Aula 5: Importação e exportação de dados

Profa. Yana Borges

Março de 2022

Pode-se entrar com dados no R de diferentes formas. O formato mais adequado vai depender do tamanho do conjunto de dados, se os dados já existem em outro formato para serem importados ou se serão digitados diretamente no R.

Existem diversas maneiras de se ler bancos de dados no R, podendo ser esses bancos de dados de diversos formatos. Os formatos mais comuns são ".xls", ".xlsx", ".txt", ".csv", entre outros.

1. Diretório

Antes de pensar em conjunto de dados, precisamos introduzir a noção de diretório, que nada mais é do que a pasta na qual você está trabalhando, onde seus dados estão salvos e etc. Em geral, o R entende que seu diretório é a pasta na qual você salvou seu script. Para descobrir o seu diretório use o comando:

```
getwd()
```

```
## [1] "D:/1-IFAM/Aranoua/Introducao-ciencia-de-dados-R/Aula5-leitura-tratamento-de-dados"
```

Ele informará qual é sua pasta de trabalho. Isso significa que todos os arquivos que estiverem salvos nessa pasta serão facilmente encontrados pelo R. Para alterar seu diretório, caso seja necessário, você pode usar o comando:

```
setwd("C:\\Users\\yanab\\Desktop\\Aula5")
```

Entre parênteses, temos por exemplo, um diretório na pasta "Aula5", que por sua vez está na área de trabalho do computador (Desktop). Também existe um método alternativo de se alterar o diretório no RStudio, aperte **"Ctrl+Shift+H"** e escolha seu novo diretório na janela que surgirá na sua tela.

2. Entrada de dados diretamente no R

2.1 Vetores

A forma mais básica de entrada de dados no R é através da função `c()`, como já vimos. A partir dela pode se criar os outros tipos de objetos como listas e data frames.

- `c()`
- `rep()`
- `seq()` A partir destas funções básicas podemos criar objetos de classes mais específicas com
- `matrix()`
- `list()`
- `data.frame()`

2.2 Entrada via teclado

2.2.1 Usando a função scan()

Esta função lê dados diretamento do console, isto é, coloca o R em modo prompt onde o usuário deve digitar cada dado seguido da tecla **Enter**. Para encerrar a entrada de dados basta digitar **Enter** duas vezes consecutivas.

Veja o seguinte resultado:

```
y <- scan()

1: 11
2: 24
3: 35
4: 29
5: 39
6: 47
7:
Read 6 items
```

Os dados também podem ser digitados em sequência, desde que separados por um espaço,

```
y <- scan()

1: 11 24
3: 35 29
5: 39 47
7:
Read 6 items
```

Este formato é mais ágil que o anterior (com `c()`, por exemplo) e é conveniente para digitar vetores longos. Esta função pode também ser usada para ler dados de um arquivo ou conexão, aceitando inclusive endereços de URLs (endereços da web).

Por padrão, a função `scan()` aceita apenas valores numéricos como entrada (lembre-se que vetores só podem ter elementos da mesma classe). Para alterar a classe de objeto de entrada, precisamos especificar o argumento what de `scan()`. Por exemplo, para entrar com um vetor de caracteres, fazemos

```
x <- scan(what = "character")

1: a
2: b
3: c
4:
Read 3 items
```

Outras classe possíveis para o argumento what são: `logical`, `integer`, `numeric`, `complex`, `character`, `raw` e `list`.

2.2.2 Uso da função `readLines()`

Esta função é particularmente útil para ler entradas na forma de texto (*strings*). Por exemplo, para ler uma linha a ser digitada na tela do R, siga o comando abaixo e digite o texto indicado. Ao terminar pressione a tecla **Enter** e o texto será armazenado no objeto texto.

```
texto <- readLines(n=1)
```

```
Estou digitando no console
```

Exercícios tópico 2

1. Usando a função `scan()` crie objetos para armazenar os seguintes valores:

- 19, 13, 19, 23, 18, 20, 25, 14, 20, 18, 22, 18, 23, 14, 19
- joaquina, armação, praia brava, praia mole, morro das pedras
- TRUE, TRUE, FALSE, FALSE, TRUE

2. Usando a função `readLines()` crie objetos para armazenar:

- Joaquina, armação, praia brava, praia mole, morro das pedras

3 Arquivos Excel

Para ler dados de uma planilha Excel é necessário previamente instalar um pacote específico,uma vez que essa funcionalidade não está inicialmente disponível no R. Para isso recomendamos instalar o pacote `readxl` usando o comando abaixo ou algum dos outros métodos de instalação citados anteriormente. Lembre-se que sempre que você quiser utilizar um pacote que já foi instalado é necessário apenas carregá-lo.

```
install.packages("readxl")
```

```
library(readxl)
```

No pacote `readxl` existe uma função chamada `read_excel()` que, como o nome sugere, tem o objetivo de ler arquivos em Excel que estão salvos no seu diretório. Para trabalhar com um banco de dados no R não basta simplesmente excutar um comando, é preciso também armazenar os dados que você deseja analisar na memória do R. Para isso precisamos criar uma variável. O comando abaixo lê o banco de dados `precipdiaria` (que deve estar salvo no diretório de trabalho) e o armazena na variável `dados`.

```
dados <- read_excel("precipdiaria.xlsx",
                            sheet = "Planilha3")
dados2 <- data.frame(read_xlsx("precipdiaria.xlsx"))
```

Como pode ser visto acima, a função `read_excel` é bem simples de ser usada, basta colocar o nome do arquivo com a extensão *.xlsx* entre aspas e armazenar na variável que deseja.

Note que no RStudio, irá aparecer na parte de histórico a variável que você criou, clicando nela você pode visualizar os dados foram lidos. Outra alternativa acessar `Environment` ou usar o comando abaixo, que também consta em `History`:

```
View(dados)
```

4 Arquivos .csv

O método mais comum de importação de dados para o R, é utilizando a função `read.table()`. Para importar um arquivo `.csv` faça:

```
covid <- read.table("covidateFev2021.csv",
                    header = TRUE,
                    sep = ";", dec = ",")
```

Argumentos:

- `"covidateFev2021.csv"`: nome do arquivo. (Considerando que o arquivo covidateFev2021.csv está no diretório).
- `header = TRUE`: significa que a primeira linha do arquivo deve ser interpretada como os nomes das colunas
- `sep = ";"`: o separador de colunas (também pode ser `"\t"` para tabulação e `""` para espaços)
- `dec = ","`: o separador de decimais

Para conferir a estrutura dos dados importados, usamos a função `str()` que serve para demonstrar a estrutura de um objeto, como o nome das colunas e suas classes:

```
str(dados)
```

```
## tibble[,11] [13,825 x 11] (S3: tbl_df/tbl/data.frame)
##  $ Estacao                  : num [1:13825] 82331 82331 82331 82331 82331 ...
##  $ Data                     : POSIXct[1:13825], format: "2000-06-01" "2000-06-01" ...
##  $ Hora                     : num [1:13825] 0 1200 0 1200 0 1200 0 1200 0 1200 ...
##  $ Precipitacao             : num [1:13825] NA 0 NA 6 NA 0 NA 0 NA 2.9 ...
##  $ TempMaxima               : num [1:13825] 32.4 NA 32.1 NA 32.9 NA 32.6 NA 32.3 NA ...
##  $ TempMinima               : num [1:13825] NA 22.5 NA 22.5 NA 21.5 NA 21.5 NA 22.5 ...
##  $ Insolacao                : num [1:13825] 3.1 NA 4.6 NA 8.8 NA 4.6 NA 10.2 NA ...
##  $ Evaporacao Piche         : num [1:13825] 2.4 NA 2 NA 1.8 NA 2 NA 1.6 NA ...
##  $ Temp Comp Media          : num [1:13825] 25.9 NA 25.9 NA 26.9 ...
##  $ Umidade Relativa Media   : num [1:13825] 87 NA 87.5 NA 76.8 ...
##  $ Velocidade do Vento Media: num [1:13825] 0.667 NA 0.667 NA 0.333 ...
```

```
str(covid)
```

```
## 'data.frame':    17530 obs. of  17 variables:
##  $ regiao               : chr  "Brasil" "Brasil" "Brasil" "Brasil" ...
##  $ estado               : chr  "" "" "" "" ...
##  $ municipio            : logi  NA NA NA NA NA NA ...
##  $ coduf                : int  76 76 76 76 76 76 76 76 76 76 ...
##  $ codmun               : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ codRegiaoSaude       : logi  NA NA NA NA NA NA ...
##  $ nomeRegiaoSaude      : logi  NA NA NA NA NA NA ...
##  $ data                 : chr  "2020-02-25" "2020-02-26" "2020-02-27" "2020-02-28" ...
##  $ semanaEpi            : int  9 9 9 9 9 10 10 10 10 10 ...
##  $ populacaoTCU2019     : int  210147125 210147125 210147125 210147125 210147125 210147125 21014712
##  $ casosAcumulado       : int  0 1 1 1 2 2 2 2 3 7 ...
##  $ casosNovos           : int  0 1 0 0 1 0 0 0 1 4 ...
##  $ obitosAcumulado      : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ obitosNovos          : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Recuperadosnovos     : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ emAcompanhamentoNovos : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ interior.metropolitana: logi  NA NA NA NA NA NA ...
```

Podemos também visualizar algumas linhas iniciais e finais do objeto importado através de duas funções auxiliares:

```
head(dados)
```

```
## # A tibble: 6 x 11
```

```
##   Estacao Data                 Hora Precipitacao TempMaxima TempMinima Insolacao
##     <dbl> <dttm>              <dbl>        <dbl>      <dbl>      <dbl>     <dbl>
## 1   82331 2000-06-01 00:00:00     0           NA       32.4       NA         3.1
## 2   82331 2000-06-01 00:00:00  1200            0       NA         22.5      NA
## 3   82331 2000-06-02 00:00:00     0           NA       32.1       NA         4.6
## 4   82331 2000-06-02 00:00:00  1200            6       NA         22.5      NA
## 5   82331 2000-06-03 00:00:00     0           NA       32.9       NA         8.8
## 6   82331 2000-06-03 00:00:00  1200            0       NA         21.5      NA
## # ... with 4 more variables: Evaporacao Piche <dbl>, Temp Comp Media <dbl>,
## #   Umidade Relativa Media <dbl>, Velocidade do Vento Media <dbl>
```

```
tail(dados)
```

```
## # A tibble: 6 x 11
##   Estacao Data                 Hora Precipitacao TempMaxima TempMinima Insolacao
##     <dbl> <dttm>              <dbl>        <dbl>      <dbl>      <dbl>     <dbl>
## 1   82331 2019-05-03 00:00:00  1200          7.7       NA         25        NA
## 2   82331 2019-05-04 00:00:00     0         NA         32.1       NA         7
## 3   82331 2019-05-04 00:00:00  1200          5.3       NA         25.5      NA
## 4   82331 2019-05-05 00:00:00     0         NA         33.7       NA         9.5
## 5   82331 2019-05-05 00:00:00  1200          0.3       NA         25.7      NA
## 6   82331 2019-05-06 00:00:00  1200          0         NA         26.9      NA
## # ... with 4 more variables: Evaporacao Piche <dbl>, Temp Comp Media <dbl>,
## #   Umidade Relativa Media <dbl>, Velocidade do Vento Media <dbl>
```

Para maiores informações consulte a documentação desta função com `read.table()`. Embora `read.table()` seja provavelmente a função mais utilizada existem outras que podem ser úteis e determinadas situações:

- `read.fwf()` é conveniente para ler fixed *width formats*
- `read.fortran()` é semelhante à anterior porém usando o estilo Fortran de especificação das colunas
- `read.csv()`, `read.csv2()`, `read.delim()` e `read.delim2()`: estas funções são praticamente iguais a `read.table()` porém com diferentes opções padrão. Em geral (mas não sempre) dados em formato csv usados no Brasil são lidos diretamente com `read.csv2()`.

5 Arquivos web

As funções permitem ainda ler dados diretamente disponíveis na web. Por exemplo, os dados do exemplo poderiam ser lidos diretamente com o comando a seguir, sem a necessidade de copiar primeiro os dados para algum local no computador do usuário:

```
petroleo <- read.csv2("https://www.gov.br/anp/pt-br/centrais-de-conteudo/dados-abertos/
 arquivos/ie/petroleo/importacoes-exportacoes-petroleo-2000-2022.csv")
petroleo[1:5,1:5]
```

```
##   ï..ANO MÃŠS   PRODUTO OPERAÃ.ÃfO.COMERCIAL IMPORTADO...EXPORTADO
## 1   2000  ABR PETRÃ"LEO         EXPORTAÃ‡ÃfO          0.000000e+00
## 2   2000  AGO PETRÃ"LEO         EXPORTAÃ‡ÃfO          0.000000e+00
## 3   2000  DEZ PETRÃ"LEO         EXPORTAÃ‡ÃfO          4.199504e+05
## 4   2000  FEV PETRÃ"LEO         EXPORTAÃ‡ÃfO          6.853779e-02
## 5   2000  JAN PETRÃ"LEO         EXPORTAÃ‡ÃfO          0.000000e+00
```

6 Carregando dados já disponíveis no R

O `R` já possui alguns conjuntos de dados que estão disponíeis logo após a instalação. Estes dados são também objetos que precisam ser carregados para ficarem disponiveis para o usuário. Normalmente, estes conjuntos

de dados são para uso de exemplo de funções.

Para carregar conjuntos de dados que são disponibilizados com o R, use o comando `data()`. Por exemplo, abaixo mostramos como carregar o conjunto `mtcars` que está no pacote `datasets`.

```
## Objetos criados até o momento nesta aula
ls()
```

```
## [1] "covid"    "dados"    "dados2"   "petroleo" "texto"
```

```
## Carrega a base de dados mtcars
data(mtcars)
## Note como agora o objeto mtcars fica disponível na sua área de
## trabalho
ls()
```

```
## [1] "covid"    "dados"    "dados2"   "mtcars"   "petroleo" "texto"
```

```
str(mtcars)
```

```
## 'data.frame':    32 obs. of  11 variables:
##  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
##  $ disp: num  160 160 108 258 360 ...
##  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
##  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
##  $ qsec: num  16.5 17 18.6 19.4 17 ...
##  $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
##  $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
##  $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
##  $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

As bases de dados também possuem páginas de documentação para explicar o que são os dados e as colunas correspondentes. Para ver o que são os dados do mtcars por exemplo, veja `?mtcars`.

O conjunto `mtcars` é disponibilizado prontamente pois faz parte do pacote `datasets`, que por padrão é sempre carregado na inicialização do R. No entanto, existem outros conjuntos de dados, disponibilizados por outros pacotes, que precisam ser carregados para que os dados possam ser disponibilizados. Por exemplo, os dados do objeto topo são do pacote `MASS`. Se tentarmos usar o comando `data()` sem antes chamar o pacote que o contém o R não o encontrará:

```
data(topo)
```

```
## Warning in data(topo): data set 'topo' not found
```

Portanto, precisamos primeiro carregar o pacote `MASS` com `library()` ou `require()` e então carregar o objeto `topo`

```
require(MASS)
```

```
## Loading required package: MASS
```

```
data(topo)
ls()
```

```
## [1] "covid"    "dados"    "dados2"   "mtcars"   "petroleo" "texto"    "topo"
```

```
str(topo)
```

```
## 'data.frame':    52 obs. of  3 variables:
##  $ x: num  0.3 1.4 2.4 3.6 5.7 1.6 2.9 3.4 3.4 4.8 ...
```

```
##  $ y: num  6.1 6.2 6.1 6.2 6.2 5.2 5.1 5.3 5.7 5.6 ...
##  $ z: int  870 793 755 690 800 800 730 728 710 780 ...
```

A função `data()` pode ainda ser usada para listar os conjutos de dados disponíveis.

```
data()
```

e também pode ser útil paa listar os conjuntos de dados disponíveis para um pacote específico, por exemplo

```
data(package = "nlme")
```

7 Importando dados de outros programas

É possível ler dados diretamente de outros formatos que não seja texto (ASCII). Isto em geral é mais eficiente e requer menos memória do que converter para formato texto. Há funções para importar dados diretamente de EpiInfo, Minitab, S-PLUS, SAS, SPSS, Stata, Systat e Octave. Além disto é comum surgir a necessidade de importar dados de planilhas eletrônicas. Muitas funções que permitem a importação de dados de outros programas são implementadas no pacote `foreign`.

A seguir listamos algumas (não todas!) destas funções:

- `read.dbf()` para arquivos DBASE
- `read.epiinfo()` para arquivos .REC do Epi-Info
- `read.mtp()` para arquivos "Minitab Portable Worksheet" read.S() para arquivos do S-PLUS, e `restore.data()` para "dumps" do S-PLUS
- `read.spss()` para dados do SPSS
- `read.systat()` para dados do SYSTAT
- `read.dta()` para dados do STATA
- `read.octave()` para dados do OCTAVE (um clone do MATLAB)
- Para dados do SAS há ao menos duas alternativas:
  - O pacote foreign disponibiliza `read.xport()` para ler do formato TRANSPORT do SAS e `read.ssd()` pode escrever dados permanentes do SAS (`.ssd` ou `.sas7bdat`) no formato TRANSPORT, se o SAS estiver disponível no seu sistema e depois usa internamente `read.xport()` para ler os dados no R.
  - O pacote Hmisc disponibiliza `sas.get()` que também requer o SAS no sistema. Para mais detalhes consulte a documentação de cada função e/ou o manual R Data Import/Export

Para mais detalhes consulte a documentação de cada função e/ou o manual R Data Import/Export.

8 Exportando objetos do R

8.1 Função `write.table()`

Para exportar objetos do R, usamos a função `write.table()`, que possui argumentos parecidos com aqueles da função `read.table()`.

A função `write.table()` é capaz de criar um arquivo de texto no formato `txt` ou `csv`, com as especificações definidas pelos argumentos.

Para ilustrar o uso desta função, considerer o conjunto de dados `iris`

```
data("iris")
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
```

```
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

Podemos exportar esse data frame com

```
write.table(iris, file = "iris.csv")
```

Por padrão, o arquivo resultante tem colunas separadas por espaço, o separador de decimal é ponto, e os nomes das linhas são também incluídos (o que geralmente é desnecessário). Para alterar essa configuração podemos fazer

```
write.table(iris, file = "iris.csv", row.names = FALSE, sep = ";", dec = ",")
```

Os argumentos são:

- `iris`: o nome do objeto a ser exportado (matriz ou data frame)
- `"iris.csv"`: nome do arquivo a ser gerado. (Considerando que o arquivo `iris.csv` será criado dentro do diretório dados).
- `row.names = FALSE`: para eliminar o nome das linhas do objeto (geralmente desnecessário), como retornado por `row.names()`
- `sep = ";"`: o separador de colunas (também pode ser `"\t"` para tabulação e `""` para espaços)
- `dec = ","`: o separador de decimais

Note que o objeto a ser exportado (nesse caso `iris`) deve ser em formato tabular, ou seja, uma matriz ou data frame. Outras classes de objetos podem ser exportadas, mas haverá uma coerção para data frame, o que pode fazer com que o resultado final não seja o esperado.

8.2 Função `write.csv2()`

Assim como `read.table()` possui as funções `read.csv()` e `read.csv2()`, a função `write.table()` possui as funções `write.table()` e `write.table2()` como wrappers. O comando acima também poderia ser executado como

```
write.csv2(iris, file = "iris.csv", row.names = FALSE)
```

Note que `row.names = FALSE` ainda é necessário para eliminar os nomes das linhas.

8.3 Função `export()`

A função `export()` do pacote `rio` pode ser utilizada para exportar objetos do R em arquivos dos mais diversos formatos.

Em arquivos `.csv`, os valores são separados por vírgula. No exemplo abaixo, é mostrado como o objeto quantitativo pode ser salvo em um arquivo `.csv` com nome quanti_exemplo. Para salvar as saídas em extensão `.txt` ou `.xlsx`, basta substituir a extensão do arquivo.

```
url <- "https://github.com/TiagoOlivoto/e-bookr/raw/master/data/data_R.xlsx"
quanti <- import(url, sheet = "QUANTI")
head(quanti)
# exportar para o diretóri padrão (csv)
export(quanti, file = "quanti_exemplo.csv")
# exportar para o diretóri padrão (txt)
export(quanti, file = "quanti_exemplo.txt")
# exportar para o diretóri padrão (xslx)
export(quanti, file = "quanti_exemplo.xlsx")
```

No formato `.xlsx` é possível informar em qual planilha o objeto será salvo. Neste caso, as planilhas existentes não serão modificadas.

```
export(quanti, file = "quanti_exemplo.xlsx", which = "quanti2")
```

Também é possível salvar diferentes objetos em diferentes planilhas do mesmo arquivo. Vamos considerar que cada nível do fator `TIPO` do objeto `quanti` deve ser salvo em uma planilha diferente.

```
linear <- subset(quanti, TIPO == "LINEAR")
quadratico <- subset(quanti, TIPO == "QUADRÁTICA")
cubico <- subset(quanti, TIPO == "CÚBICA")
export(list(linear = linear,
            quadratico = quadratico,
            cubico = cubico),
       file = "quanti_exemplo_plan.xlsx")
```

Neste caso, um arquivo chamado `quanti_exemplo_plan.xlsx` contendo as planilhas `linear`, `quadratico` e `cubico` foi criado no diretório padrão.

8.4 Função `write.xlsx`

O pacote para ler dados diretamente de arquivos do MS Excel também possui função para exportar diretamente para esse formato:

```
library(xlsx)
write.xlsx(iris,file="Iris.xlsx")
data.iris <- read.xlsx("Iris.xlsx", sheetIndex = 1)
```

O pacote `foreign` também possui funções para exportar para uma variedade de formatos. Veja a documentação em `help(package = "foreign")`.

Exercícios

Considere a tabela abaixo com o resultado de uma pesquisa que avaliou o número de fumantes e não fumantes por sexo:

	Sexo	
Condição	Masculino	Feminino
Fumante	49	54
	64	61
	37	79
	52	64
	68	29
Não fumante	27	40
	58	39
	52	44
	41	34
	30	44

1. Digite estes dados em uma planilha eletrônica em um formato apropriado para um data frame do R, e salve em um arquivo csv.
2. Importe esse arquivo para o R com read.table().
3. Crie uma nova coluna no objeto que contém estes dados, sendo a coluna com o número de pessoas multiplicada por 2 (aula anterior).
4. Exporte esse novo objeto usando a função write.table().
5. Crie esse mesmo conjunto de dados usando comandos do R (ex.: `c()`, `rep()`, `data.frame()`, etc).
6. Leia os dados da planilha `quadratico` presente no arquivo `quanti_exemplo_plan.xlsx`. Calcule a média da variável *RG*.

Consulta

O conteúdo deste tópico foi extraído de:

fernandomayer.github.io

tiagoolivoto.github.io