Introdução à Linguagem R

João Paulo Lazzarini Cyrino

10/09/2020

A Linguagem R

R é uma linguagem de programação desenvolvida por estatísticos para estatística. Foi desenvolvida no ano 2000 por Ross Ihaka e por Robert Gentleman, professores do Departamento de Estatística da Universidade de Auckland (Nova Zelândia).

É uma linguagem bastante poderosa para finalidades científicas e acabou por ser uma das mais utilizadas nos departamentos de várias áreas acadêmicas. Além disso, ela é bastante utilizada na indústria para tarefas relacionadas a estatística, ciência de dados e inteligência artificial.

Para as finalidades de nosso curso de Teoria da Gramática há dúvidas sobre qual linguagem seria melhor. Há outras possibilidades como Lisp e Haskell. No entanto, R parece ter algumas vantagens:

- R possui um ambiente de trabalho denominado RStudio, que inclui uma série de facilidades.
- A ferramenta RMarkdown permite criar documentos (como este), enquanto estamos programando. Ou seja, podemos escrever o artigo enquanto conduzimos o estudo.
- R possui muitas facilidades para trabalho com estatística e análise de dados, e isso pode ajudar muito na pesquisa.
- As funcionalidades matemáticas do R são excepcionais.

Se você estiver realmente interessado em linguística computacional, principalmente fora do escopo puramente acadêmico, recomendamos aprender Python também. Para as finalidades exclusivamente científicas, R será uma grande parceira e te ajudará imensamente com suas descobertas e teses/dissertações/artigos, etc.

Instalando R e RStudio

Atualmente é impossível dissociar R de seu ambiente de trabalho, a plataforma RStudio. Por essa razão, precisamos instalar os dois programas: a linguagem R e a plataforma RStudio.

Para baixar a linguagem R você pode clicar em um dos links abaixo, de acordo com seu sistema:

- Windows: https://cran.r-project.org/bin/windows/base/
- Mac OSX: https://cran.r-project.org/bin/macosx/

Assim que baixar, você deverá instalá-la antes de partir para baixar e instalar o RStudio. Uma vez instalada a linguagem R, você pode ir ao site https://rstudio.com/products/rstudio/download/#download e baixar a plataforma RStudio. Novamente, assim que baixar, instale a plataforma no seu computador.

A instalação tando da linguagem R como do RStudio devem ser fáceis e correrão sem maiores dificuldades. Em caso de problemas, você pode simplesmente digitar o problema no Google e certamente haverá uma solução para ele! Uma das grandes vantagens da linguagem R é sua **comunidade**: tanto os usuários como os mantenedores da linguagem são pessoas muito interessadas em ajudar, o que faz com que haja inúmeros recursos sobre a linguagem online.

Importante! Como aprender a programar?

Talvez seja sua primeira vez tentando aprender uma linguagem de programação. R é um tanto peculiar no sentido de ser mais objetiva e te levar a perder menos tempo com a criação de toda a infra-estrutura por trás de um programa. Ela nos permite dedicar somente aos cálculos e análises de nossos estudos. Essa facilidade não significa que será necessariamente fácil aprender a utilizar R e muito menos ter uma familiaridade total com a linguagem a ponto de se sentir plenamente confortável utilizando-a.

Programar é uma habilidade que se conquista com **muita** prática. Não adianta muito fazer cursos e apenas acompanhar o raciocínio do professor. É necessário tentar você mesmo e é necessário também querer explorar sozinho formas diferentes de resolver o problema.

Recomenda-se fortemente que, além de replicar mais de uma vez os códigos desse curso e estudá-los linha por linha, e além de fazer as atividades propostas, você também tente implementar esses conhecimentos em sua pesquisa atual. A forma como R funciona permite que você consiga integrar, com facilidade, o seu processo de pesquisa com o de programação. Por isso, gastar tempo praticando R a partir dos dados de sua pesquisa irá ser benéfico tanto para seu aprendizado de R como para sua pesquisa em si.

Outra coisa importante: acostume-se com não saber muita coisa e ter que pesquisar no google ou em documentos de referência como fazê-la. Nunca dominaremos totalmente os recursos todos da linguagem R, mas é importante aprender a aprender sobre eles. E também: **não se preocupe em memorizar todos os comandos, funções, etc. Isso pode ser bastante frustrante! Deixe que a prática se encarregue disso**.

Programando em R (o básico)

Aqui vamos cobrir alguns básicos de programação em R. Essa seção não é, nem de perto, um bom curso básico de R, mas é o suficiente para que você consiga acompanhar o que será feito nas próximas aulas. Ao final dela aponto alguns sites com mais cursos e recursos para você estender seu conhecimento.

Aprender uma linguagem de programação consiste, basicamente, em aprender:

1. Tipos e estruturas de dados: o que podemos calcular/computar
2. Operadores e Funções: os comandos que nos permitem realizar as operações com os dados
3. Estruturas de Controle: como controlar o fluxo do algoritmo

Aqui, no entanto, aprenderemos apenas os itens 1 e 2. Estruturas de controle em R são um tanto quanto desnecessárias a nível básico. No entanto, apresentaremos alguns elementos desse asssunto ao longo desse curso, conforme necessário. Mais importante que estruturas de controle é como visualizar gráficos, e isso está incluido brevemente aqui.

Tipos de Dados e Operações Básicas

Em R estamos, normalmente, manipulando ou números ou categorias. Categorias costumam ser descritas por texto, como por exemplo "Categoria A", ou "Singular", ou mesmo algo mais codificado como "sg.pl". Por essa razão, em R manipulamos dados de dois tipos, basicamente: `numeric` e `character`.

A função `class` nos permite descobrir qual o tipo de um dado. Veja só o que acontece quando digitamos os seguintes comandos no console do R: (aviso - tudo aquilo que aparece depois de # é um comentário e não precisa ser digitado no console. Você pode colocar comentários no seu código à vontade para tornar as coisas mais fáceis de entender, a linguagem simplesmente os ignora na hora de executar seu programa)

```
class(12) # descobrir em que tipo de dado o número 12 se enquadra
```

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

```
class("Abacate") # descobrir em que tipo de dado o texto "Abacate" se enquadra
```

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

Nota-se que dados do tipo `character` sempre são declarados utilizando aspas (podem ser duplas ou simples). Veja só o que acontece quando pedimos `class("12")` ao invés de `class(12)`:

```
class("12")
```

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

Podemos armazenar nossos dados em variáveis, o que torna nosso trabalho muito prático. Fazemos isso utilizando os operadores `<-` e `->`:

```
x <- 12 # x agora vale 12
12 -> x # um outro jeito de escrever que x agora vale 12
```

Nomes de variáveis se diferenciam de caracteres por não terem aspas. Para não confundir com números, elas também não podem começar por números. Sugerimos utilizar nomes de variáveis que sejam fáceis de entender. Quando digitamos o nome da variável no console, ele nos retornará o seu valor:

```
x
```

```
## [1] 12
```

É interessante que no RStudio há uma janela (Environment) que mostra todas as variáveis criadas ao longo do trabalho e os seus respectivos valores. Isso ajuda a não se perder no fluxo de trabalho.

Podemos fazer operações com dados numéricos como fazemos em matemática. Temos os operadores `+`, para soma, `-`, para subtração, `*` para multiplicação e `/` para divisão. Além disso podemos elevar um número n a uma potência p com `n**p` ou `n^p`. Também podemos tomar a raiz quadrada de um número n com a função `sqrt(n)`. Alguns exemplos abaixo:

```
soma <- 23 + 7 # armazenando o resultado de 23 + 7 na variável soma.
# Vamos fazer isso com outras operações também
subtr <- 30 - 23
mult <- 2 * 56
div <- 30/8.4 # repare que o . é o separador decimal
quadrado <- 8^2
raiz <- sqrt(16)
```

Podemos criar um vetor de números que nos retornará os resultados das operações acima. Fazemos isso assim:

```
c(soma, subtr, mult, div, quadrado, raiz)
```

```
## [1]  30.000000   7.000000 112.000000   3.571429  64.000000   4.000000
```

Vetores são estruturas de dados de uma dimensão. Eles aceitam dados de apenas um tipo (ou dados numéricos, ou de caracteres). Podemos armazená-los em variáveis da mesma forma que os demais dados. Vejamos algumas operações interessantes com vetores:

```
# armazenando o vetor com as operações matemáticas na variável v
v <- c(soma,subtr,mult,div,quadrado,raiz)
v # retornará o vetor
```

```
## [1]  30.000000   7.000000 112.000000   3.571429  64.000000   4.000000
```

```
v[1] # retorna o primeiro elemento do vetor
```

```
## [1] 30
```

```
v[3] <- 12 # insere o número 12 no terceiro elemento do vetor
```

```
v[1:4] # retorna os elementos de 1 a 4 no vetor
```

```
## [1] 30.000000  7.000000 12.000000  3.571429
```

```r
length(v) # retorna o tamanho do vetor
```

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

```r
v[7] <- 7 # adiciona um novo elemento 7 no vetor
length(v) # novo tamanho do vetor
```

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

```r
v[length(v)+1] <- 8 # adiciona o número 8 ao final do vetor.
# Note como nos referimos à nova posição do vetor utilizando a função length() + 1.
# Encadear funções é muito comum em programação.
v
```

```
## [1] 30.000000  7.000000 12.000000  3.571429 64.000000  4.000000  7.000000
## [8]  8.000000
```

```r
# nomes para cada posição do vetor
names(v) <- c('soma', 'sub', '12', 'div', 'quadrado', 'raiz', '7', '8')
v # veja como aparecem os nomes
```

```
##      soma       sub        12       div  quadrado      raiz         7         8
## 30.000000  7.000000 12.000000  3.571429 64.000000  4.000000  7.000000  8.000000
```

```r
v['quadrado'] # pedindo o elemento do vetor pelo nome
```

```
## quadrado
##       64
```

Manipular vetores é algo frequente quando estamos lidando com dados. Por enquanto, mais do que decorar as operações, se preocupe em entender o que está acontecendo. É interessante que podemos aplicar operações a todos os elementos dos vetores:

```r
par <- c(2,4,6,8,10)  # vetor de números pares
par^2 # mostra os membros do vetor par elevados ao quadrado
```

```
## [1]   4  16  36  64 100
```

```r
par + 1 # adiciona 1 aos membros do vetor par
```

```
## [1]  3  5  7  9 11
```

É importante entender que as operações sobre vetores geram um novo vetor e não alteram o vetor antigo. Se quisermos alterar o vetor já existente, devemos sobrescrevê-lo em sua respectiva variável:

```r
par <- par * 2 # vetor par agora ficou armazenado com seus valores multiplicados por 2
par # exibir o novo conteúdo de par
```

```
## [1]  4  8 12 16 20
```

Podemos construir uma tabelinha com dois vetores de comprimento igual através da função `cbind`:

```r
par <- c(2,4,6,8,10,12,14) # vetor par
impar <- par - 1 # vetor impar subtraindo 1 de cada valor de par
tabela <- cbind(impar,par) # tabela com duas colunas
tabela
```

```
##      impar par
## [1,]     1   2
## [2,]     3   4
## [3,]     5   6
## [4,]     7   8
```

```
## [5,]    9  10
## [6,]   11  12
## [7,]   13  14
```

Essa tabela é também denominada `matrix` (matriz) em R. Note que cada coluna tem um nome emprestado da variável que formou essa coluna. Podemos alterar os nomes das colunas com a função `colnames`:

```
colnames(tabela) <- c('x', 'x+1') # novos nomes para as colunas
tabela
```

```
##       x x+1
## [1,]  1   2
## [2,]  3   4
## [3,]  5   6
## [4,]  7   8
## [5,]  9  10
## [6,] 11  12
## [7,] 13  14
```

Podemos obter valores da tabela com a notação `[linha,coluna]`. Veja abaixo:

```
tabela[3,2] # obter o valor da terceira linha, segunda coluna.
```

```
## x+1
##   6
```

```
tabela[4,'x'] # obter o valor da quarta linha, coluna entitulada 'x'
```

```
## x
## 7
```

```
tabela[,'x+1'] # obter todos os valores da coluna 'x+1'
```

```
## [1]  2  4  6  8 10 12 14
```

```
tabela[2,] # obter todos os valores da linha 2
```

```
##   x x+1
##   3   4
```

```
nrow(tabela) # número de linhas da tabela
```

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

```
ncol(tabela) # número de colunas da tabela
```

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

Um terceiro tipo de estrutura de dados, que reune as propriedades da matriz com algumas outras funcionalidades é o quadro de dados, ou `data.frame`. Trata-se, simplesmente, de uma tabela em que cada coluna pode ser de um tipo diferente de dado. Podemos converter nossa tabela em um `data.frame` da seguinte forma:

```
tabela <- data.frame(tabela) # converte tabela em data.frame e a armazena novamente em tabela
tabela
```

```
##    x x.1
## 1  1   2
## 2  3   4
## 3  5   6
## 4  7   8
## 5  9  10
## 6 11  12
```

```
## 7 13  14
```

O quadro de dados é o que normalmente manipulamos em R. Podemos utilizar as mesmas funções que utilizamos na matriz para obter dados dele. Também podemos chamar cada coluna com a sintaxe `nome_da_tabela$nome_da_coluna`:

```
tabela$x # pedindo a coluna x
```

```
## [1]  1  3  5  7  9 11 13
```

```
# perceba que R mudou o nome da coluna x+1 da matriz para x.1 para evitar conflitos
tabela$x.1
```

```
## [1]  2  4  6  8 10 12 14
```

Se quisermos criar uma nova coluna, podemos:

```
tabela$x.2 <- tabela$x + 2 # nova coluna x.2 somando 2 aos valores da coluna x
tabela # vamos ver como ficou a tabela agora
```

```
##    x x.1 x.2
## 1  1   2   3
## 2  3   4   5
## 3  5   6   7
## 4  7   8   9
## 5  9  10  11
## 6 11  12  13
## 7 13  14  15
```

Parece bastante coisa, né? Mas acredite: manipular tabelas, matrizes e vetores em R é muito mais prático que em muitas linguagens de programação! Fica até mesmo mais fácil do que em Excel, depois que pegamos a prática.

Por enquanto apenas vimos operações para manipular dados numéricos. Há operações para manipular texto também, mas essas serão introduzidas ao longo do curso, conforme necessário.

Tipo Lógico

Há um terceiro tipo importante de dado (há outros, mas eles derivam de alguma forma desses três), que é o tipo lógico. Esse tipo só permite os valores `TRUE` ou `FALSE`, correspondendo a verdadeiro e falso. Com isso podemos fazer operações lógicas em R, o que é de grande utilidade.

Uma dessas operações são as operações de igualdade, como `==` (igual a) e `!=` (diferente). Com elas podemos comparar valores.

```
1 != 2 # verifica se 1 é diferente de 2
```

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

```
1 == 2 # verifica se 1 é igual a 2
```

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

Podemos aplicar operações lógicas para valores de tabelas. Elas nos retornará um vetor ou matriz (conforme o caso) de valores lógicos:

```
tabela == 5 # uma matriz com TRUE onde havia 5 na tabela
```

```
##          x   x.1   x.2
## [1,] FALSE FALSE FALSE
## [2,] FALSE FALSE  TRUE
## [3,]  TRUE FALSE FALSE
```

```
## [4,] FALSE FALSE FALSE
## [5,] FALSE FALSE FALSE
## [6,] FALSE FALSE FALSE
## [7,] FALSE FALSE FALSE
```

curiosamente, matematicamente os valores de TRUE podem ser somados como se fossem 1 e os de FALSE como se fossem 0. Dessa forma, podemos somar os valores verdadeiros da matriz obtendo a contagem de números 5 na tabela. Isso pode ser feito com a função `sum`, que soma os valores de uma matriz ou vetor:

```
sum(tabela == 5) # conta quantas vezes 5 aparece na tabela
```

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

Da mesma forma, podemos utilizar a função de média `mean` para calcular a proporção de 5 na tabela.

```
mean(tabela == 5) # proporção de números 5 na tabela
```

```
## [1] 0.0952381
```

Além de `==` e `!=` temos os seguintes operadores lógicos:

- `&`: operador E
- `|`: operador OU
- `!`: operador NÃO

Podemos fazer operações lógicas com esses operadores, por exemplo:

```
TRUE & TRUE # Verdadeiro e Verdadeiro = Verdadeiro
```

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

```
!(TRUE & TRUE) # Negação da conjunção acima = Falso
```

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

Os operadores se aplicam a vetores também. Dessa forma, conseguimos criar tabelas verdade com bastante facilidade:

```
# Criar permutações para p e q:
p <- c(TRUE, TRUE, FALSE, FALSE)
q <- c(TRUE, FALSE, TRUE, FALSE)
# Vamos avaliar p E q:
p & q
```

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

```
# Agora vamos avaliar p OU q:
p | q
```

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

Concluindo

Apresentamos, aqui algumas operações básicas em R. Novamente, não se preocupe tanto em decorar, mas em tentar replicar e explorá-las (tentar fazer cálculos com outros valores, matrizes diferentes, etc). Essa prática é o que traz segurança no uso da ferramenta.

Nosso próximo encontro com R explorará *Teoria dos Conjuntos* e *Lógica*!