Revisão


ECONDATA Analytics
Núcleo de Ciência de dados dos alunos de Economia da PUC-Rio

O que são variáveis?

- **Variáveis** funcionam como recipientes que guardam algum tipo de informação. Numa metáfora, na imagem ao lado a variável é a **caixa** e os dados guardados na variável são as **bolinhas**.

- Existem várias formas de armazenar dados no R! Como escolher?

```
> caixa_1
[1] "Bola 1" "Bola 2" "Bola 3"
```

```
> caixa_4
[[1]]
[1] "Bola 1" "Bola 2" "Bola 3"
```

```
> caixa_2
     [,1]
[1,] "Bola 1"
[2,] "Bola 2"
[3,] "Bola 3"
```

```
> caixa_3
   bolas
1 Bola 1
2 Bola 2
3 Bola 3
```

ECONDATA Analytics

Núcleo de Ciência de dados dos alunos de Economia da PUC-Rio

Algumas características do R

- **R é case-sensitive :** então a diferentes variáveis. “ A ” e “ a ” são símbolos diferentes e se referem a **diferentes variáveis**;
- Comandos diferentes são separados por ponto e vírgula **“ ; ”**
  Todos os símbolos alfanuméricos são permitidos, incluindo “ . ” e “ _ ” .
- Comentários começam com **“ # ”**
- Como a maioria das linguagens de programação, R permite atribuir **valores a variáveis.**
- É possível designar valores a uma variável com **“<-”** ou “=”.

```
> A = 1; a = "viagem"
> print(A);
[1] 1
> print(a)
[1] "viagem"
> .b = 5
> print(.b + 5)
[1] 10
> # Comentario
```

ECONDATA Analytics

Núcleo de Ciência de dados dos alunos de Economia da PUC-Rio

Algumas variáveis armazenadas no R

```
> pi
[1] 3.141593
> letters
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i"
[10] "j" "k" "l" "m" "n" "o" "p" "q" "r"
[19] "s" "t" "u" "v" "w" "x" "y" "z"
> LETTERS
 [1] "A" "B" "C" "D" "E" "F" "G" "H" "I"
[10] "J" "K" "L" "M" "N" "O" "P" "Q" "R"
[19] "S" "T" "U" "V" "W" "X" "Y" "Z"
> month.abb
 [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun"
 [7] "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"
> month.name
 [1] "January"   "February"  "March"
 [4] "April"     "May"       "June"
 [7] "July"      "August"    "September"
[10] "October"   "November"  "December"
>
```

Tipos de dados no R

- character (“aula”)
- numeric (1.)
- integer (3)
- logical (TRUE or FALSE)
- vetor (tipos homogêneos)
- list (parecidos com vectors, mas
- heterogêneos) matrix
- missing values (NA)

Classe x tipo de objeto

- **Classe:**
  (i) vetores (ii) matrizes, (iii) data frames, (iv) listas e (v) funções.

- **Tipo:**
  Numérico, lógico, caracter, entre outros.

```
> b <- matrix(1:10, nrow = 2)
> class(b)
[1] "matrix"
> typeof(b)
[1] "integer"
```

ECONDATA Analytics

Núcleo de Ciência de dados dos alunos de Economia da PUC-Rio

i) Vetores: c()

- Vetores são variáveis unidimensionais e homogêneos, ou seja, todos os dados precisam ser do mesmo tipo.
- Os dados precisam estar dentro do comando c() separados por vírgula. Exemplos:

```r
# Vetor de caracteres
nomes <- c("Gabriel", "Sara", "Leonardo", "Beatriz")

# Vetor de números inteiros (integers)
ate_10 <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

# Vetor lógico
logic <- c(TRUE, FALSE, FALSE, FALSE)

# ':' fará uma sequência
ate_10 <- c(1:10)

# Funções que geram um vetor
ate_10 <- seq(1, 10, by = 1)
```

i) Vetores: c()

- Podemos acessar os valores dentro dos vetores de duas formas: pela posição do dado ou de um condicional.
- Para acessar, basta usar colchetes após a variável. Exemplos:

```
1  # Vetor de caracteres
2  nomes <- c("Gabriel", "Sara", "Leonardo", "Beatriz")
3
4  # Apenas o primeiro e terceiro nomes
5  nomes[c(1,3)]
6
7  # Vetor integer
8  ate_10 <- c(1:10)
9
10 # Dados acima de 5
11 ate_10[ate_10 > 5]
12
```

```
> # Apenas o primeiro e terceiro nomes
> nomes[c(1,3)]
[1] "Gabriel"  "Leonardo"
>
```

```
> # Dados acima de 5
> ate_10[ate_10 > 5]
[1]  6  7  8  9 10
>
```

i) Vetores: c()

- Importante notar que é possível incorporar mais dados num vetor, seja colocando mais dados pontuais ou juntando vetores.

```r
# Números pares até 10
pares <- seq(0, 10, by = 2)

# Números ímpares até 10
impares <- seq(1, 10, by = 2)

# Juntando tudo
ate_10 <- c(pares, impares)

# Ordenando
sort(ate_10)

```

```
> pares
[1]  0  2  4  6  8 10
> impares
[1] 1 3 5 7 9
> ate_10
 [1]  0  1  2  3  4  5  6  7  8  9 10
```

ii) Matrizes: matrix()

- Entendendo a lógica dos vetores, a matriz é uma combinação bidimensional de vetores, também homogênea.
- A partir do comando matrix() podemos montar uma matriz. Exemplos:

```
1  # Matriz contendo os 4 nomes
2  nomes <- matrix(data = c("Gabriel", "Sara", "Leonardo", "Beatriz"),
3                  nrow = 2, byrow = TRUE)
4
5  # Matriz contendo os múltiplos de 3
6  mult_3 <- matrix(data = c(1:9)*3, nrow = 3, byrow = FALSE)
7
8  # Matriz contendo dados lógicos
9  logic <- matrix(data = c(1:9) > 5, nrow = 3, byrow = FALSE)
```

ii) Matrizes: matrix()

```
> nomes
     [,1]       [,2]
[1,] "Gabriel"  "Sara"
[2,] "Leonardo" "Beatriz"
```

```
> mult_3
     [,1] [,2] [,3]
[1,]    3   12   21
[2,]    6   15   24
[3,]    9   18   27
```

```
> logic
      [,1]  [,2] [,3]
[1,] FALSE FALSE TRUE
[2,] FALSE FALSE TRUE
[3,] FALSE  TRUE TRUE
```

ii) Matrizes: matrix()

- Assim como nos vetores, podemos acessar os dados da matriz através dos colchetes.
- Porém, como é bidimensional, precisamos indicar as **[linhas, colunas]**.

```
1  # Primeiras duas linhas e colunas
2  mult_3[c(1:2), c(1:2)]
3
4  # Todas a terceira linha
5  mult_3[3,]
6
7  # Extraindo todos os valores maiores que 10
8  mult_3[mult_3 > 10]
9
```

```
> mult_3[c(1:2), c(1:2)]
     [,1] [,2]
[1,]    3   12
[2,]    6   15
```

```
> mult_3[3,]
[1]  9 18 27
```

```
> # Extraindo todos os valores maiores que 10
> mult_3[mult_3 > 10]
[1] 12 15 18 21 24 27
```

ii) Matrizes: matrix()

De forma similar aos vetores, dois comandos auxiliam a construção de matrizes: **cbind** e **rbind**. São para juntar matrizes, colando pelas **colunas** (column) ou **linhas** (rows).

```
1  # Até 4
2  ate_4 <- matrix(data = c(1:4), nrow = 2, byrow = FALSE)
3
4  # Até 8
5  ate_8 <- matrix(data = c(5:8), nrow = 2, byrow = FALSE)
6
7  # Juntando por linha e coluna:
8
9  por_linha <- rbind(ate_4, ate_8)
10 por_coluna <- cbind(ate_4, ate_8)
11
```

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

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

iii) Data.frames: data.frame()

- Similar à matriz, é bidimensional, porém cada coluna pode ser de um tipo diferente!
- Fazemos este objeto a partir da função data.frame():

```
1  # Uma planilha com dados de uma mercearia:
2  mercearia <- data.frame(frutas = c("Banana", "Abacaxi", "Maçã"),
3                          preco = c(2.75, 4.32, 6.38),
4                          quantidade = c(37, 0, 40))
5
```

```
> mercearia
   frutas preco quantidade
1  Banana  2.75         37
2 Abacaxi  4.32          0
3    Maçã  6.38         40
```

iii) Data.frames: data.frame()

- Para acessar uma coluna do data.frame, basta utilizar o cifrão. Para dados pontuais (linha 2 e coluna 3, por exemplo) ou condicionais, ainda é possível usar os colchetes.

```
1  # Quais itens estão com preço acima de 3?
2  mercearia$frutas[mercearia$preco > 3]
3
4  # É possível criar uma coluna que não existia anteriormente
5  mercearia$disp <- mercearia$quantidade != 0
```

```
> mercearia$frutas[mercearia$preco > 3]
[1] "Abacaxi" "Maçã"
```

```
> mercearia$disp
[1]  TRUE FALSE  TRUE
```

ECONDATA Analytics

Núcleo de Ciência de dados dos alunos de Economia da PUC-Rio

iv) Listas: list()

- A lista é um objeto mais peculiar: ele é multidimensional, heterogêneo e pode ter diversas classes. Fica complexo!
- Fazemos este objeto a partir da função list():

```
1  # Lista contendo um vetor, matriz e data.frame
2  lista <- list(c(1:10),
3                matrix(c(1:10), nrow = 2),
4                data.frame(contagem = c(1:10)))
5
```

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

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

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

iv) Listas: list()

- Para acessar cada classe de uma lista, basta usar o duplo colchetes. Depois, será usado a forma de acesso que ensinamos para vetores, matrizes e dataframes: [], [,] e $:

```
1  # Lista contendo um vetor, matriz e data.frame
2  lista <- list(c(1:10),
3                matrix(c(1:10), nrow = 2),
4                data.frame(contagem = c(1:10)))
5
6  # Acessando cada um dos objetos
7
8  lista[[1]][1:4]
9  lista[[2]][1,2:4]
10 lista[[3]]$contagem[5]
11
```

```
> lista[[1]][1:4]
[1] 1 2 3 4
> lista[[2]][1,2:4]
[1] 3 5 7
> lista[[3]]$contagem[5]
[1] 5
```

Operadores Aritméticos

Operador	Descrição
+	Operador de adição
-	Operador de subtração
*	Operador de multiplicação
/	Operador de divisão
^ ou **	Operador exponencial
:	Operador de sequência
x%%y	Operador de módulo
x%/%y	Operador de divisão de inteiros (integer)

```
> 180+567
[1] 747
> 560-98
[1] 462
> 40*65
[1] 2600
> 560/4
[1] 140
> 40^3
[1] 64000
> 1:5
[1] 1 2 3 4 5
> 5%%2
[1] 1
> 5%/%2
[1] 2
```

Exercícios:

1) Calcule as seguintes expressões: a) x = 360 + 430 - 40*12/15

b) x = (2^5)/3

2) Apresente uma sequência entre 5 e 12, depois divida essa sequência por 2.

Respostas:

```
> #1
> (360+430) - (40*12)/15
[1] 758
> #2
> (2^5)/3
[1] 10.66667
> #3
> a <- 5:12
> a/2
[1] 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0
>
```

Operadores de Comparação

Operador	Descrição
==	Igual a
!=	Diferente de
>	Maior que
<	Menor que
>=	Maior ou igual a
<=	Menor ou igual a

Os operadores de comparação sempre retornam um valor lógico **TRUE ou FALSE**.

```
> x=2
> y=3
> x==y
[1] FALSE
> x!=y
[1] TRUE
> x>y
[1] FALSE
> x<y
[1] TRUE
> x<=y
[1] TRUE
> x>=y
[1] FALSE
```

Operadores Lógicos

Operador	Descrição
!	Operador “Não”
&	Operador lógico “E”
|	Operador lógico “Ou”
xor	Operador lógico “XOR” - “Ou Exclusivo”

```
> x <- 0:2
> y <- 2:0
> !y == x

[1] TRUE FALSE TRUE
```

```
> x <- 0:2
> y <- 2:0
> (x < 1) & (y > 1)

[1] TRUE FALSE FALSE
```

```
> x <- 0:2
> y <- 2:0
> (x < 1) | (y > 1)

[1] TRUE FALSE FALSE
```

```
> v <- c(1:5)
> xor((v>=2), (v<=4))

[1] TRUE FALSE FALSE
FALSE TRUE
```

ECONDATA Analytics

Núcleo de Ciência de dados dos alunos de Economia da PUC-Rio

Exercícios

1) O código abaixo vai guardar no objeto `segredo` um número inteiro entre 0 e 10. Sem olhar qual número foi guardado no objeto, resolva os itens a seguir:

```
segredo <- round(runif(1, min = 0, max = 10))
```

a. Teste se o segredo é maior ou igual a 0.
b. Teste se o segredo é menor ou igual a 10.
c. Teste se o segredo é maior que 5.
d. Teste se o segredo é par.
e. Teste se `segredo * 5` é maior que a sua idade.

Soluções

1) **a.** Teste se o segredo é maior ou igual a 0.

b. Teste se o segredo é menor ou igual a 10.

c. Teste se o segredo é maior que 5.

d. Teste se o segredo é par.

e. Teste se `segredo * 5` é maior que a sua idade.

```
> segredo <- round(runif(1, min = 0, max = 10))
> #a
> segredo >= 0
[1] TRUE
> #b
> segredo <= 10
[1] TRUE
> #c
> segredo > 5
[1] FALSE
> #d
> segredo %% 2 == 0
[1] TRUE
> #e
> segredo * 5 > 21
[1] FALSE
```

Exercícios

2) O código abaixo vai guardar nos objetos `x, y e z` números inteiros entre -4 e 2. Resolva os itens a seguir:

```
x <- round (runif (1, min = -4, max = 2))

y <- round (runif (1, min = -4, max = 2))

z <- round (runif (1, min = -4, max = 2))
```

a. Realize a soma entre x e y, elevando seu resultado a z.
b. Teste se o x é menor que y, ao mesmo tempo que y é maior que z.
c. Teste se y é maior que x ou menor que z.
d. Teste se a negação de y maior que z é verdadeira ou falsa.

Soluções

2) **a.** Realize a soma entre x e y, elevando seu resultado a z.

b. Teste se o x é menor que y, ao mesmo tempo que y é maior que z.

c. Teste se y é maior que x ou menor que z.

d. Teste se a negação de y maior que z é verdadeira ou falsa.

```
> x <- round(runif(1, min = -4, max = 2))
> y <- round(runif(1, min = -4, max = 2))
> z <- round(runif(1, min = -4, max = 2))
> #a
> (x+y)^z
[1] 1
> #b
> (x < y) & (y > z)
[1] FALSE
> #c
> (y > x) | (y < z)
[1] TRUE
> #d
> !(y > z)
[1] TRUE
```

Controles de fluxo

Operador	Descrição
if/else	“Se”, “Caso contrário”

```
> x <- 1
> if(is.numeric(x)) {
> print(“Sim”)
> } else {
> print(“Não”)
> }

[1] “Sim”
```

ECONDATA Analytics

Núcleo de Ciência de dados dos alunos de Economia da PUC-Rio

Exercícios

1) O código abaixo vai guardar no objeto `dado` um número entre 1 e 6, como um dado não viciado. Sem olhar o resultado do objeto, faça um código que:

```
dado <- sample(x = 1:6, size = 1, replace = TRUE, prob = rep(1/6, 6))
```

a. Responde se o valor é maior, menor ou igual a 3.
b. Responde se o valor do dado é par ou ímpar.

Soluções

a.

```r
dado <- sample(x = 1:6, size = 1, replace = TRUE, prob = rep(1/6, 6))

if(dado < 3) {

  print("Menor que 3.")

} else if(dado > 3) {

  print("Maior que 3.")

} else {

  print("Igual a 3.")

}

```

Soluções

b.

```
3  if(dado %% 2 == 0) {
4
5    print(dado)
6    print("É par")
7
8  } else {
9
10   print(dado)
11   print("É ímpar")
12
13 }
14
```

Funções

- **Exemplo de função:**

divisao é uma função de dois argumentos, dois números colocados pelo usuário. Pegará o valor do primeiro e dividirá pelo segundo. Caso o segundo valor seja diferente de zero, será fornecido o resultado da divisão e, caso contrário, pedirá para o usuário fornecer outro valor.

- **Exemplo no R:**

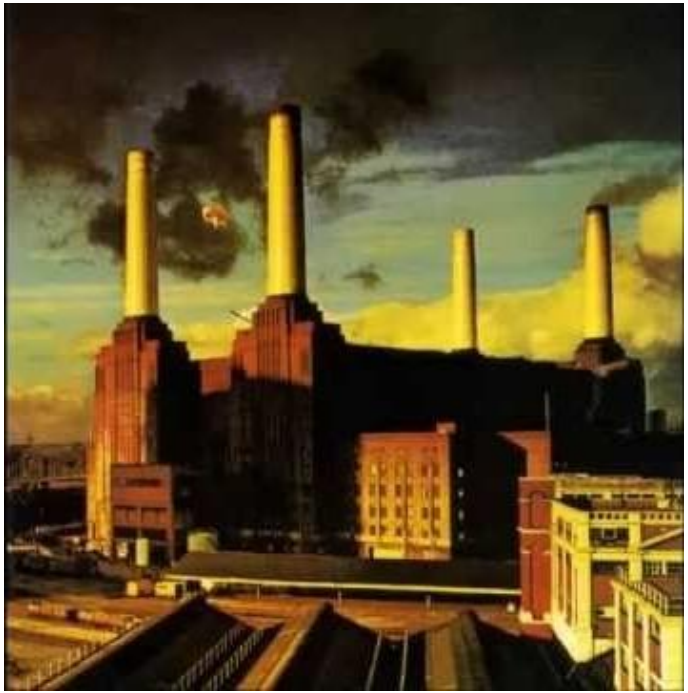
```r
divisao <- function(a, b) {
  if(b == 0) {
    print("Forneça outro valor para o denominador.")
  } else {
    valor <- a / b
    print(paste0("O valor da divisão entre ", a, " e ", b, " é ", valor, "."))
  }
}
```

Por que usar funções?

- Com tarefas repetitivas, é como escrever o código apenas uma vez e utilizá-lo quantas quiser, guardando parte do seu script no R.

- Caso seu código seja grande, dividi-lo em algumas funções torna-se mais simples de compreender e consertar, caso tenha algum erro.

- Pense que uma fábrica não tem apenas uma etapa: cada parte do processo produtivo é dividida para ganhos de escala, fácil manutenção e redução de custos.



ECONDATA Analytics

Núcleo de Ciência de dados dos alunos de Economia da PUC-Rio

Exercícios

1) a. Crie uma função que calcule o crescimento percentual do PIB anual, tendo como argumentos o PIB atual e o PIB do ano anterior.

b. Crie uma função que calcule o índice de preços ao consumidor (IPC) com base nos preços e nas quantidades de um conjunto de bens. Teste com os seguintes vetores: preço = (2, 3, 4); quantidade = (100, 150, 200).

c. Escreva uma função que receba um salário como argumento e retorne a alíquota de imposto de renda a ser aplicada sobre esse salário (OBS: Considere a tabela abaixo).

Rendimento tributável mensal	Alíquota cobrada	Valor a deduzir do IR
até R$ 1.903,98	Isento	R$ 0
de R$ 1.903,99 a R$ 2.826,65	7,5%	R$ 142,80
de R$ 2.826,66 a R$ 3.751,05	15%	R$ 354,80
de R$ 3.751,06 a R$ 4.664,68	22,5%	R$ 636,13
acima de R$ 4.664,68	27,5%	R$ 869,36

Soluções

a.

```r
crescimento_pib <- function(pib_atual, pib_anterior) {

  crescimento <- ((pib_atual - pib_anterior)/pib_anterior)*100

  return(crescimento)

}

crescimento_pib(5000,4500)
```

Soluções

b.

```r
ipc <- function(preco, quantidade) {
  peso <- quantidade/sum(quantidade)
  ipc <- sum(preco*peso)
  return(ipc)
}
preço <- c(2, 3, 4)
quantidade <- c(100, 150, 200)
ipc(preco, quantidade)
```

Soluções

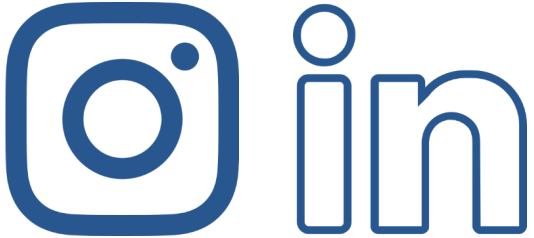
c.

```r
aliquota_irrf <- function(salario){
  if(salario <= 1903.98) {
    return(0)
  } else if(salario <= 2826.65) {
    return(0.075)
  } else if(salario <= 3751.05) {
    return(0.15)
  } else if(salario <= 4664.68) {
    return(0.225)
  } else {
    return(0.275)
  }
}
aliquota_irrf(5000)
```

Funções existentes no R

Função	Descrição
sqrt()	Raiz quadrada.
abs()	Valor absoluto.
exp()	Exponencial.
log10()	Logaritmo na base 10.
log()	Logaritmo na base e.
sin() cos() tan()	Funções trigonométricas.
asin() acos() atan()	Funções trigonométricas inversas.
sort()	Ordena os elementos de um vetor.
read.csv()	Lê um arquivo no padrão csv.
max(x)	Retorna o maior valor encontrado em x (vetor, etc).
mean(x)	Retorna a média dos elementos de x (vetor, etc).
tolower()	Reescreve uma string em letras minúsculas.
hist()	Cria um histograma.

Nos siga nas Redes Sociais!



- @econdata_analytics
- EconData Analytics

ECONDATA Analytics

Núcleo de Ciência de dados dos alunos de Economia da PUC-Rio