Treinamento - Introdução ao R

Operações básicas

Vamos então à linguagem!

O R pode funcionar como uma simples **calculadora**, que utiliza a mesma sintaxe que outros programas (como o excel):

```
1+1.3                 #Decimal definido com "."
2*3
2^3
4/2

sqrt(4)               #raíz quadrada
log(100, base = 10)   #logarítmo na base 10
log(100)              #logarítmo com base neperiana
```

Agora, utilize as operações básicas para solucionar expressão abaixo. Lembre-se de utilizar parênteses `()` para estabelecer prioridades nas operações.

$(\frac{13+2+1.5}{3}) + log_4 96$

Resultado esperado:

```
## [1] 8.792481
```

Repare que, se posicionar o parênteses de forma incorreta, o código não resultará em nenhuma mensagem de erro, pois este é um erro que chamamos de **erro lógico**, ou seja, o código roda, mas não faz o que você gostaria que ele fizesse. Esse é o tipo de erro mais difícil de ser consertado. Os erros que produzem uma mensagem, seja um aviso (**warning**) ou um erro (**error**) são chamados de **erros de sintaxe**. Nesses casos, o R retornará uma mensagem para te ajudar a corrigí-los. Os **warnings** não comprometem o funcionamento do código, mas chamam a atenção para algum ponto, já os **errors** precisam necessariamente ser corrigidos para que o código rode.

Exemplo de error:

```
((13+2+1,5)/3) + log(96, base = 4)
```

Você pode também esquecer de fechar algum parênteses, ou aspas, ou colchetes, ou chaves, nesses casos, o R ficará aguardando o comando para fechar o bloco de código sinalizando com um `+`:

```
((13+2+1.5)/3 + log(96, base = 4)
```

Se acontecer, vá até o console e aperte ESC, que o bloco será finalizado para que você possa corrigí-lo.

Os comandos `log` e `sqrt` são duas de muitas outras funções básicas que o R possui. Funções são conjuntos de instruções organizadas para realizar uma tarefa. Para todas elas, o R possui uma descrição para auxiliar no seu uso, para acessar essa ajuda use:

```
?log
```

E será aberta a descrição da função na janela `Help` do RStudio.

Se a descrição do próprio R não for suficiente para você entender como funciona a função, busque no google (de preferência em inglês). Existem diversos sites e fóruns com informações didáticas das funções do R.

Operações com vetores

Os vetores são as estruturas mais simples trabalhadas no R. Construímos um vetor com uma sequencia numérica usando:

```
c(1,3,2,5,2)
```

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

MUITA ATENÇÃO: O *c* é a função do R (*Combine Values into a Vector or List*) com a qual construímos um vetor!

Utilizamos o simbolo `:` para criar sequências de números inteiros, como:

```
1:10
```

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

Podemos utilizar outras funções para gerar sequências, como:

```
seq(from=0, to=100, by=5)
```

```
##  [1]   0   5  10  15  20  25  30  35  40  45  50  55  60  65  70  75  80  85  90
## [20]  95 100
```

```
# ou
seq(0,100,5) # Se você já souber a ordem dos argumentos da função
```

```
##  [1]   0   5  10  15  20  25  30  35  40  45  50  55  60  65  70  75  80  85  90
## [20]  95 100
```

- Crie uma sequencia utilizando a função `seq` que varie de 4 a 30, com intervalos de 3 em 3.

```
## [1]  4  7 10 13 16 19 22 25 28
```

A função `rep` gera sequências com números repetidos:

```
rep(3:5, 2)
```

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

Podemos realizar operações utilizando esses vetores:

```
c(1,4,3,2)*2
c(4,2,1,5)+c(5,2,6,1)
c(4,2,1,5)*c(5,2,6,1)
```

Repare que já esta ficando cansativo digitar os mesmos números repetidamente, vamos resolver isso criando **objetos** para armazenar nossos vetores e muito mais.

Criando objetos

O armazenamento de informações em objetos e a possível manipulação desses faz do R uma linguagem *orientada por objetos*. Para criar um objeto basta atribuir valores para as variáveis, como a seguir:

```
x = c(30.1,30.4,40,30.2,30.6,40.1)
# ou
x <- c(30.1,30.4,40,30.2,30.6,40.1)
```

```
y = c(0.26,0.3,0.36,0.24,0.27,0.35)
```

Os mais antigos costumam usar o sinal `<-`, mas tem a mesma função de `=`. Há quem prefira usar o `<-` como atribuição em objetos e `=` apenas para definir os argumentos dentro de funções. Organize-se da forma como preferir.

Para acessar os valores dentro do objeto basta:

```
x
```

```
## [1] 30.1 30.4 40.0 30.2 30.6 40.1
```

A linguagem é sensível à letras maiúsculas e minúsculas, portanto `x` é diferente de `X`:

```
X
```

O objeto `X` não foi criado.

O nome dos objetos é uma escolha pessoal, a sugestão é tentar manter um padrão para melhor organização. Alguns nomes não podem ser usados por estabelecerem papéis fixos no R, são eles:

- NA - Not available, simboliza dados faltantes
- NaN - Not a number, simboliza indefinições matemáticas
- Inf - Infinite, conceito matemático
- NULL - Null object, simboliza ausência de informação

Podemos então realizar as operações com o objeto criado:

Para realizar a operação o R alinha os dois vetores e realiza a operação elemento à elemento. Observe:

```
x + y
```

```
## [1] 30.36 30.70 40.36 30.44 30.87 40.45
```

```
x*y
```

```
## [1]  7.826  9.120 14.400  7.248  8.262 14.035
```

Se os vetores tiverem tamanhos diferentes, ele irá repetir o menor para realizar a operação elemento à elemento com todos do maior.

```
x*2
x*c(1,2)
```

Se caso o menor vetor não for múltiplo do maior, obteremos um aviso:

```
x*c(1,2,3,4)
```

```
## Warning in x * c(1, 2, 3, 4): longer object length is not a multiple of shorter
## object length
```

```
## [1]  30.1  60.8 120.0 120.8  30.6  80.2
```

Repare que o **warning** não compromente o funcionamento do código, ele só dá uma dica de que algo pode não estar da forma como você gostaria.

Podemos também armazenar a operação em outro objeto:

```
z <- (x+y)/2
z
```

Podemos também aplicar algumas funções, como exemplo:

```
sum(z)  # soma dos valores de z
```

```
## [1] 101.59
```

```r
mean(z) # média
```

```
## [1] 16.93167
```

```r
var(z)  # variância
```

```
## [1] 6.427507
```

Indexação

Acessamos somente o 3º valor do vetor criado com `[]`:

```r
z[3]
```

Também podemos acessar o número da posição 2 a 4 com:

```r
z[2:4]
```

```
## [1] 15.35 20.18 15.22
```

Para obter informações do vetor criado utilize:

```r
str(z)
```

```
##  num [1:6] 15.2 15.3 20.2 15.2 15.4 ...
```

A função `str` nos diz sobre a estrutura do vetor, que se trata de um vetor **numérico** com 6 elementos.

Os vetores também podem receber outras categorias como **caracteres**:

```r
clone <- c("GRA02", "URO01", "URO03", "GRA02", "GRA01", "URO01")
```

Outra classe são os **fatores**, esses podem ser um pouco complexos de lidar.

De forma geral, fatores são valores categorizados por `levels`, como exemplo, se transformarmos nosso vetor de caracteres `clone` em fator, serão atribuidos níveis para cada uma das palavras:

```r
clone_fator <- as.factor(clone)
str(clone_fator)
```

```
##  Factor w/ 4 levels "GRA01","GRA02",..: 2 3 4 2 1 3
```

```r
levels(clone_fator)
```

```
## [1] "GRA01" "GRA02" "URO01" "URO03"
```

Dessa forma, teremos apenas 4 níveis para um vetor com 6 elementos, já que as palavras "GRA02" e "URO01" se repetem. Podemos obter o número de elementos do vetor ou o seu comprimento com:

```r
length(clone_fator)
```

```
## [1] 6
```

Também há vetores **lógicos**, que recebem valores de verdadeiro ou falso:

```r
logico <- x > 40
logico   # Os elementos são maiores que 40?
```

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

Com ele podemos, por exemplo, identificar quais são as posições dos elementos maiores que 40:

```r
which(logico)  # Obtendo as posiçoes dos elementos TRUE
```

```
## [1] 6
```

```r
x[which(logico)] # Obtendo os números maiores que 40 do vetor x pela posição
```

```
## [1] 40.1
```

```r
# ou
x[which(x > 40)]
```

```
## [1] 40.1
```

Também podemos localizar elementos específicos com:

```r
clone %in% c("URO03", "GRA02")
```

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

Também podem ser úteis as funções `any` e `all`. Procure sobre elas.

Encontre mais sobre outros operadores lógicos, como o `>` utilizado, neste link.

Warning1

Faça uma sequência numérica, contendo 10 valores inteiros, e salve em um objeto chamado "a".

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

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

Crie outra sequência, utilizando números decimais e qualquer operação matemática, de tal forma que seus valores sejam idênticos ao objeto "a".

```r
b <- seq(from = 0.1, to = 1, 0.1)
(b <- b*10)
```

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

Os dois vetores parecem iguais, não?

Então, utilizando um operador lógico, vamos verificar o objeto "b" é igual ao objeto "a".

```r
a==b
```

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

Alguns valores não são iguais. Como isso é possivel?

```r
a==round(b)
```

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

Warning2

Não é possível misturar diferentes classes dentro de um mesmo vetor, ao tentar fazer isso repare que o R irá tentar igualar para uma única classe:

```r
errado <- c(TRUE, "vish", 1)
errado
```

```
## [1] "TRUE" "vish" "1"
```

No caso, todos os elementos foram transformados em caracter.

Algumas Dicas:

- Cuidado com a prioridade das operações, na dúvida, sempre acrescente parenteses conforme seu interesse de prioridade.
- Lembre-se que, se esquecer de fechar algum `(` ou `[` ou `"`, o console do R ficará esperando você fechar indicando um `+`, nada será processado até que você digite diretamente no console um `)` ou aperte ESC.

- Cuidado para não sobrepor objetos já criados criando outros com o mesmo nome. Use, por exemplo: altura1, altura2.
- Mantenha no seu script .R somente os comandos que funcionaram e, de preferência, adicione comentários. Você pode, por exemplo, comentar dificuldades encontradas, para que você não cometa os mesmos erros mais tarde.

  Se estiver adiantada/o em relação aos colegas, você já pode fazer os exercícios da **Sessão 1**, se não, faça-os em outro momento e nos envie dúvidas pelo fórum.

Matrizes

As matrizes são outra classe de objetos muito utilizadas no R, com elas podemos realizar operações de maior escala de forma automatizada.

Por serem usadas em operações, normalmente armazenamos nelas elementos numéricos. Para criar uma matriz, determinamos uma sequência de números e indicamos o número de linhas e colunas da matriz:

```
X <- matrix(1:12, nrow = 6, ncol = 2)
X
```

```
##      [,1] [,2]
## [1,]    1    7
## [2,]    2    8
## [3,]    3    9
## [4,]    4   10
## [5,]    5   11
## [6,]    6   12
```

Podemos também utilizar sequências já armazenadas em vetores para gerar uma matriz, desde que eles sejam numéricos:

```
W <- matrix(c(x,y), nrow = 6, ncol =2)
W
```

```
##      [,1] [,2]
## [1,] 30.1 0.26
## [2,] 30.4 0.30
## [3,] 40.0 0.36
## [4,] 30.2 0.24
## [5,] 30.6 0.27
## [6,] 40.1 0.35
```

Com elas podemos realizar operações matriciais:

```
X*2
```

```
##      [,1] [,2]
## [1,]    2   14
## [2,]    4   16
## [3,]    6   18
## [4,]    8   20
## [5,]   10   22
## [6,]   12   24
```

```
X*X
```

```
##      [,1] [,2]
## [1,]    1   49
## [2,]    4   64
## [3,]    9   81
```

```
## [4,]   16  100
## [5,]   25  121
## [6,]   36  144
```

```
X%*%t(X)          # Multiplicação matricial
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]   50   58   66   74   82   90
## [2,]   58   68   78   88   98  108
## [3,]   66   78   90  102  114  126
## [4,]   74   88  102  116  130  144
## [5,]   82   98  114  130  146  162
## [6,]   90  108  126  144  162  180
```

Utilizar essas operações exige conhecimento de álgebra de matrizes, se quiser se aprofundar a respeito, o livro *Linear Models in Statistics, Rencher (2008)* possui um boa revisão à respeito. Você também pode explorar a sintaxe do R para essas operações neste link.

Acessamos os números internos à matriz dando as coordenadas [linha,coluna], como no exemplo:

```
W[4,2] # Número posicionado na linha 4 e coluna 2
```

```
## [1] 0.24
```