estatísticas resumo e o vetor de valores padronizados deste novo vetor.
2. Crie o vetor de valores para a variável "Baixo peso ao nascer" [NA, "Normal", "Normal", "Baixo-peso", "Normal", NA, "Normal", "Normal", "Normal", "Normal"]
   - Apresente as contagens de peso "Normal" e "Baixo-peso".
   - Apresente as proporções de peso "Normal" e "Baixo-peso". **(Consulte o help da função prop.table())**
   - Apresente os percentuais de peso "Normal" e "Baixo-peso".

Trabalhando com vetores

Indexação

- Até o presente momento, vimos como aplicar funções a vetores.
- No entanto, em muitas análises, será necessário acessar valores específicos de um vetor baseado nas suas **posições** ou em algum outro **critério**.
  - Por exemplo, podemos estar interessados em analisar valores em uma localização específica do vetor, como **os dez primeiros elementos**, ou que atendem algum critério, como **todos os valores maiores que 0**.
- Para acessar valores específicos de um vetor em R usaremos **indexação**.
- Exemplo (hipotético):

```
gest.sem <- c(38.52, 0, 38.15, 0, 0, 40.97, 42.14, 0, 42.03, 0)
sexo <- c("f", "m", "f", "m", "m", "f", "f", "m", "f", "m")
# Qual o tempo de gestação do primeiro indivíduo?
gest.sem[1]
```

```
## [1] 38.52
```

Indexação

```
# Qual o sexo do 3º, 4º e 5º indivíduos?
sexo[3:5]
```

```
## [1] "f" "m" "m"
```

```
# Quantos indivíduos do sexo feminino foram pesquisados?
sum(sexo == "f")
```

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

```
# Quais são os valores de gestação que são maiores que 0?
gest.sem[gest.sem > 0]
```

```
## [1] 38.52 38.15 40.97 42.14 42.03
```

```
# Quais são os tempos de gestação das mulheres?
gest.sem[sexo == "f"]
```

```
## [1] 38.52 38.15 40.97 42.14 42.03
```

Indexando vetores com colchetes

- Para indexar um vetor, você deve utilizar colchetes `[]` após o objeto vetor.

`a[]`

- Em geral, o que você colocar dentro dos colchetes, diz ao `R` quais os valores do objeto vetor você deseja acessar.
- Existem duas formas de usar indexação para **acessar subconjuntos** de dados em um vetor:
  - Indexação **numérica**
  - Indexação **lógica**

Indexação numérica

- Com indexação numérica, você deve entrar com **inteiros** correspondendo aos valores no vetor que você deseja acessar na forma `a[indice]`, em que `a` é o vetor, e `indice` é um vetor de valores índice.
  - Por exemplo, para obter o primeiro valor de um vetor chamado `a`, você deverá escrever `a[1]`.
  - Para obter o primeiro, segundo e terceiro valores do vetor `a`, você pode escrever `a[c(1,2,3)]` ou `a[1:3]`.
  - Note que podemos utilizar as funções que conhecemos anteriormente para gerar vetores de índices, no entanto, **estes precisam ser inteiros (positivos ou negativos)**.

```
a <- c(0, 50, 2, 39, 9, 20, 17, 8, 10, 100)
# 1º elemento
a[1]
```

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

Indexação numérica

```
# Elementos 1-5
a[1:5]
```

```
## [1]  0 50  2 39  9
```

```
# Todo segundo elemento
a[seq(from = 2, to = 10, by = 2)]
```

```
## [1]  50  39  20   8 100
```

```
# Passo-a-passo
ind <- seq(from = 2, to = 10, by = 2)
ind
```

```
## [1]  2  4  6  8 10
```

```
a[ind]
```

Indexação numérica

```
## [1]  50  39  20   8 100
```

```
# Todos os elementos com exceção do 4º
a[-4]
```

```
## [1]   0  50   2   9  20  17   8  10 100
```

```
# Todo o vetor sem os primeiros 4 elementos
a[-1:-4]
```

```
## [1]   9  20  17   8  10 100
```

Indexação lógica

- A segunda forma de indexar vetores no `R` é através de **vetores lógicos**.
- Um vetor lógico é um vetor que contém apenas valores `TRUE` ou `FALSE`.
- No `R`, valores verdadeiros são designados tanto com `TRUE` ou `T`, e valores falsos com `FALSE` ou `F`.
  - Note que os valores de um vetor lógico **NÃO levam aspas**, pois não são vetores caracteres, e sim lógicos.
  - **Esta é uma nova classe de objetos que estamos aprendendo**; até agora conhecemos as seguintes classes:
    - **numérico**/`numeric`
    - **caractere**/`character`
    - **lógico**/`logic`

Indexação lógica

- Você pode criar vetores lógicos a partir de vetores já existentes utilizando **operadores de comparação**:

Operadores	Descrição
`<`	Menor que
`<=`	Menor ou igual a
`>`	Maior que
`>=`	Maior ou igual a
`==`	Igual a
`!=`	Diferente de
`!x`	Não ‘x‘
`x | y`	‘x‘ OU ‘y‘
`x & y`	‘x‘ E ‘y‘

Indexação lógica

```
# Que valores são maiores que 40?
gest.sem > 40
```

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

```
# Que valores são iguais a 0?
gest.sem == 0
```

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

Indexação lógica

- Para vetores de caracteres, você pode utilizar os operadores == ou != **(outros operadores de comparação tais como < não fazem sentido para vetores de caracteres)**.

```
sexo == "m"
```

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

```
sexo == "f"
```

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

```
sexo != "m"
```

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

Indexação lógica

- Você também pode criar um vetor lógico comparando dois vetores de mesmo comprimento.
- Quando você faz isto, o R irá comparar valores na mesma posição **(lembre-se que as operações com vetores são realizadas elemento a elemento)** .

```
pas.antes  <- c(143, 122, 113, 126, 126, 135, 135, 151, 130, 136)
pas.depois <- c(127, 129, 107, 107, 103, 135, 140, 145, 110, 115)
# O tratamento fez efeito?
pas.depois < pas.antes
```

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

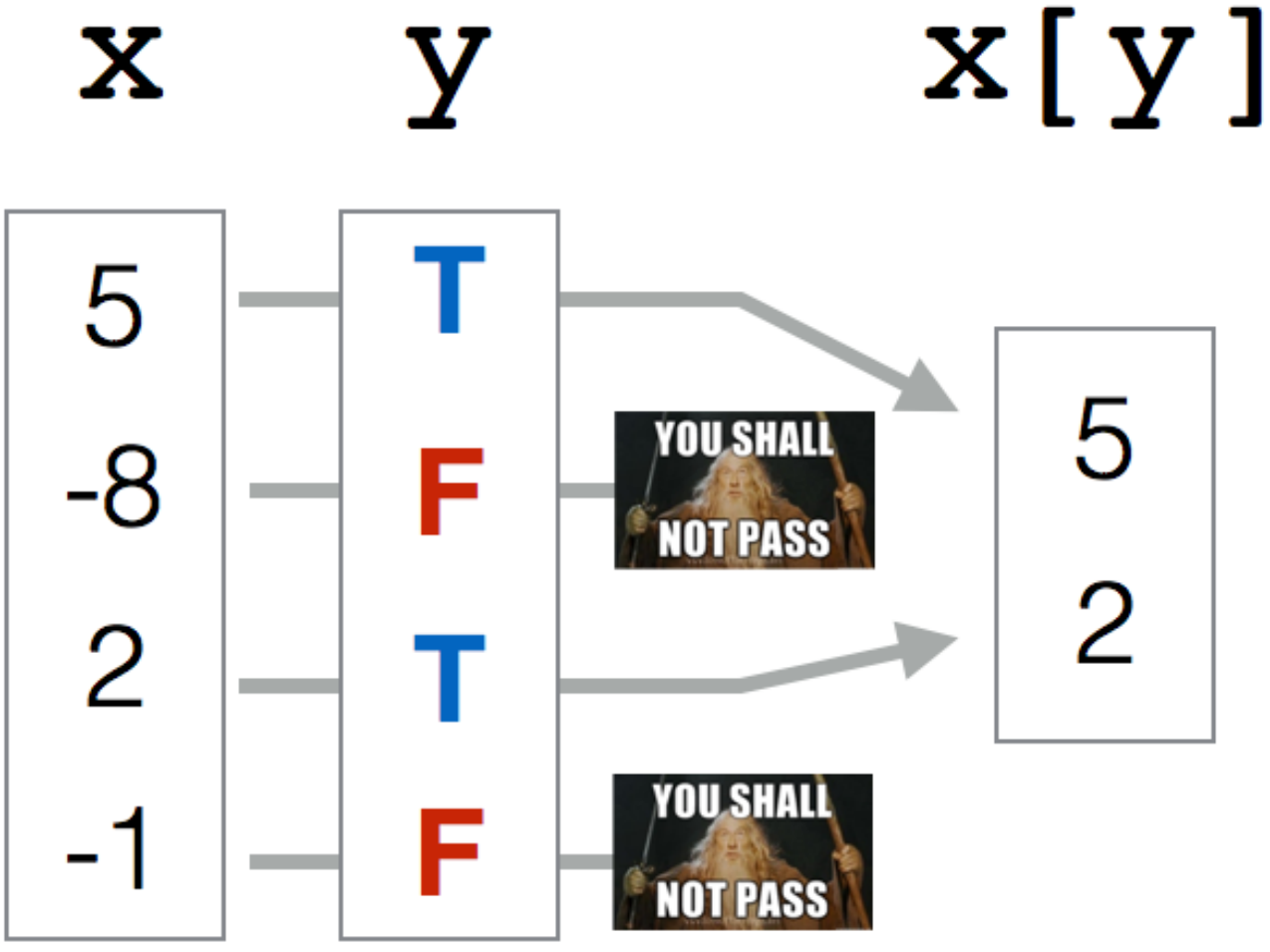
```
# PAS foi diferente após tratamento?
pas.antes != pas.depois
```

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

Indexação lógica

- Uma vez criado o vetor lógico, você pode utilizá-lo para indexar outro vetor.
- Quando você faz isto, o `R` retorna os valores do vetor para todos valores `TRUE` e irá ignorar todos os valores para os quais o vetor lógico é `FALSE`.
  - Assim, o vetor lógico, combinado com os colchetes `[]`, age como um **filtro** para o vetor que este está indexando.

Indexação lógica



Indexação lógica

```
# Quais são os tempos de gestação das mulheres?
gest.sem[sexo == "f"]
```

```
## [1] 38.52 38.15 40.97 42.14 42.03
```

```
# Quais os valores de pressão daqueles que
# possuem pressão alta?
pas.antes[pas.antes > 130]
```

```
## [1] 143 135 135 151 136
```

Usando & (E), | (OU)

- Utilizando esta noção de filtro para a indexação lógica, podemos utilizar mais de uma variável de filtro:

```
# Quais são os tempos de gestação dos indivíduos
# do sexo feminino E com pressão acima de 130?
gest.sem[sexo == "f" & pas.antes > 130]
```

```
## [1] 38.52 40.97 42.14
```

```
# Quais são os tempos de gestação dos indivíduos
# do sexo feminino OU com pressão acima de 130?
gest.sem[sexo == "f" | pas.antes > 130]
```

```
## [1] 38.52 38.15 40.97 42.14  0.00 42.03  0.00
```

Alterando valores em um vetor com indexação

- Também podemos alterar os valores de um vetor utilizando indexação lógica.
- Isto pode ser bastante útil quando desejamos remover valores inválidos de um vetor antes de realizar uma análise estatística.

```
# O valor 999 representa um valor ausente
# neste vetor de pesos
peso <- c(70, 97, 78, 57, 48, 999, 103, 83, 999)
peso[peso == 999] <- NA
mean(peso, na.rm = T)
```

```
## [1] 76.57143
```

Sua vez!

1. Crie os seguintes vetores no R:
   - **id** com valores de 1 até 10.
   - **idade**: [47, 55, 58, 80, 52, 40, 30, 74, 53, 57]
   - **sexo**: [“f”, “f”, “m”, “m”, “f”, “f”, “f”, “f”, “m”, “f”]
   - **peso**: [74.49, 40.91, 83.65, 79.30, 48.44, 53.85, 61.28, 59.66, 87.37, 82.33]
   - **altura**: [1.59, 1.77, 1.78, 1.70, 1.61, 1.72, 1.86, 1.45, 1.57, 1.64]
   - **cintura**: [82.44, 47.86, 102.94, 108.51, 71.97, 75.58, 88.11, 90.34, 111.10, 99.84]
   - **quadril**: [67.87, 70.63, 124.63, 104.13, 98.12, 80.17, 101.22, 104.65, 113.71, 88.10]
2. Apresente as estatísticas resumo para todos os vetores acima **(lembre de quais são mais adequadas para cada tipo de variável)**.
3. Apresente as estatísticas resumo dos vetores acima por sexo.

Sua vez!

4. Apresente as estatísticas resumo dos vetores acima somente para os indivíduos com idades entre 35 e 74 anos.
5. Apresente as estatísticas resumo por sexo e idade (menor que 65, maior que 65) para as seguintes variáveis:
   - IMC
   - RCQ

Bons estudos!