Introdução ao R

Erikson Kaszubowski

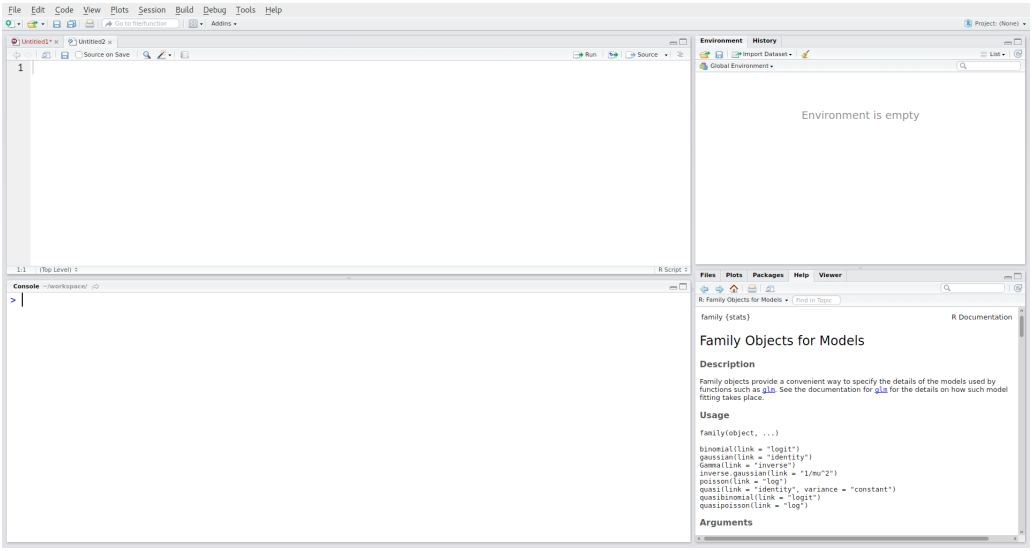
10 de maio de 2016

Conteúdo

O `R` é, ao mesmo tempo, um ambiente de desenvolvimento especializado em análise de dados e uma linguagem de programação. Por isso, é importante ter uma noção básica de alguns princípios de programação para entender melhor como conduzir uma análise no `R`. Iniciaremos com uma breve introdução à interface gráfica do `RStudio` e à sintaxe do `R`, para que vocês possam compreender melhor como executar análises.

1 Interface do `RStudio`

O próprio `R` possui uma interface gráfica bastante útil para conduzir análises de dados. O `RStudio`, porém, possui muito mais funcionalidades que facilitam a programação com o `R`. Por esse motivo, trabalharemos com o `R` diretamente dentro do `RStudio`.

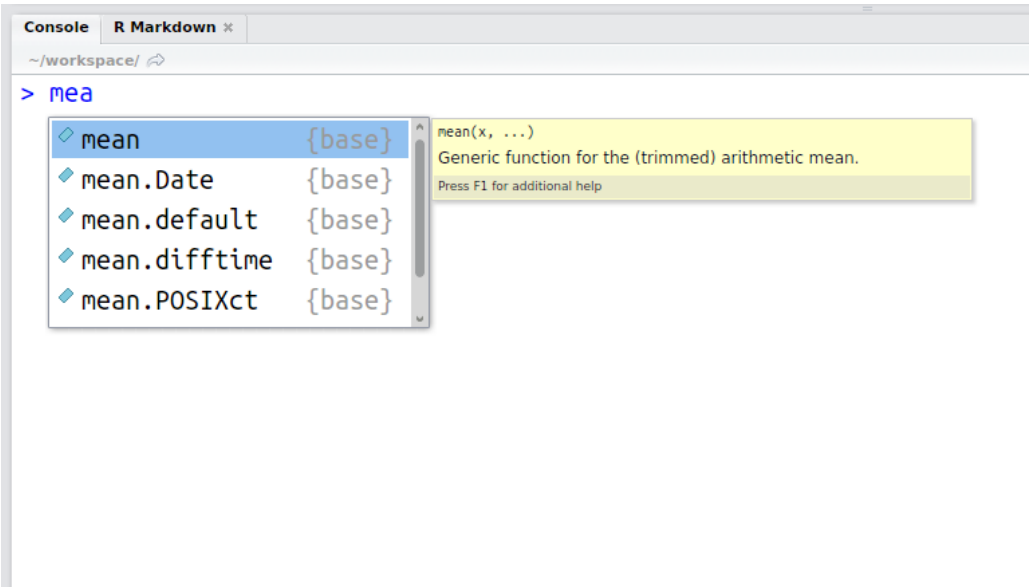


A interface do `RStudio` é dividida em quatro áreas por padrão. No quadrante superior, à esquerda, se encontra a janela para edição do código-fonte de *scripts* do `R`. Um *script* é uma sequência de comandos que pode ser salva e executada a qualquer momento, e é uma maneira útil de salvar os procedimentos de análise que foram realizados sobre um banco de dados para replicar os resultados obtidos. Se há mais de um arquivo aberto, como na imagem acima, os diferentes arquivos são acessados por meio de abas na parte superior da janela. Os comandos digitados num arquivo de *script* podem ser executados um a um clicando no botão **Run** no canto superior direito da janela, ou pressionando `Ctrl + Enter` sobre a linha desejada. Caso se queira executar o arquivo de *script* por completo, basta clicar no botão **Source**.

Nessa mesma janela também são abertas as visualizações de banco de dados que tenhamos carregado na memória do `R`. Apesar da interface do `RStudio` permitir visualizar as variáveis e casos de um banco de dados, com opções de filtro e ordenamento dos casos, não é possível editar os dados diretamente na interface gráfica. A manipulação dos dados para seleção de casos, variáveis e qualquer outro tipo de edição deve ser feita diretamente pelas funções do `R`.

Logo abaixo da janela de edição de *scripts* se encontra o *console* do R. Essa será a janela que mais utilizaremos para conduzir nossas análises. O console é uma ferramenta interativa que permite-nos digitar comandos para o `R`, que os executa imediatamente e imprime na tela os resultados. Todos os comandos que veremos nesta introdução devem ser digitados diretamente no console, logo na sequência do caractere `>`.

O `RStudio` possui uma funcionalidade chamada de *autocomplete* que pode nos ajudar bastante para lembrar os parâmetros e para que servem determinadas funções. Quando começamos a digitar qualquer comando no console, o `RStudio` automaticamente oferece uma lista de funções que possuem aqueles caracteres. Caso a lista não apareça, basta pressionarmos `Ctrl + Espaço`. Se deixamos uma das opções da lista selecionada por um curto período de tempo, aparece uma nova janela contextual explicando para que serve a função.



No quadrante superior direito temos a janela do *ambiente* (*Environment*) e do *histórico* (*History*), organizadas por abas. O ambiente permite visualizar todos os objetos e variáveis que foram criados durante um sessão, apresentando seu nome e seus valores. É possível

salvar todos os objetos do ambiente utilizando o botão com o ícone de disquete nesta janela. Da mesma forma, é possível restaurar o ambiente de um sessão anterior utilizando o botão com o ícone de pasta. Por padrão o `RStudio` salva o ambiente ativo toda vez que o aplicativo é encerrado, e o carrega automaticamente quando é reiniciado.

A janela do ambiente permite também importar bancos de dados para a memória do `R`. O botão *Import Dataset* dá a opção de carregar arquivos no computador local ou por meio da internet. É importante ressaltar que essa funcionalidade só pode ser utilizada quando os dados estão num arquivo de texto puro, como o formato `CSV` (*comma separated value*), e não serve para carregar arquivos com codificações especiais, como do Excel. Veremos mais detalhes de como abrir arquivos de banco de dados logo abaixo, quando carregarmos nosso primeiro arquivo para análise.

A aba *History* mostra uma lista com todos os comandos que foram digitados no console ao longo de uma sessão. É possível fazer busca nesta lista, facilitando a recuperação de procedimentos de análise. Da mesma forma como o ambiente, o histórico pode ser salvo e reaberto em novas sessões.

Por fim, abaixo da janela do ambiente, temos uma janela com diversas funcionalidades diferentes, cada uma demarcada por uma aba. A aba *Files* é um explorador de arquivo que permite navegar os arquivos do computador local, criar pastas, deletar e renomear arquivos. A aba *Plots* apresenta o último gráfico criado a partir de uma análise, mas é possível navegar entre gráficos utilizando os ícones de flecha presente nesta janela. O `RStudio` permite salvar os gráficos em arquivos de imagem, para inserção em documentos, por exemplo, por meio do botão *Export*. A aba *Packages* apresenta uma lista de *pacotes* disponíveis para serem carregados ou instalados. Um pacote é um conjunto de funções e estruturas de dados que estendem as funções básicas do `R`, ampliando as possibilidade de análise. Um pacote pode ser carregado na memória por meio de um clique na caixa de seleção logo ao lado do nome do pacote. Clicar no nome do pacote nos leva a uma página de ajuda dentro do próprio `RStudio` que apresenta para que serve aquele pacote. Essa janela de ajuda é a aba *Help* desta janela, e permite visualizar também a ajuda disponível para funções, como veremos logo abaixo. Finalmente, a aba *Viewer* apresenta a visualização de programas dinâmicos criados dentro do `R`.

2 Sintaxe básica do `R`

Agora que já conseguimos nos localizar na interface do `RStudio`, podemos finalmente passar para a utilização do `R` por meio do console. O `R`, sendo uma linguagem de programação,

nada mais é do que uma calculadora poderosa que nos permite realizar operações bastante complicadas. Mas antes de entrarmos nessas operações complicadas de análise, vamos ver como realizar operações mais simples, criar variáveis e manipular objetos. Compreender bem esses aspectos básicos facilitarão o entendimento de como utilizar o `R` para realizar análises de dados.

2.1 Operações aritméticas

Para realizarmos uma operação no `R`, basta digitarmos o comando no console, logo após o `>`, e pressionarmos `Enter`. O `R` processa o comando e imprime na tela o resultado. Do lado esquerdo do resultado aparece um número entre colchetes, indicando o número da linha do resultado. Esse número serve apenas para referência no caso de resultados longos e pode ser tranquilamente ignorado.

```
# Soma: '+'
122 + 37
> [1] 159

# Subtração: '-'
43 - 91
> [1] -48

# Multiplicação: '*'
378 * 9
> [1] 3402

# Divisão: '/'
507/13
> [1] 39

# Exponenciação: '^'
23^4
> [1] 279841
```

A ordem da resolução das operações segue os padrões usuais: primeiro exponenciações, seguido de multiplicações e divisões, na ordem em que aparecem, e, por fim, adições e subtrações, também na ordem em que aparecem. A prioridade na resolução de operações

pode ser alterada utilizando parênteses em torno da expressão que deve ser resolvida primeiro, como na aritmética.

2.2 Operações lógicas

Em qualquer linguagem de programação, é essencial poder determinar o valor de verdade de uma determinada condição. Para isso, são utilizados operadores *booleanos* ou *lógicos*, que avaliam a condição e retornam uma variável lógica, que só pode ter como valore `TRUE` (verdadeiro) ou `FALSE` (falso). Os operadores básicos podem verificar a igualdade ou diferença entre valores de variáveis, e é possível combiná-los em expressões mais complexas por meio de operadores de conjunção e disjunção.

```
# Igualdade: '=='
1 == 1
> [1] TRUE
1 == 2
> [1] FALSE

# Diferença: '!='
10 != 10
> [1] FALSE
123 != 254
> [1] TRUE

# Maior: '>'; ou maior e igual: '>='
# E menor: '<'; ou menor e igual: '<='
1542 < 749
> [1] FALSE
450 >= 45
> [1] TRUE

# Conjunção: '&'
10 > 11 & 0 < 1
> [1] FALSE

# Disjunção: '|'
10 > 11 | 0 < 1
```

```
> [1] TRUE
```

2.3 Criando variáveis

Uma grande vantagem de utilizarmos uma linguagem de programação para realizar nossas análise é a possibilidade de salvar os resultados de uma operação na memória. Os resultados salvos podem ser invocados mais tarde, tanto para verificar seus valores quanto para realizar novas operações sobre eles. Para salvar o resultado de uma operação na memória, precisamos criar uma variável dando-lhe um nome qualquer e utilizando o operador de *atribuição*: `=` ou `<-`.

Quando atribuímos a uma variável o resultado de uma operação, o `R` não imprime na tela os valores obtidos. O resultado pode ser visualizado invocando o nome da variável: basta digitar seu nome no console e pressionar `Enter`. Da mesma forma, é possível realizar operações diretamente sobre a variável criada, facilitando o encadeamento de operações.

O nome de uma variável no `R` precisa seguir apenas duas regras: não iniciar por um número e nem conter espaços. Como o nome é arbitrário, é importante utilizar descrições que facilitem lembrar que tipo de informação está contida em determinada variável. Quando uma variável é criada, seu nome e valor aparecem na janela *Environment*.

```
# Criando uma variável com nome 'soma1' e invocando ela em seguida.
soma1 = 10 + 45
soma1
> [1] 55

# Utilizando o valor salvo na variável para fazer uma nova operação.
sub1 = soma1 - 45
sub1
> [1] 10
```

2.4 Tipos de variáveis

O `R` possui suporte para um grande número de *tipos* de variáveis. Vamos nos focar, por enquanto, em três delas: as variáveis do tipo *numérico*, tipo *caractere*, e do tipo *lógico*. Uma variável numérica salva na memória um determinado número, que pode ser inteiro ou real. Uma variável do tipo caractere, por sua vez, salva na memória uma sequência

arbitrária de caracteres, e pode ser utilizado para representar variáveis categoriais, por exemplo. Uma variável do tipo lógico, por fim, só pode ter um de dois valores possíveis: `TRUE` (verdadeiro) ou `FALSE` (falso), e serve para indicar o valor de verdade de uma determinada condição. Uma variável do tipo caractere é criar por meio de aspas simples ou duplas. O tipo de uma variável pode ser visualizada diretamente na janela *Environment* ou por meio da função `mode`.

```
# As variáveis 'soma1' e 'sub1' são numéricas, obviamente.
mode(soma1)
> [1] "numeric"
mode(sub1)
> [1] "numeric"

# Para criarmos variáveis do tipo caractere, usamos aspas.
car1 = 'Curso de introdução ao R' # Com aspas simples...
car2 = "O R é fácil de usar!" # Ou aspas duplas.

car1
> [1] "Curso de introdução ao R"
mode(car1)
> [1] "character"

# Uma variável do tipo lógico é o resultado de uma comparação entre valores.
# Por exemplo, podemos verificar a igualdade entre variáveis com o operador '=='
# Atenção! O primeiro '=' não é sinal de igualdade, mas para atribuição, como
# vimos acima. É o sinal '==' que verifica a igualdade entre as variáveis.
igual = soma1 == sub1
igual
> [1] FALSE
mode(igual)
> [1] "logical"
```

2.5 Funções

Toda operação mais complexa que formos realizar no `R` será por meio de *funções*. As funções do `R`, como qualquer função matemática, pega um conjunto de valores de entrada, realiza uma série de operações bem definidas, e retorna um valor de saída. Para invocar

uma função no R, digitamos o nome da função corretamente (cuidado! O R é sensível a maiúsculas e minúsculas!), abrimos parênteses, digitamos os *argumentos* ou *parâmetros* da função, fechamos parênteses e pressionamos Enter. Argumentos ou parâmetros são, em geral, os valores de entrada sobre os quais queremos aplicar a nossa função, e opções que permitem modificar o comportamento da função, e devem ser separados por vírgulas.

Nós já utilizamos a função mode acima para identificar o tipo da variável. Como todo o resto desta introdução girará em torno de funções específicas para manipulação e visualização de dados, traremos apenas um exemplo de função para concatenar variáveis do tipo caracter: a função paste.

```
# A função 'paste' junta duas variáveis numa só.
# Seu argumentos são as variáveis que queremos concatenar.
paste(car1, car2)
> [1] "Curso de introdução ao R O R é fácil de usar!"
# Essa função possui um parâmetro para determinar
# qual caracter utilizar na concatenação: 'sep'.
# Para determinar esse caracter, acrescentamos o parâmetro
# 'sep' aos argumentos da função, e o definimos com um
# caractere qualquer.
paste(car1, car2, sep='...')
> [1] "Curso de introdução ao R...O R é fácil de usar!"
```

2.6 Estruturas de dados

Uma das características que torna o R uma linguagem de programação poderosa para a análise de dados é o conjunto nativo de *estruturas de dados*. Uma estrutura de dados é um objeto da linguagem de programação que permite armazenar vários valores de maneira organizada, facilitando a manipulação e a realização de operações.

2.6.1 Vetores

A estrutura de dados mais simples do R é o *vetor*. Um vetor é apenas uma sequência de valores com uma ordem definida. Para criar um novo vetor no R, utilizamos a função c. Seus argumentos são os valores que compõem o vetor, separados por vírgulas. É possível utilizar variáveis já criadas dentro de vetores. Nesse caso, o vetor conterá o valor associado àquela variável, e não a própria variável em si. Também é possível construir novos vetores

a partir de vetores já existentes, mas a estrutura do vetor não permite que os vetores permaneçam aninhados: os valores de cada vetor são concatenados num vetor maior.

Um vetor pode conter variáveis de qualquer tipo, mas é importante que todos os valores de um vetor seja do mesmo tipo. Se tentarmos criar um vetor misturando os tipo de variáveis (p.e.: `c(1, 'dois', FALSE)`), o `R` converterá todos os valores para o tipo de variável mais compatível (no exemplo, o número `1` e o lógico `FALSE` serão automaticamente convertidos para caractere).

É possível acessar elementos de um vetor por meio de índices numéricos. No `R`, o primeiro valor de um vetor corresponde ao índice `1`, e assim sucessivamente. Para obtermos um elemento específico, digitamos o nome do vetor, abrimo colchetes, digitamos o número do índice e fechamos os colchetes: `var1[3]`. É importante que o número do índice não seja maior do que o tamanho do vetor, senão será gerada uma mensagem de erro. A função `length` serve para identificar o número de elementos presentes num vetor.

Também é possível obtermos um uma subsequência de valores de um vetor. Para isso,

```
# Vetores criados com a função 'c',
# utilizando valores puros ou associados a variáveis.
numeros1 = c(23, 41, 55)
numeros2 = c(soma1, sub1)
numeros1
> [1] 23 41 55
numeros2
> [1] 55 10
# Se criamos um novo vetor a partir de vetores existentes,
# os vetores de origem são concatenados num vetor meior.
numeros3 = c(numeros1, numeros2)
numeros3
> [1] 23 41 55 55 10
length(numeros3)
> [1] 5
```

2.6.2 Listas

Uma *lista* é uma estrutura de dados que permite agrupar diferentes tipo de variáveis, inclusive outras estruturas com dimensões variadas, num único objeto indexado por chaves arbitrárias. Uma lista pode ser criada por meio da função `list` e seus argumentos são os

objetos e variáveis que comporão a lista. É importante, ainda que opcional, acrescentar o nome da chave que vai fazer referência aos objetos da lista, pois isso facilita acessá-los posteriormente. O acesso aos valores componentes de uma lista é feito por meio do sinal `$`.

```
# Criamos uma nova lista com o comando 'list', e os arugmentos são
# organizados da seguinte forma:
# chave = valor
# separados por vírgulas
lista1 = list(num1=numeros3, var1=3.14, car1=car1, car2='Listas são estranhas')
lista1
> $num1
> [1] 23 41 55 55 10
>
> $var1
> [1] 3.14
>
> $car1
> [1] "Curso de introdução ao R"
>
> $car2
> [1] "Listas são estranhas"
# Para acessar um valor específico da lista, usamos `$` após seu nome
# e com o nome da chave referente ao valor de interesse
lista1$num1
> [1] 23 41 55 55 10
```

2.6.3 *Data frames*

A estrutura de dados que mais utilizaremos ao longo do curso é o *data frame*. Um *data frame* é a estrutura padrão na qual o `R` armazena banco de dados, organizando as variáveis em colunas e os casos observados em linhas. Como a lista, o *data frame* pode conter variáveis de todos os tipos, mas ele exige que que todos os vetores de variáveis tenham o mesmo tamanho.

Um objeto do tipo *data frame* é criado por meio da função `data.frame`, e seus argumentos seguem o mesmo padrão da criação de listas: `nomeDaVariável = vetorDaVariável`, separados entre si por vírgulas. Tome cuidado para verificar se o tamanho dos vetores das variáveis possuem o mesmo tamanho, senão o comando gerará um erro!

Assim como na lista, utilizamos o operador `$` para obtermos uma variável específica pelo seu nome. Os nomes das variáveis de um *data frame* pode ser obtido por meio da função `names` – e pode ser alterado utilizando essa mesma função.

Como um *data frame* é uma organização bidimensional de dados, é possível usar os colchetes para obter uma coluna, linha ou conjunto de casos específicos. Nesse caso, é necessário especificar dois números dentro dos colchetes, separando-os por vírgulas. O primeiro número se refere à linha, ou seja, à observação registrada na linha indicada. O segundo número se refere à coluna, ou seja, à variável registrada na coluna indicada. Por exemplo, `dados[3, 2]` irá retornar o valor da segunda variável para a terceira observação.

Se desejamos recuperar todos os valores correspondentes a um caso ou a uma variável, basta deixar um dos números sem preencher. Por exemplo, `dados[10, ]` retorna os valores de todas as variáveis para o caso 10, enquanto `dados[, 10]` retorna os valores de todos os casos para a variável 10.

```
# Começamos definindo alguns vetores de mesmo tamanho,
# representando os valores de variáveis para as mesmas observações.
v1 <- c('Agronomia', 'Biologia', 'Engenharia', 'Letras')
v2 <- c(453, 319, 307, 248)
v3 <- c('Bacharelado', 'Licenciatura', 'Bacharelado', 'Licenciatura')

# Por fim, construímos o data.frame.
dados.cursos <- data.frame(curso=v1, n.alunos=v2, tipo.curso=v3)
dados.cursos
>        curso n.alunos   tipo.curso
> 1  Agronomia      453  Bacharelado
> 2   Biologia      319 Licenciatura
> 3 Engenharia      307  Bacharelado
> 4     Letras      248 Licenciatura

# Podemos explorar os nomes das variáveis...
names(dados.cursos)
> [1] "curso"      "n.alunos"   "tipo.curso"

# ... e acessar variáveis específicas pelo nome utilizando `$`
dados.cursos$tipo.curso
> [1] Bacharelado  Licenciatura Bacharelado  Licenciatura
> Levels: Bacharelado Licenciatura
```

```
# Utilizando colchetes, podemos recuperar casos, variáveis ou
# subconjuntos específicos de dados.
# P.e.: Quantos alunos tem o curso de Agronomia?
# Agronomia é o caso da linha 1,
# número de alunos é a variável da coluna 2.
dados.cursos[1, 2]
> [1] 453

# Podemos recuperar uma variávels específica, como
# quando utilizamos o `$`.
# A coluna 1 indica o nome do curso.
# O comando a seguir é equivalente à `dados.cursos$curso`
dados.cursos[, 1]
> [1] Agronomia  Biologia   Engenharia Letras
> Levels: Agronomia Biologia Engenharia Letras

# Ou podemos recuperar o vetor com todos os valores
# para uma observação específica.
# Por exemplo, o curso de Letras está na linha 4.
dados.cursos[4, ]
>    curso n.alunos   tipo.curso
> 4 Letras      248 Licenciatura
```

3 Carregando bancos de dados no R

Na maioria dos casos, ao invés de entrarmos com os dados diretamente no R, eles já estarão organizados na forma de um banco de dados em um arquivo externo salvo no computador local ou remoto. Podemos importar vários tipos de arquivos de banco de dados, mas, por padrão, o R prefere arquivos em formato de texto puro, como o CSV (*Comma Separated Value*, valores separados por vírgulas). A maioria dos programas que trabalham com bancos de dados ou planilhas, como o SPSS e o Excel, permitem salvar seus arquivos nesse formato, o que facilita trabalhar com eles dentro do R.

3.1 Utilizando o console

Vamos iniciar carregando um banco de dados por meio do console. A principal função utilizada no `R` para carregar arquivos do tipo CSV é `read.csv` – uma extensão da função `read.table` que lê arquivos de textos genéricos.

O primeiro argumento da função é o nome do arquivo, que deve ser escrito entre aspas. O segundo argumento, `header`, indica se o arquivo contém um cabeçalho com os nomes das variáveis (`TRUE` (valor padrão) se tiver, `FALSE` se não tiver). O parâmetro `sep` indica qual é o caracter que demarca a separação entre valores. Por padrão, o separador é a vírgula, mas há arquivos que usam ponto-e-vírgula ou tabulações. Por fim, o parâmetro `stringAsFactors` indica se as variáveis qualitativas devem ser importadas como variáveis do tipo caracter (`FALSE`) ou do tipo fator (`TRUE`).

A título de exemplo, vamos trabalhar com uma base de dados da OMS (Organização Mundial de Saúde) que contém várias informações importantes sobre desenvolvimento social e econômico de vários países (o arquivo original foi utilizado no curso *The Analytics Edge*, do MITx, e disponível no EdX).

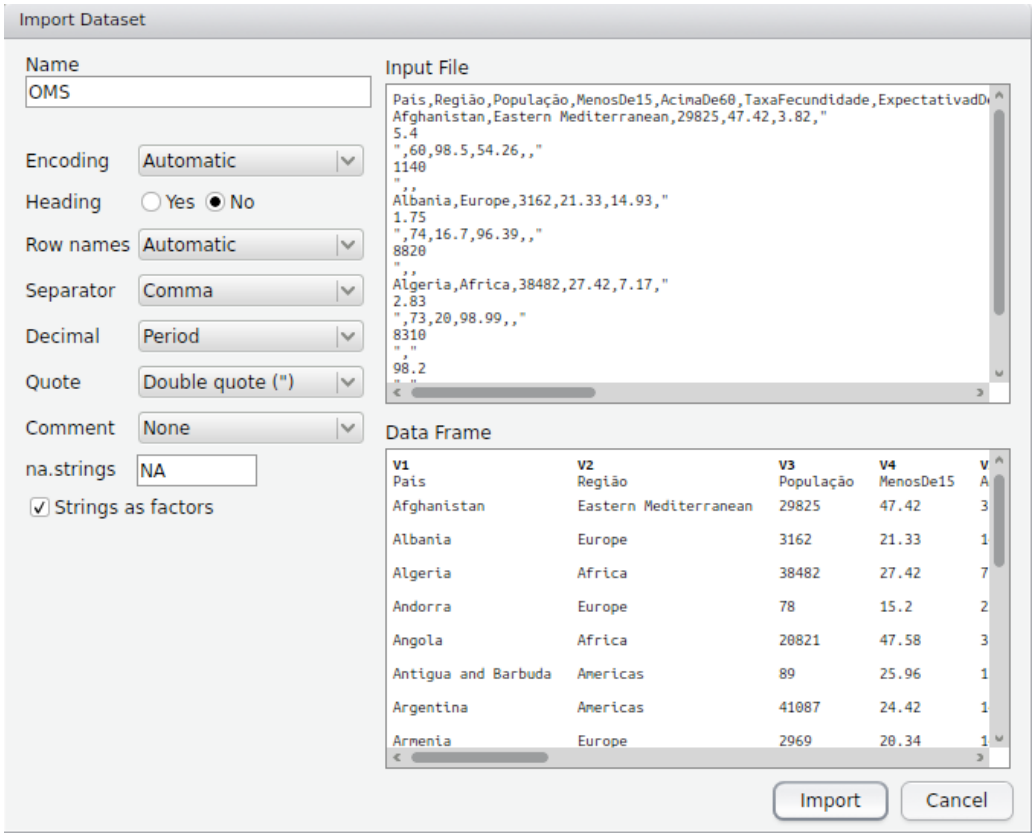
```
# Ao carregar um arquivo no R, não esqueça de atribuir os resultados
# a uma variável para salvá-lo na memória.
# Se o arquivo estiver com cabeçalho e separado por vírgulas, não é
# necessário especificar os outros parâmetros.
oms = read.csv('~/Dropbox/OMS.csv')

# Podemos verificar os nomes das variáveis...
names(oms)
>  [1] "X"                          "País"
>  [3] "Região"                     "População"
>  [5] "MenosDe15"                  "AcimaDe60"
>  [7] "TaxaFecundidade"            "ExpectativaDeVida"
>  [9] "MortalidadeInfantil"        "UsuáriosCelular"
> [11] "TaxaAlfabetização"          "PIB"
> [13] "MatrículaEscolaPrimáriaMasc" "MatrículaEscolaPrimáriaFem"

# E quantas variáveis e observações temos.
dim(oms)
> [1] 194  14
```

3.2 Utilizando a interface do `RStudio`

O `RStudio` permite importar dados no formato de texto puro, como arquivos CSV, diretamente pela interface gráfica. Na janela *Environment*, no canto superior direito, o ícone *Import Dataset* oferece duas opções: de um arquivo local (*From local file...*) ou de um arquivo remoto, pela Internet (*From Web URL...*). Independente da opção, assim que o arquivo é carregado, uma janela de diálogo aparece para configurar corretamente a importação dos dados.



Na coluna à esquerda há várias opções para configurar corretamente a importação. À direita, a janela mostra os dados brutos do arquivo na parte superior, e o resultado final da importação na parte inferior.

As opções dadas são:

- *Name*: indica o nome da variável que guardará na memória o banco de dados;
- *Encoding*: indica a codificação do arquivo (a opção *Automatic* costuma funcionar na maioria dos casos);

- *Heading*: indica se há (*Yes*) ou não (*No*) um cabeçalho no arquivo;
- *Row names*: opções para indicar nomes para as linhas (o padrão *Automatic* utiliza o número da linha);
- *Separator*: indica qual caractere separa os valores no arquivo;
- *Decimal*: indica qual caractere separa casas decimais nos valores numéricos;
- *Quote*: indica qual caractere demarca variáveis qualitativas;
- *Comment*: indica qual caracter demarca comentários no arquivo, se houver;
- *na.string*: indica como tratar valores faltantes (*missing*, o padrão do `R` é `NA`);
- *String as factors*: indica se as variáveis categoriais devem ser carregadas como fatores ou como vetores de caracteres.

3.3 Salvando bancos de dados como arquivos CSV

É importante notar que, quando um banco de dados é carregado dentro do `R`, qualquer manipulação ou alteração dos dados não é salva no arquivo original. O `R` apenas carrega os dados na memória, mas não modifica diretamente o arquivo. Por isso, se forem feitas alterações que precisam ser salvas, é necessário utilizar a função `write.csv` para criar um novo arquivo de dados a partir do *data frame* manipulado dentro do `R`.

```
# Para salvar um data.frame como arquivo CSV,
# basta utilizar a função 'write.csv', indicar
# o nome do objeto no ambiente (sem aspas) e o
# nome do arquivo a ser salvo (entre aspas).
# Cuidado! Não utilize o nome do arquivo original,
# senão o R salvará o novo arquivo por cima da
# versão antiga!
write.csv(oms, 'novoOMS.csv')
```

4 Manipulando e explorando bancos de dados

Para visualizar os dados importados dentro do `RStudio`, basta clicar no nome da variável na janela *Environment*. Uma nova aba será aberta no lado esquerdo da janela, permitindo visualizar, filtrar e fazer buscas no banco de dados – mas não é possível editar nenhum valor.

Para visualizarmos o banco de dados no console, podemos chamá-lo diretamente pelo nome da variável (`oms` em nosso exemplo), mas podemos também visualizar somente alguns

casos, o que é particularmente útil em banco de dados com muitas observações. Para isso, utilizamos a função **head**, para os primeiros casos, ou **tail** para os casos finais.

```
# Visualizando os 5 primeiros casos.
head(oms, n=5)
>   X        País                Região População MenosDe15 AcimaDe60
> 1 1 Afghanistan Eastern Mediterranean     29825     47.42      3.82
> 2 2     Albania                Europe      3162     21.33     14.93
> 3 3     Algeria                Africa     38482     27.42      7.17
> 4 4     Andorra                Europe        78     15.20     22.86
> 5 5      Angola                Africa     20821     47.58      3.84
>   TaxaFecundidade ExpectativaDeVida MortalidadeInfantil UsuáriosCelular
> 1            5.40                60                98.5           54.26
> 2            1.75                74                16.7           96.39
> 3            2.83                73                20.0           98.99
> 4              NA                82                 3.2           75.49
> 5            6.10                51               163.5           48.38
>   TaxaAlfabetização  PIB MatrículaEscolaPrimáriaMasc
> 1                NA 1140                         NA
> 2                NA 8820                         NA
> 3                NA 8310                       98.2
> 4                NA   NA                       78.4
> 5              70.1 5230                       93.1
>   MatrículaEscolaPrimáriaFem
> 1                         NA
> 2                         NA
> 3                       96.4
> 4                       79.4
> 5                       78.2

# Visualizando os 3 últimos casos.
tail(oms, n=3)
>       X     País                Região População MenosDe15 AcimaDe60
> 192 192    Yemen Eastern Mediterranean     23852     40.72      4.54
> 193 193   Zambia                Africa     14075     46.73      3.95
> 194 194 Zimbabwe                Africa     13724     40.24      5.68
>     TaxaFecundidade ExpectativaDeVida MortalidadeInfantil UsuáriosCelular
```

```
> 192            4.35                64                60.0            47.05
> 193            5.77                55                88.5            60.59
> 194            3.64                54                89.8            72.13
>     TaxaAlfabetização  PIB MatrículaEscolaPrimáriaMasc
> 192              63.9 2170                         85.5
> 193              71.2 1490                         91.4
> 194              92.2   NA                           NA
>     MatrículaEscolaPrimáriaFem
> 192                       70.5
> 193                       93.9
> 194                         NA
```

4.1 Selecionando casos e variáveis

Como o banco de dados é exportado como um *data frame*, todas as operações que vimos podem ser utilizadas, como o operador `$` para selecionar uma variável ou os colchetes para "recortar" o banco de dados como desejamos. Quando queremos extrair um conjunto específicos de vários casos ou variáveis, podemos utilizar os colchetes, mas especificando um conjunto de números com a função `c`.

Por exemplo, se queremos visualizar todos os valores da primeira e da décima variável, escrevemos `oms[, c(1, 10)]`. Da mesma forma, se queremos visualizar os três primeiros casos, escrevemos `oms[c(1, 2, 3), ]`. Se desejamos visualizar todo um subconjunto de casos e variáveis com exceção de alguns, basta acrescentar um `-` na frente do vetor: `oms[-c(1, 2, 3), -c(1, 10)]`, por exemplo, retornará todo o banco de dados, com excessão dos três primeiros casos e excluindo as variáveis 1 e 10.

Quando queremos vários casos ou variáveis em sequência, podemos indicar somente o número inicial e final, separados por `:`. Por exemplo: `oms[1:10, ]` mostrará os valores de todas as variáveis para os dez primeiros casos. Por fim, podemos filtrar casos ou variáveis utilizando condições especificadas por operadores lógicos. Operadores lógicos permitem verificar determinadas condições e retornam sempre valores lógicos (`TRUE` (verdadeiro) ou `FALSE` (falso)). Esse tipo de operação é útil para selecionar apenas um subconjunto dos dados com alguma característica de interesse. Por exemplo, nosso banco de dados `oms` possui uma variável que indica à qual região pertence o país representado na linha. Podemos utilizar essa variável e operadores lógicos para selecionar somente os países africanos: `oms[oms$Região == "Africa", ]`, ou seja, selecionar todos os casos (países,

no banco de dados `oms`) cuja variável `Região` for igual à `"Africa"`. É importante lembrar que, nesse caso, o valor da variável é categorial e por isso ela foi indicada entre aspas.

```
# Recortando o banco de dados utilizando a notação
# de colchetes. Para não comprometer a visualização dos
# resultados, vamos sempre selecionar um número pequeno de variáveis
# e de casos.
# 1) Visualizando valores de casos específicos em duas variáveis
# (nesse caso, nomes dos países e suas populações, em milhares).
oms[c(1, 3, 5), c(1, 3)]
>   X                Região
> 1 1 Eastern Mediterranean
> 3 3                Africa
> 5 5                Africa

# 2) Visualizando casos e variáveis seguidos
oms[10:12, 2:4]
>          País   Região População
> 10    Austria   Europe      8464
> 11 Azerbaijan   Europe      9309
> 12    Bahamas Americas       372

# 3) Utilizando operadores lógicos para selecionar casos
# (nesse exemplo, especificamos que queremos todos os países
# da região 'Africa' E cuja população é maior do que 100 milhões).
oms[oms$Região == 'Africa' & oms$População > 100000, c(1, 3, 11)]
>       X Região TaxaAlfabetização
> 125 125 Africa              61.3
```

4.2 Informações sumárias sobre as variáveis

Visualizar todos os valores de uma variável, ou mesmo um subconjunto deles, não permite ter uma boa percepção sobre as principais características dos dados. Uma saída é utilizar estatísticas descritivas. A função `summary` pode ser utilizada para apresentar um conjunto de estatísticas importantes sobre uma variável numérica: valor mínimo, primeiro quartil, mediana, média, terceiro quartil e valor máximo. Se for utilizado para uma variável categorial, a função retorna a contagem de casos em cada um dos níveis da variável. Se

há casos faltantes (*missing*), a função também indica sua contagem.

```
# A função 'summary' pode ser aplicada diretamente sobre
# o banco de dados. Nesse caso, ele retornará os dados sumários
# todas as variáveis. Para não prejudicarmos a visualização
# dos resultados, vamos apenas aplicar a função sobre variáveis
# específicas.
# 1) No caso de uma variável numérica:
summary(oms$População)
>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
>       1    1696    7790   36360   24540 1390000

# 2) E no caso de uma variável categorial:
summary(oms$Região)
>                Africa              Americas Eastern Mediterranean
>                    46                    35                    22
>                Europe       South-East Asia       Western Pacific
>                    53                    11                    27
```

Os resultados obtidos pela função `summary` podem ser obtidos separadamente utilizando outras funções:

- *Valor mínimo*: `min(var)`
- *Valor máximo*: `max(var)`
- *Mediana*: `median(var)`
- *Média*: `mean(var)`
- *Desvio padrão*: `sd(var)`
- *Contagem de casos*: `table(var)`

Mais detalhes sobre funções para análise exploratória de dados serão vistas no próximo módulo.

Por fim, podemos querer localizar observações que atendam determinadas características. A função `which` permite localizar entradas no banco de dados a partir de uma regra lógica. Se a condição é encontrar os valores mínimos ou máximos de uma variável, as funções `which.max` e `which.min` retornam os índices correspondentes ao valores máximo e mínimo, respectivamente.

```
# Usando a função which.max e which.min, podemos
# descobrir algumas informações interessantes no
# nosso banco de dados.
# Por exemplo: qual o país com menor número de habitantes?
which.min(oms$População)
> [1] 126

# Lembre-se que o valor retornado é o índice da observação
oms$País[126]
> [1] Niue
> 194 Levels: Afghanistan Albania Algeria Andorra ... Zimbabwe

# Outro exemplo: qual o país com maior índice de alfabetização?
which.max(oms$TaxaAlfabetização)
> [1] 44
oms$País[44]
> [1] Cuba
> 194 Levels: Afghanistan Albania Algeria Andorra ... Zimbabwe
```

4.3 Criando subconjuntos de dados

Já vimos como utilizar a notação de colchetes com índices e expressões para recuperar valores específicos do banco de dados. Muitas vezes, porém, é necessário criar novos bancos de dados que são subconjuntos do banco de dados original. Uma opção para facilitar esse trabalho é utilizar a função `subset`, que permite criar um novo subconjunto a partir do banco de dados original e regras de inclusão. Os argumentos necessários são o nome do banco de dados original e a regra para a formação do subconjunto.

```
# Por exemplo, podemos utilizar a função 'subset'
# para formar um banco de dados contendo somente
# os países das Américas.
# Nota: lembre de que a regra de inclusão precisa
# utiliza operadores lógicos, como '==', '!='
# 1) Subconjunto dos países americanos:
americas = subset(oms, oms$Região == 'Americas')
```

```r
# 2) Subconjunto dos países com 90% ou maior proporção de alfabetizados:
alfabetizados = subset(oms, oms$TaxaAlfabetização >= 90)
```

4.4 Aplicando funções a diversos subconjuntos

A função `subset` é útil quando queremos explorar um subconjunto de dados. Mas, muitas vezes, queremos avaliar simultaneamente uma mesma características em vários subconjuntos dos dados. Uma opção seria criar um banco de dados para cada subconjunto, aplicar a função nos novos bancos de dados, e avaliar os resultados. O mesmo resultado pode ser obtido muito mais rapidamente utilizando a função `tapply`. Ela exige três argumentos: uma variável que queremos analisar, um variável de indique a qual subconjunto cada valor pertence, e a função de interesse. É possível acrescentar argumentos extras à função `tapply`, que serão passadas para a função aplicada sobre a variável.

```r
# Verificamos antes qual o país com menor número de
# habitantes na base toda. E se quiséssemos saber as
# maiores populações em cada região da base de dados?
tapply(oms$População, oms$Região, max)
>                Africa              Americas Eastern Mediterranean
>                169000                318000                179000
>                Europe       South-East Asia       Western Pacific
>                143000               1240000               1390000

# E qual é a média de expectativa de vida em cada região?
# Nota: aqui utilizamos um argumento extra, `na.rm=TRUE`
# para que a média seja calculada mesmo na presença de
# valores faltantes (missing).
tapply(oms$ExpectativaDeVida, oms$Região, mean, na.rm=TRUE)
>                Africa              Americas Eastern Mediterranean
>              57.95652              74.34286              69.59091
>                Europe       South-East Asia       Western Pacific
>              76.73585              69.36364              72.33333
```