


Metodologias Informacionais com R

Módulo I: Introdução à Linguagem R

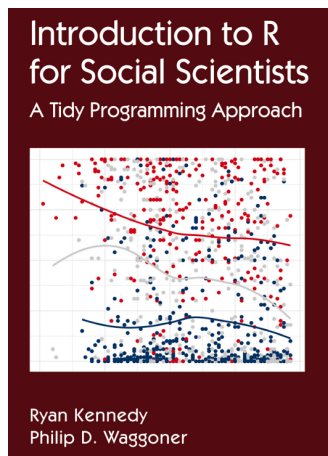
Telmo dos Santos Klipp (telmo.klipp@inpe.br)

Informações Gerais sobre o Curso

- Materiais disponibilizados via **Classroom**;
- O aprendizado requer a prática que será constante nas aulas;
- Será cobrado ao menos uma atividade semanal .

Bibliografia Básica:

- Kennedy, R., & Waggoner, P. D. (2021). Introduction to r for social scientists: a tidy programming approach. CRC Press.



Bibliografia Complementar:

- Wickham, H., & Grolemund, G. (2016). R for data science: import, tidy, transform, visualize, and model data. " O'Reilly Media, Inc.". Disponível em: <https://r4ds.had.co.nz/index.html>. Acesso em: 12 de maio, 2023. (Online)
- Damiani, A. et. al., (2022). Ciência de Dados em R. Curso-R. Disponível em: <https://livro.curso-r.com>. Acesso em: 12 de maio, 2023. (Online)
- de Aquino, J. A. (2014). R para cientistas sociais. Editora da UESC (editus). Disponível em: <http://www.uesc.br/editora/>. Acesso em: 12 de maio, 2023.
- de Oliveira, P. F., Guerra, S., McDonnell, R. (2018). Ciência de Dados com R: Introdução. Editora IBPAD. Disponível em: <https://cdr.ibpad.com.br/index.html>. Acesso em: 12 de maio, 2023. (Online)

Na última aula vimos:

- O que são o R (ferramenta e linguagem) e o RStudio (IDE).
- Como acessar e instalar o R e o RStudio.
- Motivos para usar o R.
- Apresentação do RStudio.
- Conceitos iniciais do R: comandos, funções, buscar ajuda no R e *scripting*.

Solução da tarefa 2 da aula anterior

- As funções `paste()` e `paste0()` concatenam blocos de texto (ou caracteres). Além disso, qualquer objeto que possa ser convertido para texto pode ser concatenado. A conversão é feita automaticamente pelas funções. Portanto, números e textos podem ser concatenados. No entanto, `paste()` adiciona por padrão espaços entre os componentes que serão concatenados enquanto `paste0()` une por padrão os elementos sem a adição de espaços. A forma de concatenação padrão dessas funções pode ser alterada usando o parâmetro `sep`. Por fim, a tarefa pode ser solucionada escrevendo no *R script* qualquer um dos comandos:

```
paste("O dobro de ", 2, " é ", 2*2, "!", sep = " ")
paste0("O dobro de ", 2, " é ", 2*2, "!")
```

Mas números e textos são representados de forma diferente no R?
O que é conversão de objetos? O que são **funções** e **objetos**?

Fundamentos do R

- Entre os fundamentos do R, duas regras gerais se aplicam:
 - 1. Tudo no R é um objeto;
 - 2. Qualquer coisa que realize uma tarefa é uma função.
- Além disso:
 - O R é um ambiente/linguagem de *scripting*. Isso significa que o código é interpretado no momento de sua execução;
 - Toda linguagem de programação tem uma sintaxe específica e formas de representação de dados.

Fundamentos do R – objetos

No R, objetos são estruturas que representam e armazenam diferentes tipos de dados. São exemplos de objetos:

```
7          # numeric
7L         # integer
'R'        # character
FALSE      # logical
3i         # complex
paste0()   # function
```

São operações permitidas entre objetos:

```
1 + 1; 1 < 2; 4 + 3i;
```

São objetos, mas não é permitida a operação:

```
1 + '1'
'1' + "dois"
```

Fundamentos do R – objetos

Assim como no mundo real, objetos no R tem atributos. Por enquanto, estamos interessados apenas no atributo do objeto que indica seu tipo. Este pode ser revelado pela função `class()`.

```
class(1 + 1)
```

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

```
class("abcd")
```

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

```
class(7i)
```

```
## [1] "complex"
```

```
class(is.complex(7i))
```

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

```
class(version)
```

```
## [1] "simple.list"
```

```
class(class)
```

```
## [1] "function"
```

Atributos determinam o que pode ser feito com os objetos individualmente e entre os mesmos.

Fundamentos do R – objetos

Mas onde estão esses objetos afinal? Objetos podem ser persistidos (armazenados) na memória do computador (RAM) com o operador de atribuição `<-` e permanecerem disponíveis para uso na sessão do R atual, sendo listados na aba Ambiente (do RStudio). Teste alguns exemplos conforme a sintaxe do R abaixo. Dica: use "Alt + -" no RStudio para obter `<-`.

Sintaxe : `nome_do_objeto <- valor`

Fundamentos do R – objetos

Exemplos de atribuição:

```
letras <- "abcdefgh"
numero_complexo <- 3i
numero_inteiro <- 3L
numero_real <- 2.7
verdade <- TRUE
falso <- FALSE
rinfo <- version
rstudio_info <- rstudioapi::versionInfo()
```

Fundamentos do R – objetos

Alguna recomendação para nomes de objetos? Procure usar nomes curtos e intuitivos. Considere que a linguagem R é *case sensitive* (difere entre letras maiúsculas e minúsculas), sendo assim:

```
(number <- 1 + 1)
```

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

```
(Number <- 4 + 3)
```

```
## [1] 7
```

```
number == Number
```

```
(number <- 4 + 3)
```

```
## [1] 7
```

```
Number
```

```
## [1] 7
```

```
number == Number
```

Fundamentos do R – objetos

Alguma recomendação para nomes de objetos? Os nomes podem combinar números e letras, mas nunca começar por números.

```
number9 <- 9  
9number <- 9
```

No entanto, como boa prática, evite usar números e letras combinados em um único nome.

Fundamentos do R – objetos

Alguma recomendação para nomes de objetos? Evite usar símbolos nos nomes.

```
eureka! <- "Achei a solução!"  
number&two <- 2  
number-two <- "two"  
number.two <- 2
```

Considere apenas o uso do símbolo (underline) para separar palavras em um nome.

```
numero_dois <- 2  
numero_complexo <- 3i  
numero_inteiro <- 3L  
numero_real <- 2.7
```

Fundamentos do R – objetos

Alguma recomendação para nomes de objetos? Evite usar nomes de funções em objetos.

```
class("two")  
class <- 2
```

Além disso, existem palavras reservadas no R.

```
TRUE <- 1  
FALSE <- 0
```

Palavras reservadas podem ser consultadas usando `?reserved`.

Fundamentos do R – objetos

É possível fazer atribuições em um objeto com valores à esquerda do operador que assume a forma `->`. No entanto, essa forma de atribuição raramente é usada, pois dificulta a clareza do código.

```
# Comando para demonstrar atribuição em objeto à esquerda  
4 + 3 -> number
```

O sinal `=` (igualdade) também pode ser usado para fazer atribuições, mas deve ser usado em situações específicas (obs: voltaremos nesse assunto).

Curiosidade. Existe uma função para fazer atribuições em objetos (`assign`).

```
(assign('number', 7))
```

```
## [1] 7
```

Questões e Prática

- Como identificar qual o tipo de um objeto?
- Caso eu queira obter informações de como atribuir valores a um objeto, como posso proceder?
- Crie um *R script* que persista na memória objetos para representar dois anos quaisquer (ex: 1700 e 2000) e realizar as seguintes operações:
 - Obter o valor absoluto da diferença entre os anos e armazenar o resultado. Obs: a função `abs()` fornece o valor absoluto;
 - Compare os valores de todos os objetos armazenados com o resultado obtido usando o operador lógico `>=`;
 - Mostre os comandos e os resultados obtidos carregando o *script*.
- É possível fazer comparações lógicas entre caracteres (ex: 'A' < 'B')? Faça alguns testes.

Fundamentos do R – objetos

Conversão? Um objeto pode ser convertido em outro. Além disso, é possível testar o tipo de um objeto.

```
number <- paste0('4', '3')  
number <- as.numeric(number) + 7  
number <- as.character(number)
```

Qual o tipo do objeto criado acima? Qual o seu valor?

```
is.numeric(number)
```

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

```
is.character(number)
```

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

```
number
```

```
## [1] "50"
```


Fundamentos do R – objetos

Conversão? Funções no estilo `as.tipo_do_objeto()` e `is.tipo_do_objeto()` servem para conversão e teste do tipo do objeto.

O que acontecerá nas conversões listadas a seguir?

```
number <- as.numeric("16.95")
number <- as.numeric("16,95")
number <- as.integer("45L")
number <- as.integer("12.5")
number <- as.integer(TRUE)
flag <- as.logical(1)
flag <- as.logical('1')
flag <- as.logical("False")
flag <- as.logical("FalsE")
flag <- as.logical("true")
```

```
# gera tipo numérico
# produz NA
# produz NA
# gera tipo inteiro
# gera tipo inteiro
# gera tipo lógico
# produz NA
# gera tipo lógico
# produz NA
# gera tipo lógico
```

Mas quais e quantos tipos de objetos existem no R?

Fundamentos do R – objetos

- Existem 6 tipos básicos (atômicos) de objetos no R:
 - numeric;
 - integer;
 - character;
 - logical;
 - complex;
 - raw.
- São exemplos de objetos não básicos no R:
 - factor;
 - data.frame;
 - function;
 - lists;
 - matrices, arrays.

RStudio – Ambiente e Histórico

- Na aba **Ambiente** são listadas informações dos objetos e funções carregados na sessão atual do R. É possível interagir, importar/exportar dados e obter informações sobre alocação de memória (do computador) ou liberar memória que esta sem uso no R.
- A aba **Histórico** fornece a lista de comandos executados no console, também disponíveis interativamente usando as setas ↑ e ↓ do teclado.

Algumas funções no R executam algumas dessas tarefas:

```
ls()           # listar objetos/funções
objects()      # similar ao ls
rm()           # remover objetos da sessão atual do R
gc()           # liberar a memória sem uso
history()      # mostrar o histórico de comandos
```

Fundamentos do R – funções

No R, funções realizam tarefas manipulando dados. Assim, em muitas das vezes, nos desejaremos que uma função receba um objeto, opere sobre o mesmo e, ao final, retorne um novo objeto. Uma função tem a seguinte estrutura:

```
Sintaxe : nome_da_funcao <- function() {}
```

A sintaxe acima também é chamada de declaração (ou definição) de uma função. Já a chamada (ou invocamento) de uma função tem a forma:

```
Sintaxe : nome_da_funcao()
```

Fundamentos do R – funções

Uma função no R:

- Pode receber parâmetros (argumentos) na forma de objetos;
- Pode realizar operações sobre objetos;
- Retorna ao menos um novo objeto.

Exemplo de declaração de função:

```
returnNull <- function() {}  
returnNull()
```

Uma função que não recebe nada e nem manipula dados não nos interessa.

Questões e Prática

Crie uma função para somar dois números informados pelo usuário.

Abaixo segue um protótipo com os argumentos necessários para a função:

```
soma <- function(a, b) {}
```

Uma possível solução:

```
soma <- function(a, b) {  
  resultado <- a + b  
  resultado  
}
```

Chamada da função:

```
soma(4, 3)
```

```
## [1] 7
```

Fundamentos do R – funções

Já sabemos que uma função pode ter nenhum ou muitos parâmetros e retornar objetos. Uma função também pode ter parâmetros declarados com valores padrão.

```
soma <- function(a = 0, b = 0) {  
  # Função para demonstrar o uso do sinal "=" e "return"  
  resultado <- a + b  
  return(resultado)  
}
```

A propósito, o sinal de **=** (igualdade) geralmente é usado para atribuição de valores em parâmetros de uma função. Pode ser usado na declaração de uma função para atribuir valores padrão (*default*) ou na chamada de uma função para passar dados (objetos) através dos parâmetros. Isso configura uma boa prática.

Questões e Prática

Elabore uma função para somar, subtrair, multiplicar e dividir dois números informados pelo usuário. A função deve imprimir todos os resultados.

Uma possível solução:

```
calcula <- function(a = 0, b = 1) {  
  # Imprime operações matemáticas básicas  
  print(a + b)  
  print(a - b)  
  print(a * b)  
  print(a / b)  
}
```


Fundamentos do R – funções

Alguma recomendação para nomes de funções? Procure usar verbos para nomear funções.

Além disso:

- [The tidyverse style guide](#)

```
add_row()  
permute()
```

- [Google's R Style Guide](#)

```
DoNothing()
```

Comentários em Código

O símbolo # (hashtag) permite adicionar comentários no código.
Tudo que estiver à direita do mesmo será ignorado.

- Comentários podem ser usados em funções e *sripts*.
- Documentam o código.
- Favorecem o entendimento para os leitores, inclusive, para aqueles que não conhecem à linguagem.
- São particularmente úteis no processo de codificação.

Comentários em Código

- **Recomendações para o uso de comentários?**
 - Use para explicar o que uma ou mais linhas de código fazem.
 - Evite usar para explicar o como é feito.
 - Tente economizar no uso de comentários.
 - Tente usar explicações simples e objetivas.
- **The tidyverse style guide** diverge da primeira recomendação da lista acima. Segundo o mesmo o código deve ser claro o suficiente para representar o que está sendo feito. Neste caso, comentários devem documentar descobertas e decisões tomadas em um processo de análise de dados.

Questões e Prática

Elabore uma função que calcule a hipotenusa de um triângulo retângulo conforme o Teorema de Pitágoras: $c^2 = a^2 + b^2$.

Abaixo segue um protótipo com os argumentos necessários para a função:

```
calcula_hipotenusa <- function(a, b) {}
```

Uma possível solução:

```
calcula_hipotenusa <- function(a, b) {  
  return(sqrt(a^2 + b^2))  
}  
calcula_hipotenusa(3, 4)
```

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

Questões e Prática

Observe o seguinte exemplo de função:

```
calcula_hipotenusa <- function(a, b) {  
  if(!is.numeric(a)) {  
    stop('"a" must be numeric\n',  
        'You have provided an object of class: ', class(a))  
  }  
  sqrt(a^2 + b^2)  
}
```

```
calcula_hipotenusa <- function(a, b) {  
  #if(!is.numeric(a)) {  
  #  stop('"a" must be numeric\n',  
  #      'You have provided an object of class: ', class(a))  
  #}  
  sqrt(a^2 + b^2)  
}  
calcula_hipotenusa(3, 4)
```

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

Desafio

Considerando o exemplo de função anterior. Que melhorias seriam possíveis para incrementar a proteção contra entradas erradas inseridas por usuários? Tente codificar essas melhorias.

Tarefa para a semana

📌 A tarefa desta semana vai ser disponibilizada no mural do classroom. Trata-se de um quiz com questões e exercícios práticos. Quando a tarefa for liberada você receberá um email de aviso.

Metotologias Informacionais com 

Muito Obrigado pela Atenção!