1 Fundamentos Computacionais em Estatística

Uma linguagem de programação define a sintaxe e semântica que devem ser utilizadas para escrever programas de computadores. Dessa forma, estudar uma linguagem de programação consiste em aprender técnicas para elaborar códigos que possam ser interpretados por computadores e que podem ajudar a resolver numericamente, problemas que não possuem uma solução analítica.

Existem muitas linguagens de programação, alguns exemplos são:

✓ `C`, Fortran, Java, Julia, Python, Octave, Scilab e `R`;

✓ Matlab, `Ox` e SAS são produtos, mas também tem uma linguagem embutida.

A utilização de uma linguagem específica depende da necessidade e do que se deseja programar. No contexto de Estatística, aprender uma linguagem de programação está direcionada na construção de novos métodos e análise de dados. Assim, a linguagem de programação que será utilizada na disciplina é `R`, pelas seguintes razões:

✓ é um *software* de código aberto e livre;

✓ direcionada para Estatística;

✓ tem uma comunidade diversificada de usuários.

`R` é uma linguagem de programação desenvolvida em 1995, originalmente, por Ross Ihaka e Robert Gentleman do Departamento de Estatística da Universidade de Auckland, Nova Zelândia. O nome `R` provém em partes das inicias dos criadores e também de um jogo figurado com a linguagem `S` (*Bell Laboratories*), pois `R` é uma implementação de código aberto dessa linguagem. Atualmente é mantido por uma comunidade de colaboradores voluntários que contribuem com código fonte da linguagem e com a expansão de funcionalidades por bibliotecas.

1.1 Básico de R

1.1.1 Como calculadora

`R` é uma linguagem de programação em que os códigos são executados quando terminamos de digitá-los no console e por padrão, cada linha de comando é representado pelo símbolo >, indicando que o `R` está pronto e aguardando um código. Dessa forma, podemos utilizar diretamente o console do `R` para uma variedade de cálculos numéricos, desde que sejam comandos curtos de uma linha.

Os símbolos usuais de expressões numéricas em `R` são: $+$ (adição), $-$ (subtração), $*$ (multiplicação), $/$ (divisão) e $\wedge$ (potenciação). Além disso, podemos controlar a ordem das operações utilizando parênteses ( ). Assim como na Matemática, o `R` também segue a mesma regra para executar operações:

$$\text{Parênteses} \rightarrow \text{Potênciação} \rightarrow \text{Multiplicação e divisão} \rightarrow \text{Adição e subtração.}$$

Quando digitamos um comando no console e pressionamos *enter*, o `R` executa, faz os cálculos que pedimos e imprime o resultado. Entretanto, se iniciamos um comando, mas não o concluímos, como por exemplo, para determinar $5 \times (4 + 2)$:

```
> 5*(4+
+ digitar o restante do código
```

O console mudará o símbolo > para um +, indicando que espera o término do código. Em situações como essa, na qual não lembramos o que foi esquecido, pressionamos *Esc* para cancelar o código e repetimos novamente o comando. Dessa forma, com o cursor colocado no console, podemos utilizar a seta para cima (↑) e a seta para baixo (↓) do teclado para percorrer pelos comandos executados anteriormente.

`R` também disponibiliza funções matemáticas usuais como as que são encontradas em uma calculadora. Na Tabela 1 são apresentadas algumas funções aritméticas e como são utilizadas na linguagem.

Tabela 1: Funções aritméticas

Função	Interpretação	Função	Interpretação
`sqrt( )`	raiz quadrada	`abs( )`	valor absoluto
`sin( )`	seno	`cos( )`	cosseno
`tan( )`	tangente	`exp( )`	exponencial
`log( )`	logaritmo natural	`log(x, base)`	logaritmo
`factorial( )`	fatorial	`max( )`	máximo valor
`min( )`	menor valor		

Por *default*, a saída do `R` mostra apenas 7 dígitos significativos, mas podemos alterar a exibição para `x` dígitos usando a sintaxe:

```
options(digits = x)
```

Entretanto, quando `R` imprime números grandes ou pequenos além do limite determinado de sete dígitos significativos, os números são exibidos com uma notação eletrônica científica. Na notação, qualquer número x pode ser expresso como xe^y, o que em notação científica representa exatamente $x \times 10^y$. A notação eletrônica serve para facilitar a leitura do usuário e os dígitos extras ainda são armazenados no `R`, mesmo que não sejam mostrados, por exemplo:

```
> 2342151012900
[1] 2.342151e+12
> 0.0000002533
[1] 2.533e-07
```

★ **Observação**:

► Conseguindo ajuda

O `R` possui uma extensa documentação que pode ser acessada com o função `?` ou `help( )`. Os arquivos de ajuda inclui alguns itens como:

- ✓ `Description`: fornece uma breve declaração sobre as operações realizadas;
- ✓ `Usage`: especifica a forma da função em termos de como ela deve ser transmitida ao console do `R`;
- ✓ `Arguments`: são fornecidos detalhes sobre o que cada argumento faz, bem como os possíveis valores que eles podem assumir;
- ✓ `References`: fornece citações relevantes para o comando ou a metodologia por trás da função;
- ✓ `Examples`: fornece código executável que você pode copiar e colar no console, demonstrando a função em ação.

► Rstudio

Para trechos mais longos de código e função, por conveniência, primeiro escrevemos os códigos em um editor e assim, executá-los no console somente quando terminamos. Na própria linguagem existe um editor de código `R` embutido, entretanto, um dos ambientes de desenvolvimento integrado (IDE) mais utilizados é o Rstudio. O Rstudio que é uma corporação responsável pelo IDE (de mesmo nome) para o `R`, disponível em edições comerciais e de código aberto para Windows, Mac e Linux. Ao abrir o Rstudio, haverá quatro quadrantes que representam o editor, o console, o environment e o output. Eles vêm nessa ordem e depois podem ser re-organizadas a critério do usuário. As funções dos principais painéis são:

- ✓ `Editor`: local onde os códigos são escritos;
- ✓ `Console`: local onde os códigos são compilados e as saídas são recebidas;

- ✓ `Environment`: painel com todos os objetos criados na sessão;
- ✓ `Files`: mostra os arquivos no diretório de trabalho;
- ✓ `Plots`: painel onde os gráficos são apresentados;
- ✓ `Help`: janela onde a documentação das funções são apresentadas;
- ✓ `History`: painel com um histórico dos comandos rodados.

Alguns atalhos úteis do Rstudio:

- ✓ `CTRL + enter`: roda as linhas selecionadas no editor;
- ✓ `ALT + -`: operador de atribuição.

1.1.2 Objetos

Objeto é uma referência a um local da memória do computador que possui um valor, podendo ser uma variável, função ou estrutura de dados. Em `R`, os tipos básicos de objetos do `R` são:

- ✓ vetores e listas;
- ✓ matrizes e arrays;
- ✓ data-frames;
- ✓ funções.

Os três primeiros tipos são objetos que armazenam dados e que diferem entre si na forma de armazenar e operar com os dados. O último (função) é um tipo de objeto especial que recebe algum "input" e produz um "output".

Os objetos permitem que os dados armazenados na memória possam ser acessados mais tarde com um determinado nome, sem a necessidade de escrever o cálculo novamente. Para atribuir um nome a um objeto, utilizamos o operador de atribuição <- (recebe). O operador = também pode ser utilizado, mas alguns usuários preferem evitar o uso do símbolo = para não confundir com igualdade matemática.

Por exemplo, consideremos o seguinte código:

```
> x <- 100
```

Este código indica que esta executando as seguintes tarefas:

i) criando um objeto, contendo o valor 10;

ii) vinculando o objeto a um nome, nesse caso `x`.

Para o valor atribuído ao objeto ser exibido na tela, basta digitar o nome do objeto ou podemos mostrar o resultado de uma tarefa colocando-a entre parênteses.

Além disso, todo objeto em `R` tem uma classe, que descreve a sua forma e como o mesmo será manipulado pelas diferentes funções, as cincos classes básicas de objetos, também chamados de objetos "atômicos" são:

- ✓ `character`;
- ✓ `numeric`;
- ✓ `integer`;
- ✓ `complex`;
- ✓ `logical`.

Para saber a classe de um objetivo, utilizamos a função `class( )`.

Em relação aos nomes dos objetos, eles devem ser compostos por letras e podem conter, mas nunca começar com números, pontos e *underline*. Além disso, o `R` faz distinção entre letras maiúsculas e minúsculas, por exemplo, `aa`, `Aa` e `AA` são interpretados como nomes de três objetos diferentes pela linguagem.

Por outro lado, evite utilizar nomes:

✓ com símbolos;

✓ que sejam de objetos do sistema (em geral funções, ou constantes tais como o número π) como, por exemplo: `c, q, s, t, C, D, F, I, T, diff, exp, log, mean, pi, range, rank, var`;

✓ com palavras reservadas, como `TRUE, NULL, if, function`, entre outros. Podemos ver a lista completa de nomes resevados, usando o comando `?Reserved`.

1.1.3 Vetor e matriz

► Vetores

Vetores são o tipo básico e mais simples de objeto para armazenar dados no `R`. Para criar vetores de comprimento maior do que um, podemos utilizar a função `c( )` (c de concatenar) separando os elementos por vírgula como mostra a sintaxe:

```
c(elemento1,elemento2,...,elementoN)
```

em que `elementoi` ($i = 1, \ldots, N$) são elementos de uma mesma classe.

Por outro lado, as funções `rep( )`, `seq( )` e `:` são usadas para facilitar a criação de vetores que tenham alguma lei de formação. Na linguagem, essas funções são utilizadas da seguinte forma:

✓ `rep(x, r)` - repetir o valor ou o vetor x em r vezes;

✓ `seq(from, to, by)` - criar uma sequência de números que começam em *from* e vão até *to* de *by* em it by. Para gerar um vetor com n números, espaçados igualmente entre os valores *from* e *to* podemos utilizar `length.out = n`;

✓ `a:b` - criar uma sequência de números inteiros de a até b.

★ **Observação**:

✓ Os colchetes `[ ]` são utilizados para selecionar qualquer número de elementos do vetor, enquanto que a função `length( )` é usada para determinar o total de elementos de um vetor;

✓ A função `rbind( )` agrupa vetores por linha e a `cbind( )` por coluna;

✓ Podemos fazer qualquer operação aritmética entre vetor e um escalar e, também, entre vetores. Porém, no caso de operações com vetores de diferentes comprimentos, o `R` ficará repetindo o vetor de menor comprimento até completar o vetor de maior comprimento. Esse comportamento é denominado de reciclagem.

► Matrizes

A ideia de matriz no `R` é similar a da matriz que conhecemos na matemática, uma vez que seus elementos são indexados pelo índice da linha e da coluna correspondente, como mostra a matriz A:

$$A_{n\times p} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1p} \\ a_{21} & a_{22} & \cdots & a_{2p} \\ \cdots & \cdots & \ddots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{np} \end{pmatrix}.$$

Para a linguagem, matrizes são vetores de uma mesma classe arranjados em duas dimensões. Dessa forma, uma matriz é criada a partir um vetor utilizando a função `matrix` que tem a seguinte expressão:

```
matrix(data, nrow = n, ncol = p, byrow = FALSE (ou TRUE))
```

em que `data` é a fonte dos dados, `nrow` é o número de linha, `ncol` é o número de colunas e `byrow` indica se

a matriz será preenchida por linhas ou colunas. Por "default", a matriz é preenchida por colunas (`byrow = FALSE`). Se considerarmos `data = 0`, temos uma matriz de zeros e se `data = 1`, vamos gerar uma matriz de uns;

Uma outra forma de criar matrizes é a partir da utilização das funções `cbind` (conecta vetores formando colunas de uma matriz com cada vetor) e `rbind` (conecta vetores formando linhas de uma matriz com cada vetor), cujas sintaxes são dada por:

```
cbind(vetor1, ..., vetorN)
rbind(vetor1, ..., vetorN)
```

Algumas funções para tralharmos com matrizes são apresentadas na Tabela 2.

Tabela 2: Funções com matrizes

Função	Interpretação	Função	Interpretação
`t( )`	matriz transposta	`diag(k)`	matriz identidade $k \times k$
`dim( )`	dimensão da matriz	`ncol( )`	número de colunas
`nrow( )`	número de linhas	`rownames( )`	nome associado a linha
`colnames( )`	nome associado a coluna		

★ **Observação**:

✓ As operações com matrizes seguem a mesma lógica dos vetores. Ao realizar qualquer operação com uma constante, a operação é feita para todos os elementos da matriz. Entretanto, se quisermos realizar a multiplicação matricial, devemos utilizar o operador $\%*\%$;

✓ Para selecionarmos um elemento em uma matriz, devemos informar entre colchetes (`[ , ]`) em qual linha e coluna se encontra o dado que queremos.

✓ Seja A uma matriz $n \times p$, então o código `A[i,]` seleciona a i-ésima linha da matriz A, enquanto que o código `A[,j]` seleciona a j-ésima coluna. Para selecionar mais de uma coluna ou linha utilizamos a função `c( )` para criar vetores;

✓ Para excluir ou omitir linhas ou colunas de uma matriz, utilizamos colchetes novamente, mas com índices negativos.

1.1.4 Valores não numéricos

► Valores lógicos

Em `R`, os valores lógicos (ou booleanos) são objetos com valor `TRUE` ou `FALSE`, frequentemente abreviados como `T` ou `F`, respectivamente e utilizados para indicar se uma condição foi satisfeita ou não. Por causa dessa natureza binária, o valor `FALSE` é armazenado como `0L` e `TRUE` como `1L`, o que permite que operações como adição, subtração, multiplicação e divisão possam ser realizadas.

Uma expressão lógica é formada utilizando os operadores de comparação apresentados na Tabela 3 e de lógica apresentados na Tabela 4.

Tabela 3: Operadores de comparação

Operador	Interpretação	Operador	Interpretação
==	Igual a	!=	Não é igual a
>	Maior do que	<	Menor do que
>=	Maior ou igual a	<=	Menor ou igual a

Tabela 4: Operadores lógicos

Operador	Interpretação	Operador	Interpretação
&	AND (vetor elemento a elemento)	&&	AND (escalar)
|	OR (vetor elemento a elemento)	||	OR (escalar)
!	NOT		

► Caracter

Caracter são variáveis de texto (ou `string`) muito comuns em banco de dados e frequentemente são utilizadas para especificar locais de pastas, fornecer um argumento para uma função, fornecer uma saída de texto, entre outras utilidades. As variáveis de texto são indicadas por aspas duplas (“ ”) ou simples (‘ ’). Para criar uma variável de texto, basta digitar o texto entre um par de aspas. Na Tabela 5 são apresentadas algumas funções para manipular textos.

Tabela 5: Algumas funções para `string`

Função	Interpretação
`toupper( )`	Converte para maiúsculo
`tolower( )`	Converte para minúsculo
`nchar( )`	Quantidade de caracteres incluindo o espaço
`sub(pattern = , replacement = , x = )`	Substitui conteúdo
`grep(pattern = ,x = )`	Pesquisa termos no vetor `string`
`grepl(pattern = ,x = )`	Retorna se um termo está na `string`

★ **Observação**:

✓ A função `paste( )` também pode ser utilizada para criar vetores de caracteres;

✓ Para fornecer uma saída de texto, podemos utilizar a função `cat( )`. Além disso, o código `\n` é utilizado para começar uma nova linha e `\t` para um espaço horizontal.

► Fator

Fatores podem ser vistos como vetores de inteiros que possuem rótulos (*levels*) e são criados a partir da função `factor`, como mostra a seguinte sintaxe:

```
factor(x)
```

em que `x` representa um vetor de texto. Os fatores são úteis para representar uma variável categorizada (nominal ou ordinal).

Em algumas situações, como por exemplo em caselas de referência de modelos estatísticos e nas barras de um gráfico, a ordem dos rótulos de um fator pode importar. A disposição dos rótulos segue a ordem alfabética a menos que uma ordenação diferente seja definida pelo usuário no argumento `levels` como em:

```
factor(x, levels = ordem)
```

em que `ordem` representa a ordem dos rótulos.

★ **Observação**: A função `levels( )` também pode ser utilizada para retorna os rótulos do fator.

1.1.5 Valores especiais

Existem ainda alguns valores especiais usados pelo `R`:

✓ `pi`: armazena o valor desta constante matemática;

✓ `NA` (Not Available): denota dado faltante/indisponível.

✓ `NaN` (Not a Number): denota indefinições matemáticas;

✓ `Inf` e `-Inf`: mais e menos infinito, respectivamente.

✓ `NULL`: representa a ausência de informação. Conceitualmente, a diferença entre `NA` e `NULL` é sutil, mas, no `R`, o `NA` está mais alinhado com os conceitos de estatística (ou como gostaríamos que os dados faltantes se comportassem em análise de dados) e o `NULL` está em sintonia com comportamentos de lógica de programação.

1.1.6 Data frame e lista

► Lista

Listas são um tipo especial de vetor que aceita elementos de classes diferentes em um único objeto. Uma lista é criada utilizando a função `list` com argumentos separados por vírgula, como mostra a sintaxe

```
list(componente1, componente2, ..., componenteN)
```

em que `componentei` ($i = 1, \ldots, N$) são vetores de diferentes comprimentos e classes.

Supondo que atribuímos o nome `my_lista` para um objeto do tipo lista, podemos recuperar um único elemento da lista utilizando índices inseridos entre colchetes duplos (`[[ ]][ ]`) de acordo com a seguinte sintaxe:

```
my_lista[[i]][j] ♯ para vetor com mais de um elemento
my_lista[[i]] ♯ para vetor com apenas um elemento
```

em que o índice i representa o i-ésimo componente da lista e j representa o j-ésimo elemento do vetor. Também podemos recuperar os elementos de um vetor dentro de uma lista usando índices inseridos entre colchetes simples (`[ ]`), como ilustra sintaxe:

```
my_lista[i]
```

em que o índice i representa o i-ésimo componente da lista.

Além disso, os componentes de uma lista podem ser nomeados quando a lista é criada, usando argumentos do formulário `nome1 = componente1`, `nome2 = componente2`, etc., ou podem ser nomeados posteriormente, atribuindo um valor ao atributo de nomes, como podemos observar, respectivamente:

```
list(nome1 = componente1, nome2 = componente2, ..., nomeN = componenteN)
♯ ou
names(my_lista) <- c('nome1', 'nome2', ..., 'nomeN')
```

Nesse caso, a seguinte sintaxe, mostra como podemos recuperar os elementos da lista utilizando o símbolo `$`:

```
my_lista$nomei
```

em `nomei` é o nome do i-ésimo componente da lista.

★ **Observação**: a função `length` pode ser utilizada para verificar o número de componentes de uma lista.

► Data frame

Os *data frame*'s são listas em que todos os componentes possuem o mesmo comprimento. Em síntese, *data frames* são tabelas de dados, em que cada componente dessa lista pode ser pensado como uma coluna da tabela. Para criar um *data frame*, utilizamos a função `as.data.frame`, como mostra a seguinte sintaxe:

```
as.data.frame(my_lista)
```

em que `my_lista` é uma lista criada a partir da função `list`. Em seu formato, os *data frame*'s são bem

parecidos com as matrizes, no entanto, cada coluna pode armazenar um tipo de classe diferente, ou seja, podemos ter colunas de valores numéricos e colunas de caracteres no mesmo objeto.

Uma forma de criar um *data frame* do zero consiste em utilizar a sintaxe dada por:

```
data.frame(nome1 = vetor1, ..., nomeN = vetorN)
```

Algumas funções úteis para trabalhar com *data frame*'s são apresentados na Tabela 6.

Tabela 6: Algumas funções para utilizar em *data frames*

Função	Interpretação	Função	Interpretação
`str( )`	Estrutura do objeto e as classes das colunas	`names( )`	Nomes das colunas
`head( )`	Mostra as primeiras 6 linhas	`tail( )`	Mostra as últimas 6 linhas
`ncol( )`	Número de colunas	`nrow( )`	Número de linhas

★ **Observação**: Para importar um *data frame*, podemos utilizar as seguintes funções:

✓ `read.table('nome.txt', header = TRUE, sep = ',')`, em que `header` interpreta o nome das colunas do conjunto de dados e `sep` identifica o separador de casas decimais;

✓ `read.csv('nome.csv', sep = ',')`;

✓ `read_excel('nome.xlsx')`. Nesse caso, o pacote `readxl` será necessário para importar o conjunto de dados.

1.1.7 Gráfico

Um recurso particularmente popular do `R` são suas ferramentas para construção de gráficos para visualização de dados e modelos, o que atrai muitos para R em primeiro lugar. A linguagem já vem com uma função básica que permite fazer gráficos de acordo com a seguinte sintaxe:

```
plot(x, y, type = ' ')
```

em que `x` são os valores da abscissa, `y` são os valores da ordenada e `type` é o tipo de gráfico: pontos (`p`), linhas (`l`), ambos (`b`), nenhum (`n`), entre outros. Alguns argumentos de gráficos são apresentados na Tabela 7.

Tabela 7: Alguns argumentos da função `plot`

Argumento	Interpretação
`xlab = ' '`	Título do eixo x
`ylab = ' '`	Título do eixo y
`col = ' '`	Cor do pontos
`pch =`	tipo de pontos do gráfico
`cex =`	Tamanho do ponto e textos
`lwd =`	Espessura da linha
`lty =`	Tipo de tracejado da linha

★ **Observação**:

✓ Algumas cores básicas são: `red`, `blue`, `green`, `magenta` e `yellow`;

✓ Alguns tipos de ponto: 8 (∗), 15 (■), 16 (●), 17 (▲) e 18 (♦);

✓ Gráficos estatísticos podem ser construídos por meio das funções: `hist` (histograma), `barplot` (gráfico de barras) e `boxplot` (box-plot).

1.2 Básico de Programação

Utilizar a linguagem R como uma calculadora pode ser útil por si só. Mas também pode nos ajudar a criar e testar fragmentos de código que desejamos incorporar em nossos programas em R, assim como a aprender sobre as novas funções R à medida que as conhecemos.

1.2.1 Boas práticas de programação

Algumas dicas para tornar um programa claro ser entendido para futuras correções e modificações:

✓ Inicie cada programa com alguns comentários, indicando o nome do programa, o autor, a data em que foi criado e o que o programa faz. Também utilize comentários para blocos grandes de códigos. No R, anteceda a linha com um símbolo de sustenido (♯), e tudo o que vier a seguir será um comentário, como mostra o exemplo a seguir:

```
> ♯ comentário em R
```

✓ Como os programas R são executados em um ambiente em que já pode haver variáveis definidas pelo usuário, geralmente é uma boa prática limpar o espaço de trabalho antes de executar um programa, para garantir o mesmo ponto de partida de cada vez. Assim, tentaremos iniciar todos os programas com o comando `rm(list = ls())`, que remove todos os objetos no espaço de trabalho. Obviamente, quando um programa usa objetos existentes (resultados de programas anteriores) limpar o espaço de trabalho é um erro;

✓ Utilize linhas em branco para separar seções de códigos em partes relacionadas;

✓ Utilize identação para distinguir a parte interna de uma instrução

1.2.2 Controle de fluxo

É muito comum que alguém queira que um pedaço do código seja executado se uma certa condição for verdadeira, também é comum querer que um pedaço de código seja repetido várias vezes. Para casos como esses, existem as estruturas de controle de fluxo. As principais ferramentas são:
Condicional - permitem executar um código de acordo com uma condição
Estrutura de condição `if`: executa um bloco apenas se uma condição for satisfeita. A forma básica é dada por:

```
if(condição) true_action
if(condição){
       true_action
  }else{
       false_action
  }
```

Se a condição for satisfeita, `true_action` é executada, caso contrário, a opção `false_action` é executada.

Loop - permitem executar repetidamente o código
Estrutura de loops `for`: executa um bloco repetidas vezes enquanto uma condição for verdadeira, executando uma instrução (geralmente de incremento ou decremento de uma variável) após cada execução. A forma básica

```
for(item in vetor){
       action
 }
```

Para cada item do vetor, `action` é chamado uma vez, atualizando o valor do item a cada vez. Para item é usual as letras `i, j` ou `k`.

Há também duas formas de terminar um loop `for` mais cedo:

- `next` - sai da iteração atual;

- `break` - sai do loop `for`.

Estrutura de loops `while`: executa um bloco enquanto uma condição for verdadeira. Loop `for` são úteis quando conhecemos antecipadamente o conjunto de valores que deseja iterar. Se não soubermos a estrutura `while` pode ser usada.

```
while(condição){
       action
 }
```

1.3 Funções

Uma função em `R` é muito parecida com uma função matemática e é dividida em três componentes: argumentos, corpo e saída. A função recebe entradas (argumentos) e aplica algum código a elas (corpo). A partir disso, obtém uma saída (valor retornado). Sua forma geral é expressa por:

```
nome <- function(argumento_1, argumento_2,...){
       action
       return(output)
 }
```

Um comando não obrigatório, mas que é bastante comum no final das funções é o `return`. Nossa função vai obter algum resultado e este comando faz com que o `R` retorne o objeto dentro dos parênteses de `return ( )` no console.