Linguagem de Programação: R

Departamento de Sistemas de Informação
Universidade Federal de Sergipe (UFS) – Itabaiana, SE – Brasil
Docente: André Luis Meneses Silva
Docentes: Jéssica Santos Portilio
Rafael Rezende Santana Carvalho
Carlos Eduardo Pereira Silva

1. Introdução

Foi criada originalmente por Ross Ihaka e por Robert Gentleman no departamento de Estatística da universidade de Auckland, Nova Zelândia, em 1995, com a finalidade de obter um melhor ambiente de software para laboratórios de estatística.

R é uma linguagem de programação e ambiente de software livre e código aberto (open source). R é uma linguagem de programação estatística e gráfica que vem se especializando na manipulação, análise e visualização de dados, sendo atualmente considerada uma das melhores ferramentas para essa finalidade**.**

A linguagem R é uma linguagem de programação dinamicamente tipada, ou seja, podemos modificar os tipos de dados contidos em variáveis em programas que já estejam em execução, com isso, temos uma melhoria na hora de programarmos, pois não precisaremos realizar conversões dos tipos de dados.

R tem três princípios (Chambers 2016):

- **Princípio do Objeto**: Tudo que existe em R é um objeto;
- **Princípio da Função**: Tudo que acontece no R é uma chamada de função;
- **Princípio da Interface**: Interfaces para outros programas são parte do R.

O R é uma linguagem baseada em linhas de comando, e as linhas de comando, são executadas uma de cada vez, no console, ou por meio de alguns IDEs (ambientes de desenvolvimento integrados) como RStudio. Assim que o prompt de comando está visível na tela do console, o R indica que o usuário está pronto para inserir as linhas de comando. O símbolo padrão do prompt de comando é “>,” porém ele pode ser alterado. Para isso, use a linha de comando, por exemplo:

```
options(prompt = "R>")
# Toda vez que o console iniciar, começarar por 'R>'
10
[1] 10
```

- Instalação do R: https://cran.r-project.org/bin/windows/base/
- Instalação do RStudio: https://rstudio.com/products/rstudio/download/#download

2. Lexemas do R

O conjunto de símbolos que podem ser utilizados no R depende do sistema operacional e do país em que o R está sendo executado. Basicamente, todos os símbolos alfanuméricos podem ser utilizados, mas para evitar problemas quanto ao uso das letras aos nomes, opte pelos caracteres ASCII.

A escolha do nome associado a um objeto tem algumas regras:

- Deve consistir em letras, dígitos, . e _;
- Os nomes devem ser iniciado por uma letra ou um ponto não seguido de um número, isto é, Ex.: .123, 1n, dentre outros;
- As letras maiúsculas se distinguem das letras minúsculas;
- Não pode iniciar por _ ou dígito, é retornado um erro no console caso isso ocorra;
- Não pode usar qualquer uma das palavras reservadas pela linguagem, isto é, TRUE, FALSE, if, for, dentre outras, que pode ser consultado usando o comando ?Reserved().

Como convenção se pode usar:

Tipo	Exemplo
apenas letras minúsculas	idadealuno
palavras separadas por .	idade.aluno
palavras separadas por _	idade_aluno
CamelCase minúsculas	idadeAluno
CamelCase maiúsculas	IdadeAluno

3. Comentários em R

Comentários nos scripts do R são seguidos do símbolo #, e tudo que estiver após # não será executado.
Exemplo: # Gráfico dos números de 1 a 10

```
plot(1:10)
```

4. Palavras reservadas

Em programação de computadores, uma palavra reservada é uma palavra que, em algumas linguagens de programação, não pode ser utilizada como um identificador por ser reservada para uso da gramática da linguagem.

Palavras Reservadas	Uso
If, else, repeat, while, function, for, in, next, break	Usados em loops, condicionais e funções
TRUE, FALSE	Constantes lógicas
NULL	Valor ausente ou indefinido
Inf	Infinito
NaN	Not a Number (não número)
NA	Not Available (não disponível)
NA_integer_, NA_real_, NA_complex_, NA_ character_	Constantes de vetores com valores ausent
…	Permite que uma função passe argumento para outra

5. Comandos

A tabela abaixo reúne alguns comandos usados para a manipulação do ambiente de trabalho em **R**:

Comando:	Resultado:
getwd()	pasta atual

setwd(“pasta”)	altera pasta atual
dir.create(“teste”)	cria a pasta *teste*
dir()	lista arquivos na pasta atual
save.image(“arquivo.R”)	grava estado da sessão em *arquivo.R*
load(“arquivo.R”)	recupera a sessão gravada em *arquivo.R*
ls()	lista os objetos carregados na workspace atual
rm(objectlist)	remove um ou mais objetos
help(options)	fornece informação sobre as opções disponíveis
options()	exibe e modifica as opções disponíveis
history(#)	exibe os últimos # commandos (default = 25)
savehistory(“myfile”)	grava o histórico de comandos no arquivo myfile (default =.Rhistory)
loadhistory(“myfile”)	recupera o histórico de comandos no arquivo myfile (default = .Rhistory)
save.image(“myfile”)	grava o workspace no arquivo myfile (default = .RData)
save(objectlist, file=”myfile”)	grava objetos específicos em um arquivo
load(“myfile”)	carrega para a sessão atual o workspace
sink(“nomeArquivo.txt”)	redireciona todas as saídas do R para o arquivo. O arquivo gravado é fecha com sink()
q()	encerra a sessão de R (oferecendo ao usuário salvar o workspace.)
Operador de ajuda	Efeito
help.start()	ajuda geral
help(“foo”) (?foo)	ajuda sobre a função foo (aspas optionais)
help.search(“foo”) (??foo)	busca pela string ‘foo’ no sistema de ajuda
example(“foo”)	exemplos de uso da função foo
RSiteSearch(“foo”)	busca pela string ‘foo’ no sistema de ajuda online e mailing lists
vignette()	lista todas as vignettes disponíveis para os pacotes instalados
vignette(“foo”)	lista todas as vignettes disponíveis para o tópico “foo”

data()	lista todos os exemplos disponíveis de conjunto de dados contidos nos pac instalados

6. Instalação de pacotes

Por default R carrega pacotes padrões
incluindo: base, datasets, utils, grDevices, graphics, stats e methods.

Função	Descrição
.libPaths()	exibe localização da biblioteca.
install.packages("*pacote*")	instala *pacote*.
library()	exibe pacotes instalados.
library(*pacote*)	Carrega *pacote* para uso.
installed.packages()	exibe pacotes instalados.
remove.packages("*pacote*")	desinstala *pacote*.
search()	informa os pacotes carregados para uso.
update.packages()	atualiza pacotes instalados.
help(package="*pacote*")	Documentação sobre *pacote*
library(help = "*pacote*")	Sobre *pacote* (mais compacto que help(package="pkt")

7. Operadores de Comparação em R

Esses operadores trabalham sempre com dois valores de retorno possíveis: **TRUE** (verdadeiro) ou **FALSE** (falso), também denotados pelas letras maiúsculas **T** e **F**, respectivamente.

No R, além dos estados TRUE ou FALSE, há também um terceiro para indicar valor ausente *(*missing value*)* identificado por NA.

Também conhecidos como operadores relacionais, permitem estabelecer a relação entre dois valores de entrada, e retornar um valor lógico verdadeiro ou falso dependendo da relação.

Por exemplo, um operador de comparação pode comparar dois números e dizer se eles são iguais ou não.

Os operadores de comparação em R são os seguintes:

Operador	Significado
==	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.

Também conhecidos como operadores booleanos, permitem trabalhar com múltiplas condições relacionais na mesma expressão, e retornam valores lógicos verdadeiro ou falso.

Os operadores lógicos disponíveis em R são:

Operador	Descrição	Exemplo	Explicação
&	AND lógico	> x <− 0:2 > y <− 2:0 > (x < 1) & (y > 1) [1] TRUE FALSE FALSE	Versão vetorizada. Compara dois elementos do tipo vetor e retorna um vetor de TRUEs e FALSEs
&&	AND lógico	> x <− 0:2 > y <− 2:0 > (x < 1) && (y > 1) [1] TRUE	Versão não-vetorizada. Compara apenas o primeiro valor de cada vetor, retornando um valor lógico.
|	OR lógico	> x <− 0:2 > y <− 2:0 > (x < 1) | (y > 1) [1] TRUE FALSE FALSE	Versão vetorizada. Compara dois elementos do tipo vetor e retorna um vetor de TRUEs e FALSEs
||	OR lógico	> x <− 0:2 > y <− 2:0 > (x < 1) || (y > 1) [1] TRUE	Versão não-vetorizada. Compara apenas o primeiro valor de cada vetor, retornando um valor lógico.
!	NOT lógico	> x <− 0:2 > y <− 2:0 > !y == x [1] TRUE FALSE TRUE	Negação lógica. Retorna um valor lógico único ou um vetor de TRUE / FALSE.

Operador	Descrição	Exemplo	Explicação
xor	XOR	> x <− TRUE > y <− FALSE > xor(x,y) [1] TRUE	Ou Exclusivo. Retorna valor lógico TRUE se ambos os valores de entrada forem diferentes entre si, e retorna FALSE se os valores forem iguais.

Além desses operadores lógicos, também temos disponível em R
a função ***isTRUE(x)***, que verifica se o valor lógico armazenado na variável x é verdadeiro ou não.

8. Principais operadores aritméticos

Operador	Descrição
+	Operador de adição
-	Operador de subtração
*	Operador de multiplicação
/	Operador de divisão
:	Operador de sequência
^	Operador exponencial
%%	Operador de módulo

Alguns exemplos de operações aritméticas e lógicas:

```r
2+2 # Soma
[1] 4
8-3 # Subtração
[1] 5
3*8 # Multiplicação
[1] 24
8/2 # Divisão
[1] 4
2^8 # Potências
[1] 256
(2+4)/7 # Prioridades de solução. Diferente de 2+4/7
[1] 0.8571429
10%/%3 # Parte inteira da divisão. O inteiro da divisão de um número por outro
[1] 3
10%%3 # Módulo. O resto da divisão de um número por outro
[1] 1
matrix(c(1, 2, 3),
       nrow = 3,
       ncol = 1)%*%matrix(c(4,2,3),
                          nrow = 1,
                          ncol = 3) # Produto de matrizes
```

```
     [,1] [,2] [,3]
[1,]    4    2    3
[2,]    8    4    6
[3,]   12    6    9
matrix(c(1, 2, 3),
       nrow = 1,
       ncol = 3)%o%matrix(c(4,2,3),
                          nrow = 1,
                          ncol = 3) # Produto diádico
, , 1, 1

     [,1] [,2] [,3]
[1,]    4    8   12

, , 1, 2

     [,1] [,2] [,3]
[1,]    2    4    6

, , 1, 3
```

```
     [,1] [,2] [,3]
[1,]    3    6    9
matrix(c(1, 3, 2, 4),
       nrow = 2,
       ncol = 2)%x%matrix(c(0, 6, 5, 7),
                          nrow = 2,
                          ncol = 2) # Produto de Kronecker
     [,1] [,2] [,3] [,4]
[1,]    0    5    0   10
[2,]    6    7   12   14
[3,]    0   15    0   20
[4,]   18   21   24   28
c("a", "b", "c")%in%c("b", "c") # Operador de correspondência
[1] FALSE  TRUE  TRUE
2<3 # Comparar, menor
[1] TRUE
3>3 # Comparar, maior
[1] FALSE
3<=3 # Comparar, menor ou igual
[1] TRUE
2>=3 # Comparar, maior ou igual
[1] FALSE
3==3 # Comparar, exatamente igual
[1] TRUE
```

```
[1] TRUE
2!=3 # Comparar, diferente
[1] TRUE
!3==3 # Lógico, NÃO. Inverter resultado de teste lógico
[1] FALSE
3==3 & 3!=3 # Lógico, critério aditivo E. Operação elementar
[1] FALSE
3==3 | 3!=3 # Lógico, critério aditivo OU. Operação elementar
[1] TRUE
3==3 && 3!=3 # Lógico, E.
[1] FALSE
3==3 || 3!=3 # Lógico, OU.
[1] TRUE
```

```
x <- c(TRUE, FALSE, 0, 6)
x
[1] 1 0 0 6
y <- c(FALSE, TRUE, FALSE, TRUE)
y
[1] FALSE  TRUE FALSE  TRUE
!x
[1] FALSE  TRUE  TRUE FALSE
x&y
[1] FALSE FALSE FALSE  TRUE
x&&y
[1] FALSE
x|y
[1]  TRUE  TRUE FALSE  TRUE
x||y
[1] TRUE
```

9. Manipulação de textos (ou strings)

O R possui várias funções para manipular textos (ou *strings*). No entanto, as funções do base não possuem uma interface consistente e cada uma tem a sua forma de passar os parâmetros, dificultando a programação durante a análise.

- Sintaxe unificada, o que auxilia na memorização das funções e leitura do código.
- Todas as funções são vetorizadas.
- Construído sobre a biblioteca ICU, implementada em C e C++, uma garantia de resultados mais rápidos.

Regras básicas do pacote

- As funções de manipulação de texto começam com str_. Caso esqueça o nome de uma função, basta digitar stringr::str_ e apertar TAB para ver quais são as opções.
- O primeiro argumento da função é sempre uma *string* ou um vetor de *strings*.

9.1 Funções básicas

str_length: Esta função recebe como argumento um vetor de *strings* e retorna o número de caracteres de cada *string*. O espaço " " é considerado um caractere.

```
Exemplo: str_length(c("São Paulo", "Rio de Janeiro",
                      "Rio Grande do Norte", "Acre"))
## [1]  9 14 19  4
```

str_to_upper, str_to_lower, str_to_title

Essas funções servem para modificar a caixa das letras. Alguns exemplos:

```
s <- "Somos a curso-r"
str_to_lower(s)
## [1] "somos a curso-r"
str_to_upper(s)
## [1] "SOMOS A CURSO-R"
str_to_title(s)
## [1] "Somos A Curso-R"
```

str_trim: A função str_trim() ajuda removendo os espaços excedentes antes e depois da string.

```
string_aparada <- str_trim(s)
as.factor(string_aparada)
## [1] M F F M F M
## Levels: F M
```

str_sub: Utilizada para obter uma parte fixa de uma string

```
Exemplos: s <- c("01-Feminino", "02-Masculino", "03-Indefinido")
# pegar do quarto até o último caractere
str_sub(s, start = 4)
## [1] "Feminino"   "Masculino"  "Indefinido"
```

```
# pegar apenas os dois primeiros caracteres
str_sub(s, end = 2)
## [1] "01" "02" "03"
```

obtendo os últimos 2 caracteres.

```
s <- c("Feminino-01", "Masculino-02", "Indefinido-03")
str_sub(s, end = -4)
## [1] "Feminino"   "Masculino"  "Indefinido"
str_sub(s, start = -2)
## [1] "01" "02" "03"
```

usando os argumentos start e end conjuntamente.

```
s <- c("__SP__", "__MG__", "__RJ__")
```

```
str_sub(s, 3, 4)
## [1] "SP" "MG" "RJ"
```

str_c: Concatena strings em uma única string.

```
string1 <- "O valor p é: "
string2 <- 0.03

str_c(string1, string2)
## [1] "O valor p é: 0.03"
```

Pode misturar objetos com strings definidas diretamente na função.

```
string1 <- "Brigadeiro"
string2 <- "bom"
string3 <- "melhor"

str_c(string1, " é a prova de que não existe nada tão ", string2,
      " que não possa ficar ", string3, ".")
## [1] "Brigadeiro é a prova de que não existe nada tão bom que não possa
ficar melhor."
```

10. Expressões Regulares (regex)

Permitem identificar conjuntos de caracteres, palavras e outros padrões por meio de uma sintaxe concisa.

Vamos estudar expressões regulares por meio de exemplos e da função str_detect(). Ela retorna TRUE se uma *string* atende a uma expressão regular e FALSE caso contrário.
Por exemplo:

```
str_detect("sao paulo", pattern = "paulo$")
## [1] TRUE
str_detect("sao paulo sp", pattern = "paulo$")
## [1] FALSE
```

10.1 Funções que utilizam regex

str_detect()

Retorna TRUE se a regex é compatível com a string e FALSE caso contrário.

```
library(stringr)
str_detect("sao paulo", pattern = "paulo$")
## [1] TRUE
```

```
str_detect("sao paulo sp", pattern = "paulo$")
## [1] FALSE
```

str_replace() e str_replace_all()

Substituem um padrão (ou todos) encontrado por um outro padrão.

Substituindo apenas a primeira ocorrência:

```
cidades <- c("S. José do Rio Preto", "São Paulo", "S. José dos Campos",
"São Roque", "S. S. da Grama")

str_replace(cidades, "S[.]", "São")
## [1] "São José do Rio Preto" "São Paulo"           "São José dos Campos"
## [4] "São Roque"             "São S. da Grama"
```

str_extract() e str_extract_all()

As funções `str_extract()` e `str_extract_all()` extraem padrões de uma *string*. No exemplo abaixo, pegamos apenas os sobrenomes de cada integrante do grupo.

```
r_core_group <- c(
  'Douglas Bates', 'John Chambers', 'Peter Dalgaard',
  'Robert Gentleman', 'Kurt Hornik', 'Ross Ihaka', 'Tomas Kalibera',
  'Michael Lawrence', 'Friedrich Leisch', 'Uwe Ligges', '...'
)

sobrenomes <- str_extract(r_core_group, '[:alpha:]+$')
sobrenomes
##  [1] "Bates"     "Chambers"  "Dalgaard"  "Gentleman" "Hornik"    "Ihaka"
##  [7] "Kalibera"  "Lawrence"  "Leisch"    "Ligges"    NA
```

str_split() e str_split_fixed()

Essas funções separam uma *string* em várias de acordo com um separador.

```
texto <- 'Durante um longo período de tempo o "R" foi escrito "P" como no
alfabeto cirílico. O seu nome no alfabeto fenício era "rech". Seu significado
era o de uma cabeça, representada pela adaptação do hieróglifo egípcio de
uma cabeça. Transformou-se no "rô" dos gregos. Os romanos modificaram o
rô acrescentando um pequeno traço para diferenciá-lo do no nosso P.'

str_split(texto, fixed('.'))
## [[1]]
## [1] "Durante um longo período de tempo o \"R\" foi escrito \"P\" como no
alfabeto cirílico"
## [2] " O seu nome no alfabeto fenício era \"rech\""
## [3] " Seu significado era o de uma cabeça, representada pela adaptação
do hieróglifo egípcio de uma cabeça"
## [4] " Transformou-se no \"rô\" dos gregos"
```

```
## [5] " Os romanos modificaram o rô acrescentando um pequeno traço para diferenciá-lo do no nosso P"
## [6] ""
```

str_subset()

A função str_subset() retorna somente as strings compatíveis com a regex.

```
frases <- c('a roupa do rei', 'de roma', 'o rato roeu')
str_subset(frases, 'd[eo]')
## [1] "a roupa do rei" "de roma"
```

É o mesmo que fazer subset usando a função str_detect.

```
frases[str_detect(frases, "d[eo]")]
## [1] "a roupa do rei" "de roma"
```

Referências:

https://bookdown.org/wevsena/curso_r_tce/curso_r_tce.html#manipulacao-de-dados

https://bendeivide.github.io/cursor/nbasico#comandos-workspace

https://livro.curso-r.com/7-4-o-pacote-stringr.html

https://tiagoolivoto.github.io/e-bookr/manipula.html

https://carloscinelli.com/files/Cinelli%20(2016)%20-%20R%20course.pdf

https://didatica.tech/a-linguagem-r/