

R e RStudio para Iniciantes

**Material de Apoio para Cursos Quantitativos do Instituto de Economia da
Universidade Federal do Rio de Janeiro (IE/UFRJ)**

GPEQ/UFRJ

2024-03-26

Índice

Prefácio

O que você vai aprender

Pretendemos que você domine o *mínimo* necessário de programação em R para executar as tarefas que podem ser requisitadas pelo seu professor, independentemente do curso da área quantitativa em que estiver. Em outras palavras, se te pedirem algo que deva ser elaborado com auxílio de programação em R, você será capaz de fazê-lo após ler este material¹.

Na prática, o quê significa *dominar o mínimo necessário de programação em R*? Inclui entender alguns *conceitos* básicos – para quê serve a programação em nosso contexto, o que é a linguagem de programação R, o que é o RStudio, entre outros – assim como a *sintaxe* da linguagem – ou seja, o ato de escrever um código interpretável propriamente dito.

O que você não vai aprender

Não estamos em um curso de Ciência da Computação: você não irá aprender terminologias difíceis e/ou como a programação, de modo geral, funciona nos *detalhes*. Em outras palavras, vamos nos concentrar apenas em entender o necessário para construir e executar *códigos* em R (não se preocupe, ainda explicaremos o que é um *código em R*) a partir das tarefas que seu professor poderá pedir.

Além disso, o material não te dará proficiência em R. O que queremos dizer com isso? Bom, queremos dizer que você não será uma pessoa que dominará o R de forma *avançada*. Novamente: aqui, te ensinaremos apenas o necessário para que consiga concluir os cursos da área quantitativa. Mas, se você realmente quiser alcançar níveis mais altos, alguns livros podem te ajudar:

- [R for Data Science \(2ª edição\)](#)
- [Ciência de Dados em R](#)
- [Data Science for Psychologists](#)
- [An Introduction to R for Research](#)

¹Esperamos que os empecilhos que apareçam não sejam por conta de alguma dificuldade no ato de programar em si, mas por dúvidas com relação à matéria propriamente dita. De qualquer forma, fique tranquilo: se você não entendeu alguma parte do material, estaremos **sempre** abertos a te ajudar!

Preciso saber alguma coisa de forma antecipada?

Não. Você não precisa saber absolutamente *nada* de programação em R – não precisa nem mesmo saber o que o termo *programação* significa. O intuito do material é justamente te introduzir aos conceitos mais básicos!

A única coisa que você precisará será de acesso à um computador com internet. Utilizar um computador é necessário pois é nele onde ocorre o ato de programar; ter internet é importante porque, ao longo dos capítulos, precisaremos que você realize o *download* de certos arquivos – seja para instalar o R e o RStudio ou para *importar* algum arquivo diretamente para este último (não se preocupe, ainda explicaremos o que *importação* de um arquivo significa).

Como o material está organizado

O material está organizado em sete capítulos: o primeiro, que te mostra a motivação para programar, além de outros seis que buscam, em primeiro lugar, te guiar na instalação do R e RStudio e, na sequência, ensinar comandos e conceitos básicos que serão necessários ao longo dos cursos. Com intuito de facilitar o aprendizado, cada capítulo foi repartido em um certo número de seções (e subseções, quando necessário).

A lista de capítulos pode ser observada no menu à *esquerda*. Por sua vez, a lista de seções do capítulo em que você estiver pode ser observada no menu à *direita*. Perceba que, para ser direcionado a um determinado capítulo/seção, basta clicar em seu nome.

Prefácio

1 Por que programar?

Instalação



2 Instalando o R

3 Instalando o RStudio

Programando em R



4 Primeiros passos

5 Objetos

6 Funções e pacotes

7 Importando dados

Índice

Prefácio

O que você vai aprender

O que você **não** vai aprender

Preciso saber alguma coisa de forma antecipada?

Como o material está organizado

Dúvidas e sugestões: com quem falar?

“Caramba, queria tanto acessar uma parte específica do material que não lembro muito bem onde está... E agora?” Sem problemas: você pode pesquisar partes do texto ou palavras-chave no campo em branco logo acima do Prefácio!

Dúvidas e sugestões: com quem falar?

“Ué, no meu computador não aparece isso!”

“Caramba, achei aquele trequinho ali meio confuso... podia melhorar...”

“Nossa, que material show!”

Surgiu alguma dúvida ou então quer dar alguma sugestão de melhoria? Estamos totalmente abertos à qualquer tipo de crítica! Envie uma mensagem para pedro.hemsley@ie.ufrj.br.

1 Por que programar?

De forma simplificada, é possível definir o ato de programar como a passagem de determinados comandos para o computador, com a finalidade de que ele execute determinada tarefa. Se você deseja algo que pode ser feito de forma mais eficiente por uma máquina, provavelmente escreverá um código que seja interpretável por esta, de modo que seu desejo se concretize.

A capacidade de programar tornou-se uma habilidade essencial, especialmente para aqueles que desejam explorar o mundo da estatística e da matemática aplicados à determinada ciência social. Por exemplo, no contexto de interseção entre economia e matemática – principalmente na elaboração e solução de modelos teóricos – e entre economia e estatística – testando hipóteses e realizando previsões – a programação se coloca como uma ferramenta muito útil para economizar tempo de cálculo e garantir que, caso necessário, o mesmo processo seja concluído múltiplas vezes sem erros. Em outras palavras, a programação aplicada à determinada ciência social, como a economia, traz duas principais vantagens, exploradas melhoras a seguir.

1.1 Redução no tempo de cálculo

A primeira vantagem é a redução no tempo de cálculo de certos procedimentos que, se feitos de forma manual, levariam vários minutos, horas ou até mesmo dias. Vamos deixar mais claro com um exemplo.

No ensino fundamental, você aprendeu a resolver um sistema de equações simultâneas com 2 variáveis e 2 equações, muito provavelmente pelo método de substituição. Não levava muito tempo, certo? Acontece que, na cadeira de Álgebra Linear, você aprenderá como solucionar sistemas de n equações e n variáveis. Normalmente, quanto maior n , maior será a dificuldade de encontrar a solução do sistema. Ainda que existam *algoritmos* que permitam encontrar a solução de forma mais rápida, certo tempo será perdido se você os replicar de forma *manual*.

Com auxílio da programação, no entanto, é possível implementar estes mesmos algoritmos para obter o resultado de forma quase que *instantânea*. *O tempo que você levaria fazendo o procedimento manual praticamente se reduz a zero – ou fica mínimo, em relação ao inicial.* Observe que você ainda deve focar em saber como o algoritmo funciona, do contrário não será capaz de julgar se o que a máquina fez é realmente aquilo que você desejava.

1.2 Automação de processos

Na seção anterior, repare que estávamos discorrendo implicitamente sobre cálculos de ocorrência única – ou seja, realizamos o cálculo uma vez e não teríamos mais interesse de fazê-lo novamente em um futuro próximo. No entanto, outro benefício prático do ato de programar é a automação de tarefas repetitivas. Com a programação, é possível escrever e salvar *scripts* que automatizam tarefas tediosas de manipulação e análise de dados, permitindo que os pesquisadores se concentrem em questões analíticas de maior relevância.

Por exemplo, imagine que alguém te peça para calcular a média de certos valores que mudam de dia para dia. Você pode facilmente elaborar um *script* que, a partir de determinados números (sem especificar quais são), calcule sua média. Uma vez escrito e salvo, você pode passar a executá-lo sempre que quiser – no exemplo, todos os dias.

1.3 Vamos programar!

Em suma, aprender a programar oferece uma série de vantagens tangíveis para quem trabalha com estatística e matemática. Ela torna o trabalho mais eficiente e produtivo, permitindo que os profissionais explorem dados de maneiras antes inimagináveis e desenvolvam soluções personalizadas para os desafios enfrentados em suas áreas de atuação.

No restante do material, aprenderemos a programar utilizando a *linguagem de programação R*. Em outras palavras, aprenderemos sua *sintaxe*, isto é, a forma de escrever comandos corretamente para que a máquina seja capaz de interpretar e executar o que queremos como resultado.

2 Objetos

Na apresentação dessa parte do material, trouxemos uma citação que, em parte, dizia:

Everything that exists is an object.

Um objeto é simplesmente um nome que guarda um valor ou código. Não há como ser mais direto: tudo que existe no R é um *objeto*, inclusive as funções que veremos no capítulo seguinte. Nesse capítulo, veremos com detalhe os objetos que são designados à armazenar dados. Antes, no entanto, vamos dar um passo para trás e explicar o que são dados.

2.1 Dados

Segundo a Oxford Languages, dados são

fatos e estatísticas coletadas de forma conjunta para referência ou análise.

Na prática, dados nos mostram informações sobre determinado indivíduo ou situação que procuramos descrever, seja uma pessoa, instituição, comportamento, condição geográfica, etc. O número de horas que você dormiu essa noite é um dado. A lista que relata quem é ou não calvo na sua família é uma *coleção* de dados. A expectativa, hoje, de quanto será a inflação acumulada nos próximos 12 meses é um dado. A variação percentual do Produto Interno Bruto (PIB) real no último trimestre é um dado. A lista que mostra a sequência de variações do PIB real nos últimos dez trimestres é uma *série temporal*, isto é, dados em sequência ao longo do tempo.

2.1.1 Tipo & Forma

Vamos nos aprofundar um pouco mais. Ao lidar formalmente com dados, **devemos ter mente que eles são compostos por uma ou mais variáveis e seus valores**. Uma *variável* é uma *dimensão ou propriedade que descreve uma unidade de observação* (por exemplo, uma pessoa) e normalmente pode assumir valores diferentes. Por outro lado, os *valores* são as *instâncias concretas que uma variável atribui a cada unidade de observação e são ainda caracterizados por seu intervalo* (por exemplo, valores categóricos versus valores contínuos) e seu *tipo* (por

exemplo, valores lógicos, numéricos ou de caracteres). Estaremos interessados no *tipo* dos dados. A Tabela ?? apresenta os que podem aparecer com maior frequência.

Tabela 2.1: Tipos mais comuns de dados

Tipo	Serve para representar...	Exemplo
Númerico	números do tipo <i>integer</i> (inteiro) ou <i>double</i> (reais)	1, 3.2, 0.89
Texto (<i>string</i>)	caracteres (letras, palavras ou setenças)	“Ana jogou bola”
Lógico	valores verdade do tipo lógico (valores booleanos)	TRUE, FALSE, NA
Tempo	datas e horas	14/04/1999

Voltando ao primeiro exemplo, uma pessoa pode ser descrita pelas variáveis *nome*, *número de horas dormidas* e *se dormiu ou não mais de oito horas*. Os valores correspondentes a essas variáveis seriam do tipo texto (por exemplo, “Pedro”), numéricos (número de horas) e lógicos (TRUE ou FALSE, definido em função do tempo descansado¹). **Note a diferença entre *dado* e *valor***. O número 10 é um valor, sem significado. Por outro lado, “10 horas dormidas” é um dado, caracterizado pelo valor 10 e pela variável “*horas dormidas*”.

Outro aspecto importante sobre os dados está em sua forma, ou seja, como os dados podem ser organizados. A Tabela ?? apresenta as formas mais comuns de organização.

Tabela 2.2: Formas pelas quais os dados podem ser organizados

Formato	Os dados se apresentam como...	Exemplo
Escalar	elementos individuais	“AB”, 4, TRUE
Retangular	dados organizados em i linhas e j colunas	Vetores e Tabelas de Dados
Não-retangular	junção de uma ou mais estruturas de dados	Listas

Um escalar é um elemento único, que pode ser de qualquer tipo. Ou seja, a representação elementar de um dado se dá através de um escalar! Por exemplo, o tipo sanguíneo de determinada pessoa, representado pelos caracteres “AB”, é um escalar do tipo texto. Você pode pensar no escalar como um dado organizado em 1 linha e 1 coluna.

Por sua vez, dados retangulares são àqueles cuja organização ocorre em i linhas e j colunas, tal que $i, j \in \mathbb{N}$ e $i > 1$ ou $j > 1$. As formas retangulares mais comuns são *vetores*, *matrizes* e *tabelas de dados*. Uma matriz é uma forma de organização de dados *numéricos* em i linhas

¹Se o número de horas que a pessoa descansou for maior do que 8, então a variável deverá apresentar valor igual a TRUE – ou seja, é verdade que a pessoa dormiu mais de 8 horas. Caso contrário, FALSE.

e j colunas. Quando uma matriz possui i linhas e 1 coluna *ou* 1 linha e j colunas, chamamos de vetor-coluna e vetor-linha, respectivamente; em muitos casos, chamamos apenas de *vetor*. Assim, o vetor é um caso especial de matriz unidimensional. As tabelas de dados, por outro lado, possuem i linhas e j colunas, tal que $i > 1$ e $j > 1$. Além disso, aceitam todos os tipos de dado – por exemplo, numéricos, de textos ou lógicos – em qualquer que seja a combinação de linha e coluna.

Por sua vez, dados não-retangulares se referem a toda organização de dados que não seja feita em linhas e colunas relacionadas entre si. A forma mais comum é a lista. Observe um exemplo abaixo: cada característica pode ser entendida como um elemento de uma lista. Apesar de pertencerem a mesma estrutura, os elementos não se comunicam entre si.

Gênero	Jorge Masculino	Laís Feminino	Matheus Masculino	Laura Feminino	Nathália Feminino
Idade	Jorge 18	Laís 23	Matheus 22	Laura 21	Nathália 21
Altura (cm)	Jorge 180	Laís 170	Matheus 170	Laura 175	Nathália 168
Peso (kg)	Jorge 76	Laís 65	Matheus 70	Laura 68	Nathália 66

Nesse caso em específico, conseguimos fazer a transição para uma tabela (forma retangular) pois todos os elementos são características das mesmas pessoas. Em uma tabela de dados, automaticamente temos uma relação entre os dados: cada linha contém características de uma unidade específica.

Tabela 2.4

Nome	Gênero	Idade	Altura (cm)	Peso (kg)
Jorge	Masculino	18	180	76
Laís	Feminino	23	170	65
Matheus	Masculino	22	175	70
Laura	Feminino	21	181	68
Nathália	Feminino	21	168	66

2.2 Estruturas de Dados no R

Na seção anterior, vimos os conceitos de *tipo* e *forma*. Tenha em mente que são duas definições que existem independentemente de qualquer linguagem de programação – elas versam sobre *dados* de forma geral.

Por outro lado, agora veremos o conceito e alguns exemplos de *estrutura de dados* para o R. **A estrutura de dados é a forma pela qual o R classificará um objeto em relação ao tipo e a forma dos dados que contém.** Existe uma estrutura de dados para cada combinação de tipo e forma? **Não.** Compreender as principais estruturas disponíveis no R requer vê-las como uma combinação de

- (a) *algum* formato de dados
- (b) o fato de conterem um único ou vários tipos de dados

2.2.1 Criando e armazenando objetos na memória

Antes de conhecê-las, no entanto, vamos entender melhor os comandos para criar e armazenar *qualquer* objeto (seja ele para armazenar dados, como nesse capítulo, ou para criar funções, que serão vistas no próximo) na memória do R.

Para *criar* e *armazenar* um objeto, sempre escreveremos inicialmente seu nome (escolhido por você), seguido de um dos *operadores de atribuição* (ou *assignment operators*, como são conhecidos) e, por fim, o objeto propriamente dito com as informações de nosso interesse. O principal operador de atribuição para se criar objetos é `<-`. Outro operador que é comumente utilizado para cumprir a mesma tarefa é `=`. Ainda que exista uma leve diferença entre ambos, ao longo dos cursos será possível utilizar o operador de sua preferência. Por ser o ideal, utilizaremos `<-` no restante do material.

```
nome_do_objeto <- >objeto com informações<
nome do objeto = >objeto com informações<
```

No parágrafo anterior, observe que está escrito ‘*criar e armazenar*’. Nós poderíamos simplesmente criar um objeto, sem armazená-lo na memória do R. Nesse caso, não teríamos o nome do objeto disponível na aba **Environment** (ou seja, ele não seria armazenado no nosso ambiente de trabalho) e seria bem mais complicado registrar todas as mudanças que viermos a fazer nele. Aconteceria apenas a ocorrência de uma única saída no Console com a estrutura do objeto criado (de forma semelhante ao que fizemos no capítulo anterior) – o quê não tem grande utilidade para nós, exceto caso você queira verificar a estrutura do objeto antes de realmente armazená-lo.