Lição 1: Fundamentos

Bem-vinde ao curso de Introdução à Estatística para Linguistas!

Primeiro, vamos explorar alguns fundamentos da linguagem. O R pode funcionar como uma calculadora. Digite `4 + 9` no *script* desta lição[8] e pressione CTRL + ENTER para ver o resultado.

```
4 + 9
```

```
## [1] 13
```

O R apresenta o resultado 13. Mas usar o R para fazer esse tipo de cálculo, em vez de usar uma simples calculadora, pode parecer desnecessário. Na maior parte das vezes, usamos o R para automatizar algum processo ou evitar repetição. Podemos querer usar o resultado de `4 + 9` em outros cálculos. Para isso, em vez de redigitar 4 + 9 várias vezes, podemos guardar o resultado em uma variável.

Para atribuir um valor a uma variável, use o operador de atribuição `<-`, que é o símbolo de menor seguido do símbolo de menos. O operador de atribuição representa iconicamente uma flecha, que atribui o valor à direita a uma variável à esquerda. Para atribuir o resultado de 4 + 9 a uma nova variável chamada `x`, digite `x <- 4 + 9` e rode essa linha de comando com CTRL + ENTER.

```
x <- 4 + 9
```

Você deve ter percebido que o R não exibiu o resultado 13 desta vez. Quando se usa o operador de atribuição, o R assume que você não precisa do resultado imediatamente, mas sim para algum uso futuro. Para ver o conteúdo da variável `x`, digite `x` e pressione CTRL + ENTER.

```
x
```

```
## [1] 13
```

[8] Os *scripts* de todas as lições podem ser acessados pelos links disponibilizados na seção "Links para *scripts* das lições".

Como uma variável, `x` agora pode entrar em outros cálculos. Qual seria o resultado de x + 10?

- 21
- 23
- 25

Vamos testar o resultado da expressão (pra qual já sabemos o resultado). Digite `x + 10` e pressione CTRL + ENTER.

```
x + 10
```

```
## [1] 23
```

O uso de atribuição de variáveis é recursivo. Podemos atribuir o valor de x + 10 a uma nova variável `y`. Tente fazer isso agora.

```
y <- x + 10
```

Agora visualize o valor de `y`.

```
y
```

```
## [1] 23
```

Acho que deu pra pegar o jeitão da coisa, né? A variável `x` terá o valor de 13 até que se atribua novo valor a ela ou até que se encerre a sessão do R sem salvar o espaço de trabalho.

Procure no RStudio a janela com as abas Environment e History. Em Environment, o R lista os objetos criados e seus respectivos valores. Na aba History, o R mostra o histórico de comandos.

Para nós, linguistas, o interesse em usar o R é em sua capacidade de lidar bem não só com números, mas também com caracteres e elementos textuais. Criamos uma variável com uma sequência de caracteres usando as aspas. Digite `z <- "gato"` para criar a variável `z`.

```
z <- "gato"
```

Agora visualize o conteúdo de `z`.

```
z
```

```
## [1] "gato"
```

Note as aspas em volta do termo `gato`, que indica que a variável contém uma sequência de caracteres. O que você acha que vai acontecer se tentarmos somar a variável `z` com o número 1?

- O R retornará o plural de `gato`;
- O R retornará "gato" + 1;
- O R retornará um erro seguido de uma explicação;
- O resultado é imprevisível;
- Aparecerá uma mensagem de que seu computador se autodestruirá em 5 segundos

Toda vez que o R não consegue executar uma linha de comando, ele dá uma mensagem de erro e indica o que está errado. Ao tentar computar `z + 1`, ele indicará que um dos argumentos não é numérico e que, portanto, não consegue fazer a soma. Ao ocorrer uma mensagem de erro, não se desespere! Basta ler a mensagem e tentar descobrir onde estava o erro. Se quiser testar, digite `z + 1` no Console para ver o resultado.

Aqui vale a pena mencionar um recurso que o R e diversas outras linguagens de programação têm: pode-se voltar a uma linha de comando anterior pressionando a flecha para cima ↑, estando com o cursor no Console. Faça isso até voltar ao comando `x + 10`, que executamos acima.

```
x + 10
## [1] 23
```

Até agora, inserimos apenas um elemento nas variáveis `x`, `y` e `z`. Para juntar mais de um elemento, usamos a função `c()`, em que `c` significa "combinar". Suponhamos então que você queira juntar os números 1, 5 e 7. Digite `c(1, 5, 7)`. Note que não há espaço entre o nome da função e a abertura de parênteses, e que a letra `c` é minúscula.

```
c(1, 5, 7)
## [1] 1 5 7
```

Também é possível juntar outros objetos previamente criados. Vamos juntar as variáveis `x`, `y`, criadas anteriormente, e o número 230. Além disso, vamos guardar esses elementos dentro de um objeto, que chamaremos de `aleatorio`. Digite `aleatorio <- c(x, y, 230)`.

```
aleatorio <- c(x, y, 230)
```

Agora veja o conteúdo do objeto `aleatorio`.

```
aleatorio
```

```
## [1]  13  23 230
```

O objeto `aleatorio`, assim como `x`, `y` e `z`, é um *vetor*. Vetores são sequências de dados de um mesmo tipo (numérico, de caracteres, ou lógico). Este último, que ainda não vimos, conteria apenas valores TRUE (= "verdadeiro") ou FALSE (= "falso"). Mas fiquemos, por ora, apenas nos vetores numéricos ou de caracteres. Vetores podem ter qualquer número de elementos – e muitas vezes vamos trabalhar com vetores de centenas ou milhares de dados.

Sempre que tiver dúvidas sobre uma função específica, você pode chamar o Help do R por meio do comando `?` seguido do nome da função, sem os parênteses. Por exemplo, digite `?c` para visualizar mais informações sobre essa função na aba Help.

```
?c
```

Veja na aba Help da janela Files, Plots, Packages... que o R abriu a ajuda sobre essa função. Fique à vontade para explorá-la.

Outro operador útil no R é `:`, que representa um intervalo. Crie um vetor chamado `numeros` que contém os números de 20 a 39. Digite `numeros <- c(20:39)`.

```
numeros <- c(20:39)
```

Veja agora o conteúdo do vetor recém-criado.

```
numeros
```

```
##  [1] 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
```

Esse é um jeito muito mais fácil de criar um vetor numérico do que digitar c(20, 21, 22, 23...).

Você notou que os vetores que criamos não contêm caracteres especiais, como diacríticos (p.ex. ´^), espaços e outros como - & $...? Ao nomear objetos no R, evite esses caracteres!

Vejamos agora outras funções úteis no R. Algo importante a se fazer a cada nova sessão do R é gerenciar o espaço de trabalho. Para isso, usamos as funções `getwd()` ou `setwd()`. `wd` nessas funções significa "working directory", ou diretório de trabalho, que é a pasta em seu sistema que o R está usando como referência para interagir com arquivos fora do R (p.ex., seu arquivo de dados linguísticos!).

Vamos descobrir qual é o diretório de trabalho atual. Digite `getwd()`.

```
getwd()
## [1] "/Users/livia/Dropbox/_R/RMarkdown/IEL_203_EbookABRALIN"
```

N.B.: O resultado em seu computador será diferente do que aparece aqui!

Observe o formato com que o R retorna o atual diretório de trabalho. Trata-se do caminho completo para a pasta, entre aspas, e com barras para a direita `/`. Para mudar o diretório de trabalho, usamos a função `setwd()`. Dentro dos parênteses dessa função, digitamos o caminho completo do novo diretório no mesmo formato – entre aspas e com barras para a direita. P.ex.: `setwd("C:/Users/IntroaEstatistica")`.

Mas há um jeito relativamente mais fácil de fazer isso no RStudio. Procure a janela que contém as abas Files, Plots, Packages... e clique sobre a aba Files. Dentro dessa aba, no canto superior direito, clique sobre as reticências ..., como se vê na Figura 1.1.

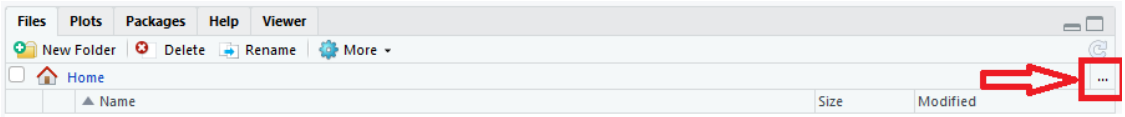

Figura 1.1: Localização das reticências na aba Files. Fonte: própria.

Abrirá uma janela que permite navegar até a pasta desejada. Selecione a pasta que deseja transformar em novo diretório de trabalho (Figura 1.2). Recomendo que você crie uma nova pasta em seu computador só para isso!

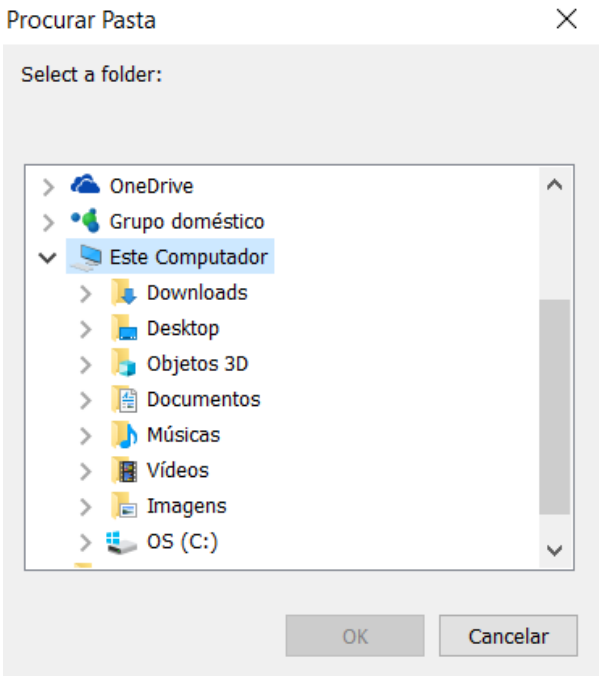

Figura 1.2: Janela Procurar Pasta. Fonte: própria.

Você verá seu conteúdo na aba Files. Em seguida, clique sobre "More" e em "Set As Working Directory" (Figura 1.3). Faça isso agora.

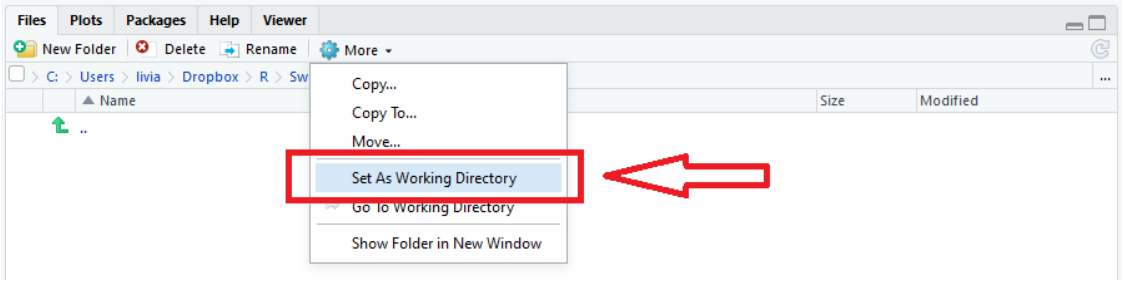

Figura 1.3: Definir diretório de trabalho pela janela. Fonte: própria.

```
setwd("~/Dropbox/_R/swirl/Introducao_a_Estatistica_para_Linguistas/data")
```

N.B.: O resultado em seu computador provavelmente será diferente do que aparece aqui!

O R executou a linha de comando no Console e estabeleceu um novo diretório de trabalho. Nesse caso, utilizamos a interface do RStudio para executar um comando sem ter que digitar uma linha. Embora isso seja mais fácil quando executado uma única vez, essa ação pode ser mais trabalhosa se se tornar repetitiva. Por exemplo, se quisermos

estabelecer o mesmo diretório de trabalho a cada vez que se inicia uma sessão no R, aí já não vale mais a pena estabelecer o diretório de trabalho desse modo.

Uma boa ideia é selecionar e copiar (CTRL/Command + C) essa linha de comando do Console e colá-la no seu *script*. Faça isso agora. Note que, ao fazer uma alteração no arquivo do *script*, aparece um asterisco vermelho ao lado do nome do arquivo. Isso indica que há modificações não salvas. Salve com CTRL/Command + S. Aliás, é uma boa ideia fazer isso periodicamente, para não perder seu trabalho!

Os arquivos de *scripts* funcionam como qualquer outro arquivo comum. Pode-se abri-los, fechá-los, salvá-los com outro nome com Salvar como... Se em seus *scripts* você for usar sequências de caracteres em português, uma dica é usar a opção File > Save with encoding e escolher a opção UTF-8, que lê corretamente caracteres especiais como diacríticos (´ ` ^). (Mas lembre-se que é bom evitá-los ao criar ou nomear objetos no R!)

Por fim, vamos ver mais uma função que será usada extensivamente em suas análises – `read_csv()`. Na maior parte dos casos para as análises estatísticas, queremos analisar uma planilha de dados preparada previamente em um programa como o Excel ou o Calc. A função `read_csv()` faz parte dos pacotes readr e tidyverse – este último, um conjunto de pacotes para manipulação de dados. Os pacotes (também chamados de "bibliotecas") são conjuntos de funções criadas por usuários do R e disponibilizadas aos demais usuários, e que não fazem parte da instalação base do R. Sempre que quiser instalar um novo pacote, você deve usar a função `install.packages()`, um comando que você usou para instalar os pacotes swirl e vários outros para este curso!

Para usar funções que não fazem parte da instalação base do R, é necessário acessar, a cada nova sessão, o pacote/biblioteca por meio da função `library()`. Aplique-a ao pacote tidyverse.

```
library(tidyverse)
```

A planilha DadosRT tem o formato .csv (= comma separated values). Para ler esse tipo de arquivo no R, podemos usar a função `read_csv()`, do pacote tidyverse. Rode a linha de comando a seguir com a função `read_csv()` – já pronta! –, para que o R leia os dados da planilha e os insira dentro de um objeto chamado `dados`. Basta colocar o cursor

em qualquer ponto da linha de comando e pressionar CTRL + ENTER. É possível que demore um tempinho para rodá-la; enquanto o R executa esse comando, note que aparece um círculo vermelho no topo direito da janela Console (STOP), que indica que o R está executando determinado comando e que você deve esperar antes de dar novos comandos.

```
dados <- read_csv("DadosRT.csv",
                  col_types =
                    cols(.default = col_factor(),
                         cont.precedente = col_character(),
                         ocorrencia = col_character(),
                         cont.seguinte = col_character(),
                         IDADE = col_integer(),
                         INDICE.SOCIO = col_double(),
                         FREQUENCIA = col_double()
                         )
                  )
```

Essa linha de comando é mais complexa do que as anteriores, de modo que vale a pena analisar sua estrutura. Primeiro, note que, mesmo que o comando ocupe mais de uma linha, trata-se de uma única linha de comando. Ao ler `dados <- read_csv("DadosRT.csv",`, o R reconhece que esta é uma linha de comando incompleta, e isso é sinalizado com o símbolo "+" no Console. Ela está incompleta porque, nessa primeira linha, um parêntese foi aberto, mas não foi fechado. Essa é uma regra sagrada na programação: tudo que abre tem que fechar! Há, ademais, uma vírgula, que indica que haverá mais argumentos na função.

Usamos a função `read_csv()` com dois argumentos, file e col_types. Todas as funções têm parênteses e, dentro deles, podem ser inseridos zero, um ou mais argumentos – como nas funções `getwd()`, `library()` e `c()`, respectivamente. Acesse a Ajuda da função `read_csv()`.

```
?read_csv
```

O primeiro argumento, file, foi preenchido com "DadosRT.csv". Porque colocamos esse argumento em sua posição *default*, não foi necessário digitar file = "DadosRT.csv" para especificar qual argumento é esse. O segundo argumento usado foi col_types. Esse argumento tomou ele mesmo outra função, chamada `cols()`, para

definir o tipo das colunas nesse dataframe. Note aí a recursividade! Na função `cols()`, especificamos que o tipo *default* de cada coluna é `factor`, exceto pelas colunas cont.precedente, ocorrencia e cont.seguinte, que devem ser lidas como "character", e as colunas IDADE, INDICE.SOCIO e FREQUENCIA, que devem ser lidas como "integer" e "double" (tipos de número). Na Lição 3, vamos ver essa tipologia de dados com mais detalhes.

Note, além disso, que `col_factor()`, `col_character()`, `col_integer()` e `col_double()` são também funções – algo que sabemos pelos parênteses. É possível criar outras especificações dentro dessas funções como, por exemplo, mudar a ordem dos níveis de uma variável. Também veremos isso na Lição 3. Por ora, interessa chamar a atenção para essa estrutura mais geral de funções, que sempre têm parênteses, e cujos argumentos, dentro dos parênteses, são separados por vírgulas. Os diferentes argumentos das funções `read_csv()` e `cols()` foram colocados em linhas distintas no *script* justamente para mais bem visualizar a estrutura da função `read_csv()` e as demais funções nela encaixadas.

Por fim, você também deve ter notado, na Ajuda da função `read_csv()`, que existem vários outros argumentos nas funções da família read_delim. Alguns argumentos podem ter um valor *default*; quando o usuário não especifica nada para determinado argumento, o R assume que seu valor é o *default*, pré-especificado pelo autor ou autora da função. Por exemplo, para `read_csv()`, o argumento col_names é pré-especificado como TRUE, de modo que, se nada for especificado quanto aos nomes das colunas, o R assume que a primeira linha contém o nome das colunas do dataframe.

Vamos inspecionar, então, o conteúdo do objeto recém-criado `dados`.

```
dados
```

```
## # A tibble: 9,226 × 20
##    VD         PARTICIPANTE SEXO.GENERO IDADE FAIXA.ETARIA ESCOLARID
ADE
##    <fct>      <fct>        <fct>       <int> <fct>        <fct>
##  1 retroflexo IvanaB       feminino       30 1a           fundament
al
##  2 retroflexo IvanaB       feminino       30 1a           fundament
al
##  3 retroflexo IvanaB       feminino       30 1a           fundament
```

```
al
##  4 tepe       IvanaB       feminino       30 1a           fundament
al
##  5 retroflexo IvanaB       feminino       30 1a           fundament
al
##  6 retroflexo IvanaB       feminino       30 1a           fundament
al
##  7 retroflexo IvanaB       feminino       30 1a           fundament
al
##  8 retroflexo IvanaB       feminino       30 1a           fundament
al
##  9 tepe       IvanaB       feminino       30 1a           fundament
al
## 10 retroflexo IvanaB       feminino       30 1a           fundament
al
## # … with 9,216 more rows, and 14 more variables: REGIAO <fct>,
## #   INDICE.SOCIO <dbl>, ORIGEM.PAIS <fct>, CONT.FON.PREC <fct>,
## #   CONT.FON.SEG <fct>, TONICIDADE <fct>, POSICAO.R <fct>,
## #   CLASSE.MORFOLOGICA <fct>, FREQUENCIA <dbl>, ESTILO <fct>,
## #   ITEM.LEXICAL <fct>, cont.precedente <chr>, ocorrencia <chr>,
## #   cont.seguinte <chr>
```

Dentro do universo tidyverse, o R retorna um tibble – um resumo de um dataframe. O arquivo DadosRT, agora disponível em `dados`, é uma planilha com dados da pronúncia variável de /r/ em coda, em palavras como "porta" e "mulher", na fala de paulistanos. Encontraremos esse arquivo novamente em lições futuras.

Nesta lição, você aprendeu alguns comandos fundamentais no R, que serão bastante utilizados nas próximas lições – como criar vetores, como acessar a Ajuda, como usar as funções `c()`, `getwd()`, `setwd()`, `library()`, `read_csv()`. Além disso, vimos as quatro janelas que compõem o ambiente do RStudio.

Embora possa parecer complicado digitar linhas de comando em vez de clicar em botões, normalmente só se tem que digitar os comandos uma ou poucas vezes. Na maior parte do tempo, trabalha-se com *scripts*, que é um conjunto de linhas de comando que são guardadas em arquivos de extensão .R.

Para saber mais

Recomendo que você faça as lições 1 ("Basic Building Blocks") e 2 ("Workspace and Files") do curso R Programming, do swirl.

Exercícios

1. Crie um vetor chamado `numeros1` que contém os números 1, 3 e 5.
2. Crie um vetor chamado `numeros2` que contém os números 11, 13 e 15.
3. Crie um vetor chamado `classe_morfologica` que contém os termos "substantivo", "adjetivo", "adverbio", "pronome" e "verbo".
4. Guarde o caminho do atual diretório de trabalho em uma variável chamada `dir_antigo`.
5. Defina como diretório de trabalho a pasta em que se localiza a planilha DadosRT.csv.
6. Cheque se o novo diretório de trabalho está correto.
7. Instale o pacote lme4. Ele será usado em lições futuras para criar modelos de efeitos mistos.
8. Carregue o pacote tidyverse.
9. Carregue a planilha DadosRT.csv em um objeto chamado `dadosR`. Use a função `read_csv()` para isso.
10. Inspecione os elementos do vetor `numeros1`.
11. Se somarmos os vetores `numeros1` e `numeros2`, qual será o resultado?
    a. Um vetor de um elemento
    b. Um vetor de três elementos
    c. Um erro
12. Some os vetores `numeros1` e `numeros2` e cheque a resposta.
13. Se multiplicarmos o vetor `numeros1` por 2, qual será o resultado?
    a. Um vetor de um elemento
    b. Um vetor de três elementos
    c. Um erro
14. Multiplique o vetor `numeros1` por 2. O sinal de multiplicação no R é o asterisco *.
15. Inspecione os valores do vetor `classe_morfologica`.
16. Se somarmos o vetor `classe_morfologica` com 2, qual será o resultado?

a. Um vetor de um elemento
b. Um vetor de três elementos
c. Um erro

17. A função `length()` informa quantos elementos há dentro de um objeto do R. Veja a descrição dessa função por meio da Ajuda.
18. Escolha a linha de comando correta para aplicar a função `length()` no vetor `classe_morfologica`.
    a. `length(classe_morfologica)`
    b. `length("classe_morfologica")`
    c. `classe_morfologica <- length()`
19. Aplique a função `length()` no vetor `classe_morfologica`.
20. Aplique a função `length()` no vetor `numeros2` e atribua o resultado a uma variável chamada `N.numeros2`.
21. Qual é o conteúdo do vetor `N.numeros2`?
    a. Um vetor de um elemento
    b. Um vetor de três elementos
    c. Um erro
22. Qual é o valor contido no vetor `N.numeros2`?
    a. 1
    b. 3
    c. 5
23. Defina o antigo diretório de trabalho novamente como o diretório de trabalho atual. Use a função `setwd()` e a variável `dir_antigo` para isso.