Aula 1 - Introdução ao R

Table of contents

1 Aula 1

Dr. Gabriel Bertolini

2º semestre 2023 - PPGGEO UFRGS

Oficina de programação

Aula de introdução ao R, incluindo variaveis, objetos, funções.

1.1 R

Linguagem de programação colaborativa - CRAN

Orientada ao objeto e programação funcional com comando pype native e tidyverse

Bibliotecas podem ser baixadas diretamente via console com install.packages("nome_do_pacote"), ou direto do Github com a função remotes::

1.2 Tidyverse

Estilo de programação funcional baseado na encadeação de funções através do operador %>% com funções uteis para limpeza, organização, modelamento e plotagem de dados- através de varios pacotes especificios.

- **dplyr -** estabelece o %>% para prog. funcional , e funções base para manipulação de banco de dados, como criação e manipulação de colunas, seleção de colunas, filtragem entre outros.
- **ggplot -** gráficos, mapas
- **purr -** funções para iteração de função (map())
- **tidyr -** funções para manipular banco de dados, como nest(), pivot_longer e _wider()
- **stringr-** funções para dados string (character), para limpeza, tokenização e outros.

1.3 Bibliografia

R for Data Science (2e) - https://r4ds.hadley.nz/ ou em espanhol (https://es.r4ds.hadley.nz/) na 1 edição

Geocomputation with R - https://r.geocompx.org/ ou em espanhol em https://r.geocompx.org/es/

The R Graph Gallery - https://r-graph-gallery.com/

Stack overflow - https://stackoverflow.com/

Canais de Youtube

R ladies e Beatriz Milz (https://beamilz.com/talks/)

Julia Silge https://www.youtube.com/@JuliaSilge

Geostats guy (Python e R para geoestatistica) https://www.youtube.com/@GeostatsGuyLectures

Posit - videos sobre pacotes tidyverse - https://www.youtube.com/@PositPBC/videos

1.4 Operações matemáticas

```
5 + 2 #soma
```

```
[1] 7
```

```
10 - 8 #subtração
```

```
[1] 2
```

```
8.2 / 2 #divisão
```

```
[1] 4.1
```

```
5 * 2 #multiplicação
```

```
[1] 10
```

```
5 ** 3 #elevado ao cubo
```

```
[1] 125
```

```
5 ^ 3 #elevado ao cubo
```

```
[1] 125
```

```
6.02 * 10^-23 # Número de Avogrado
```

```
[1] 6.02e-23
```

```
6.02e-23 # Número de Avogrado (mais simples)
```

```
[1] 6.02e-23
```

```r
sqrt(81) #raiz quadrada
```

```
[1] 9
```

```r
exp(8) # Exponencial
```

```
[1] 2980.958
```

```r
log(5) #log natural
```

```
[1] 1.609438
```

```r
log10(5) #log base 10
```

```
[1] 0.69897
```

```r
pi # pi
```

```
[1] 3.141593
```

```r
5 + (3 * 5 / 2) #operações mais elaboradas
```

```
[1] 12.5
```

1.5 Criar variavéis

```r
A<-1
# Criar variavel

A=1
# Criar variavel

myNumber<-5
```

```r
#Camel case - prática de escrever palavras ou frase, onde cada palavra é iniciada com maiú

myNumber = 5

myNumber
```

```
[1] 5
```

```r
myNumber <- 7.1

calculo <- 5 + (3 * 5 / 2)

myNumber * calculo
```

```
[1] 88.75
```

Tipos e atribuição de objetos

- numeric ou double - decimal
- integer - inteiro
- factor - categórica
- "character" - texto
- boleano - binário

```r
myInteger <- as.integer(myNumber)
# transformar objeto em inteiro

myInteger <- is.integer(myNumber)
# perguntar se o objeto é inteiro

myInteger
```

```
[1] FALSE
```

```r
myWord = "7"

#erro
```

```r
#myWord * 2

myWord <- "geologia"
#string- texto puro

myWord
```

```
[1] "geologia"
```

```r
is.character(myWord)
```

```
[1] TRUE
```

```r
mySentence <- "esta é a minha frase"

mySentence
```

```
[1] "esta é a minha frase"
```

```r
myFactor <- factor(myWord)
#valor categórica, com vários níveis

myFactor
```

```
[1] geologia
Levels: geologia
```

```r
is.factor(myFactor)
```

```
[1] TRUE
```

```r
#o R também trabalha com classes lógicas, TRUE or FALSE

#variável boleana (binária, lógica)
```

```r
myTRUE <- TRUE

4 == 4
```

```
[1] TRUE
```

```r
#4 é igual a 4?

a<-4

a==4
```

```
[1] TRUE
```

1.6 Testando tipo de variável

```r
class(myNumber)
```

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

```r
#variável numérica - qualquer número DECIMAL

class(myInteger)
```

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

```r
#variável numérica - número INTEIRO

class(myWord)
```

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

```r
#variável caracter

class(myFactor)
```

```
[1] "factor"
```

```
#fator
class(mySentence)
```

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

```
class(myTRUE)
```

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

```
#variável lógica
```

1.7 Operadores lógicos

```
1>0
```

```
[1] TRUE
```

```
# Maior
2>=2
```

```
[1] TRUE
```

```
#Maior igual
1<2
```

```
[1] TRUE
```

```
# Menor
1<=2
```

```
[1] TRUE
```

```r
# Menor igual

2==2
```

```
[1] TRUE
```

```r
# igual

1!=2
```

```
[1] TRUE
```

```r
# diferente

A<-data.frame(2,3,2,3)
A$X2
```

```
[1] 2
```

```r
# Indexador de coluna

ifelse(2>1|1>3,"CERTO","ERRADO")
```

```
[1] "CERTO"
```

```r
# | significa OU

ifelse(2<1 &1>3,"CERTO","ERRADO")
```

```
[1] "ERRADO"
```

```r
# & E

A<-c("A","B","C","D","E")
c("A","B") %in% A
```

```
[1] TRUE TRUE
```

```
# dentro de um vetor
```

2 Objetos

2.1 Vetor

Estrutura mais simples de dados, consiste uma série de valores concatenados de 1 a n. Estes valores podem ser quaisquer tipo de variaveis com double, character e bolean.

```
c("valores","quaisquer",999,pi,TRUE,1e-01)
```

```
[1] "valores"          "quaisquer"        "999"              "3.14159265358979"
[5] "TRUE"             "0.1"
```

```
# Função c() serve para concatenar

myNumbers <- c(1, 2, 3, 4, 5)
#vetor - forma de concatenar uma serie de dados

myNumbers2 <- 1:5

myNumbers3 <- seq(from=1,to=10,by=0.5)

myNumbers3[3]
```

```
[1] 2
```

```
myNumbers3[myNumbers3 > 3]
```

```
 [1]  3.5  4.0  4.5  5.0  5.5  6.0  6.5  7.0  7.5  8.0  8.5  9.0  9.5 10.0
```

```
myNumbers[myNumbers >= 3]
```

```
[1] 3 4 5
```

```r
myNumbers[myNumbers!= 3]
```

```
[1] 1 2 4 5
```

```r
myNumbers3[myNumbers3 > 3]
```

```
 [1]  3.5  4.0  4.5  5.0  5.5  6.0  6.5  7.0  7.5  8.0  8.5  9.0  9.5 10.0
```

```r
myWords = c("geoquímica", "estrutural", "estratigrafia")
#strings - fator

myWords
```

```
[1] "geoquímica"    "estrutural"    "estratigrafia"
```

```r
myWords[3]
```

```
[1] "estratigrafia"
```

```r
myWords[4]
```

```
[1] NA
```

2.1.1 Operação com vetores

```r
meuVetor <- seq(85, 274, by = pi)

#quanto menor o desvio padrão, mais homogênea é a amostra

length(meuVetor)
```

```
[1] 61
```

```r
(n <- length(meuVetor))
```

```
[1] 61
```

```r
dp <- sd(meuVetor)
#indica o grau de variação de um conjunto de elementos

SE <- dp / sqrt(n)

SE
```

```
[1] 7.140935
```

```r
#medida de variação de uma média amostral
#em relação à média da população
```

2.2 Matriz

Forma mais simples de dado bidemensional (colunas e linhas). As colunas e linhas são identificadas por sua posição.

```r
myMatrix <- matrix(1:6,nrow=3)

myMatrix
```

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

```r
y <- seq(1, 7, by = 1)

myMatrix2 <- matrix(y, nrow = 3, ncol = 2)
```

```
Warning in matrix(y, nrow = 3, ncol = 2): comprimento dos dados [7] não é um
submúltiplo ou múltiplo do número de linhas [3]
```

```
myMatrix2
```

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

```
# Juntando vetores para formar matriz

x <- 1:4

y <- 10:13

cbind (x,y)
```

```
     x  y
[1,] 1 10
[2,] 2 11
[3,] 3 12
[4,] 4 13
```

```
#cbind = Column Bind

#as colunas devem ter o mesmo número de linhas

rbind(x, y)
```

```
  [,1] [,2] [,3] [,4]
x    1    2    3    4
y   10   11   12   13
```

```
#rbind = Row Bind

matriz <- cbind(x, y)

matriz
```

```
     x  y
[1,] 1 10
[2,] 2 11
[3,] 3 12
[4,] 4 13
```

```
#matrix matemática

#indexação

myMatrix[,1]
```

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

```
#coluna 1

myMatrix[1,]
```

```
[1] 1 4
```

```
#linha 1

myMatrix[2,2]
```

```
[1] 5
```

```
# Linhas 2 e coluna 2
```

2.3 Dataframe

```
df <- as.data.frame(matriz)

df
```

```
  x  y
1 1 10
2 2 11
3 3 12
4 4 13
```

```r
#banco de dados

class(matriz)
```

```
[1] "matrix" "array"
```

```r
class(df)
```

```
[1] "data.frame"
```

```r
str(df)
```

```
'data.frame':   4 obs. of  2 variables:
 $ x: int  1 2 3 4
 $ y: int  10 11 12 13
```

```r
# Estruturas

length(df)
```

```
[1] 2
```

```r
# Indexação

df$y
```

```
[1] 10 11 12 13
```

```r
length(df$y)
```

```
[1] 4
```

```r
x <- 1:4

y <- 10:13

DF2<-as.data.frame(cbind (x,y))

DF2$y
```

```
[1] 10 11 12 13
```

2.4 Lista

```r
#matrizes empilhadas
x <- 1:4

y <- 10:13

z <- 2:7

myList <- list(x, y, z)

myList[[2]]
```

```
[1] 10 11 12 13
```

```r
myList[[3]][2]
```

```
[1] 3
```

```r
myLists <- list(matriz,z)

# Indexação
myLists[[1]][3,2]
```

```
 y
12
```

3 Funções

Maneira útil para automatizar processo.

Objetivo: função para verificar se um dado valor é maior do que outro

```
maior <- function(parametro1, parametro2) {

if (parametro1 < parametro2)

{return(parametro2) }

else {return(parametro1)}

}

soma<-function(A,B){

A+B

}

a = 4

b = 10

soma(a,b)
```

```
[1] 14
```

```
maior(soma(a^2,b-2),39)
```

```
[1] 39
```

```
maior(parametro1 = 4, parametro2 = 10)
```

```
[1] 10
```

```r
maior(parametro1 = b, parametro2 = a)
```

```
[1] 10
```

```r
maior(b, a)
```

```
[1] 10
```

4 Exercícios

1. Vetores e operações vetoriais

   a) Criar vetores de diferentes tipos de variavéis

```r
A<-c(1,2,3,4,5)
B<-c(TRUE,FALSE,FALSE,TRUE,TRUE)
C<-seq(from=1,to=10,by=0.5)
D<-c("A","B","C","D","B")
```

   b) Filtre por alguma variável;

```r
A[A>2]
```

```
[1] 3 4 5
```

```r
B[B==TRUE]
```

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

```r
C[A>2.5 & A<7.5]
```

```
 [1]  2.0  2.5  3.0  4.5  5.0  5.5  7.0  7.5  8.0  9.5 10.0
```

```r
D[D=="B"]
```

```
[1] "B" "B"
```

2. Crie uma dataframe com:

   a) 4 colunas: numerica, inteiros, caractere e booleana

```r
# R base
data.frame(A[1:5],B[1:5],C[1:5],D[1:5])
```

```
  A.1.5. B.1.5. C.1.5. D.1.5.
1      1   TRUE    1.0      A
2      2  FALSE    1.5      B
3      3  FALSE    2.0      C
4      4   TRUE    2.5      D
5      5   TRUE    3.0      B
```

```r
# Tidy
DF<-tibble::tibble(A=A[1:5],
                   B=B[1:5],
                   C=C[1:5],
                   D=D[1:5])
```

b) Filtre por alguma variável;

```r
DF[1:3]# Cols
```

```
# A tibble: 5 x 3
      A B         C
  <dbl> <lgl> <dbl>
1     1 TRUE    1
2     2 FALSE   1.5
3     3 FALSE   2
4     4 TRUE    2.5
5     5 TRUE    3
```

```r
DF[,1:3]# Cols
```

```
# A tibble: 5 x 3
      A B         C
  <dbl> <lgl> <dbl>
1     1 TRUE    1
2     2 FALSE   1.5
3     3 FALSE   2
4     4 TRUE    2.5
5     5 TRUE    3
```

```r
DF[1:3,]# Linhas
```

```
# A tibble: 3 x 4
      A B         C D
  <dbl> <lgl> <dbl> <chr>
1     1 TRUE    1   A
2     2 FALSE   1.5 B
3     3 FALSE   2   C
```

```
DF[1:3,1:3]# Cols & Linhas
```

```
# A tibble: 3 x 3
      A B         C
  <dbl> <lgl> <dbl>
1     1 TRUE    1
2     2 FALSE   1.5
3     3 FALSE   2
```

```
DF[DF$D=="B",]
```

```
# A tibble: 2 x 4
      A B         C D
  <dbl> <lgl> <dbl> <chr>
1     2 FALSE   1.5 B
2     5 TRUE    3   B
```

```
DF[DF$D %in% c("B","D"),]
```

```
# A tibble: 3 x 4
      A B         C D
  <dbl> <lgl> <dbl> <chr>
1     2 FALSE   1.5 B
2     4 TRUE    2.5 D
3     5 TRUE    3   B
```

```
DF[DF$C>=2,]
```

```
# A tibble: 3 x 4
      A B         C D
  <dbl> <lgl> <dbl> <chr>
1     3 FALSE   2   C
2     4 TRUE    2.5 D
3     5 TRUE    3   B
```

3. Crie uma função com a seguinte expressão matemática $X^2+2Y+17$

```
funcao_quad<-function(X,Y){
  return(
    (X^2)+2*Y+17
  )
}

funcao_quad(1,1)
```

```
[1] 20
```

```
funcao_quad(2,2)
```

```
[1] 25
```

4. Crie função para calcular erro padrão:

$$SE = \frac{\sigma \times CI}{N}$$

onde:

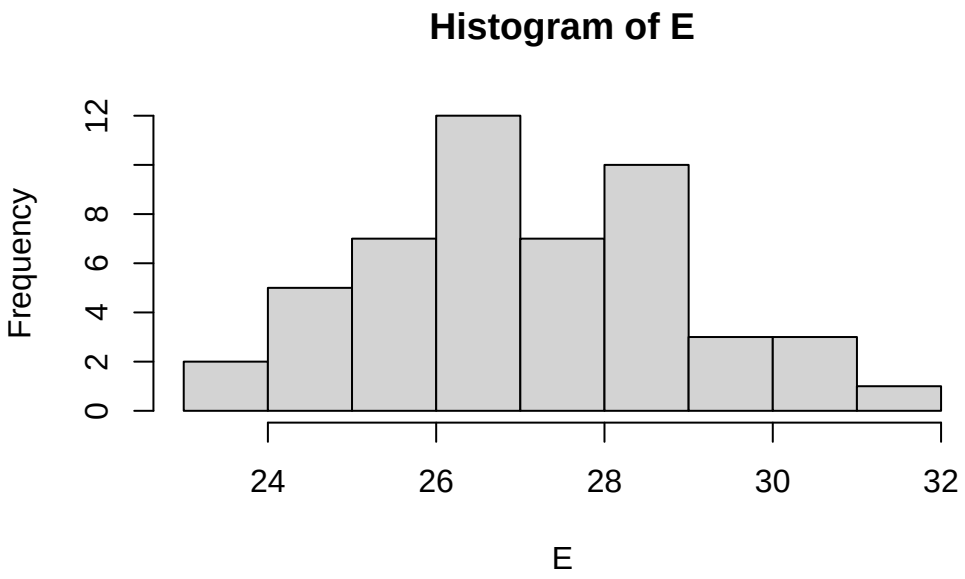
- Sigma ( ) = desvio padrão
- CI = intervalo de confiança (1 =68.3 % ,2 = 95.4 % ,3 =99.7 %,4 =99.994% ….)
- N = Número de amostras

```
SE<-function(vetor,IC){
  SD<-sd(vetor)
  N<-length(vetor)
  SE<-(SD*IC)/sqrt(N)

   return(
    SE
  )
}
set.seed(123)

E<-rnorm (27.1,sd=2,n=50)
```

```r
hist(E)
```




```r
mean(E)
```

```
[1] 27.16881
```

```r
SE(E,2)
```

```
[1] 0.5237512
```

```r
# 27.16881 ± 0.5237512
```

5 Atalhos

- Alt + - (hífen): cria o símbolo de atribuição "<-" no seu script.
- Ctrl + Shift + M: cria o operador %>% (pipe) do pacote dplyr no script.

- Ctrl + Shift + C: Comenta ou descomenta a linha de código ou a seleção atual.
- Ctrl + Shift + K: Compilar em PDF no Markdown.
- Ctrl + Shift + H: opção para alterar o diretório de trabalho
- Ctrl + Shift + R: Insere uma nova seção de código.
- Ctrl + Shift + N: abre um novo script
- Ctrl + Alt + R: executa o código inteiro
- Ctrl + Alt + E: executa o código a partir da linha atual
- Ctrl + Alt + P: executa o próximo chunk
- Ctrl + L: limpar o console
- Ctrl + F: localizar e substituir
- Esc: interrompe o comando atualmente em execução.