Capítulo 2 - Importando e Exportando dados no R

Marcelo Prudente

14 de agosto de 2018

Contents

Seção 1

Introdução

A importação de dados é uma da parte mais importantes da análise de dados no **R**. Embora seja uma tarefa fácil, muitas vezes o iniciante se confunde com a estratégia de importação. Este capítulo apresentará diversas formas de importação de dados com foco na eficiência e rapidez. Por isso, **não exite consultar este material quando estiver com dúvidas sobre a importação dos dados**. Além disso, esse capítulo mostrará como mesclar dados no **R**, tarefa fácil e muito útil no dia a dia.

DICA: você pode aprofundar os tópicos desta aula neste site:

- **Tutorial de importação de dados do Datacamp**

Seção 2

Importando Dados no R

2.1 Qual o diretório (pasta) de trabalho?

A primeira pergunta a ser feita quando importamos dados é: onde estão os meus dados? Por isso, duas funções importantes para a organização do trabalho são a **getwd()** e a **setwd()**.*Digite*: **getwd()** em seu console. O que você obtêm? Lembre-se que você pode pedir ajuda por meio do **?getwd()**.

Com essa função, você saberá qual a pasta de trabalho corrente vinculada a sua seção do **R**. No entanto, é provável que os seus dados não estejam ali. Como fazer, então?

2.2 Escolhendo o diretório de trabalho

Em regra, seus arquivos estão em alguma pasta no seu computador. Para melhorar a organização do trabalho no curso, a principal sugestão é criar uma pasta específica na qual ficarão os seus dados e scripts. Há duas possíbilidades para criar uma pasta:

1) ir na pasta diretamente, clicar com o botão direito e criar nova pasta.
2) criar a pasta por meio do **R**.

```
# criar a pasta para o curso
dir.create("C:/curso_r")
```

Conforme mostramos abaixo, é fácil definir a pasta de trabalho onde estão os seus arquivos.

```
# Estabelecer diretorio de trabalho
setwd("C:/curso_r")
```

Esse comando permite baixar arquivos com maior facilidade, pois cria-se uma conexão entre o **R** e sua pasta. Assim, podemos listar todos os arquivos contidos na pasta.

```r
# lista os arquivos da pasta
list.files()
```

Ainda, é possível criar novas subpastas para guardar seus dados.

```r
dir.create("meus_dados")
```

Seção 3

Os tipos de dados e preparação para leitura

Depois de identificar a pasta, o passo necessário é saber com qual tipo de dados estamos lidando. Há um infinidade de extensões, algumas mais, outras menos conhecidas. Em geral, as bases governamentais estão disponibilizadas em três principais extensões:

- formatos de texto (*.csv* ou *.txt*);
- em excel (*.xls* ou *.xlsx*);
- em dbf (*.dbf*, no caso dos dados do DATASUS)

Há ainda outros formatos relacionados aos diversos programas estatísticos do mercado:

- *.dta*, do STATA
- *.sav*, do SPSS
- *.sas7bdat*, do SAS
- *.fwf*, formato fixo
- entre outros

Além dessas, á os formatos de shapefile (*.shp*), xml (*.xml*), html, entre outros.

Se você leu o capítulo até aqui, lembre-se: **é muito importante saber qual a extensão do seu arquivo**. Sem isso, a leitura de dados pode se tornar um pesadelo. Não sei se mencionei: **aprender a importar dados corretamente é um passo muito importante para a correta análise**.

3.1 Separadores, decimais e encoding

De modo geral, a leitura de dados em todos os pacotes tem uma fórmula bastante comum. Não há necessidade de decorar, apenas entendero funcionamento:

1. Determinar o local do arquivo (ex: **“C:/curso_r/meu_arquivo.extensao”)
2. Determinar o separador do arquivo (**“,”**, **“;”**, **tab**, **separador fixo**)

3. Determinar os separadores dos números decimais (**“.”**, **“,”**)
4. Determinar se a primeira linha contém o nome das variáveis (**header == TRUE**)
5. Identificar o encoding do arquivo

O item 1 já foi abordado anteriormente. Lembre-se de identificar onde o seu arquivo está.

No caso do item 2, deve-se prestar muita atenção, sobretudo quando trabalhamos com arquivos de texto (**.txt**) e csv (**.csv**). Se você abrir um csv ou txt no bloco de notas, notará que as colunas estão separadas por algum dos separadores indicados (**“,”**, **“;”**, **tab**) ou, em alguns casos, diversos números emparelhados (**separador fixo**). A solução para esse problema é simples, conforme demonstraremos abaixo, mas deve **SEMPRE** ser observada. Não se preocupe, pois ao longo do tempo essa não será uma grande questão para você.

Outra fonte de problemas é a separação dos decimais (item 3). No **R**, as vírgulas não separam decimais como ocorre na notação brasileira. Os programadores sabem disso e incorporam em seus pacores de importação a opção para identificar se o decimal no arquivo original está separado por ponto ou por vírgula. Portanto, fique atento. Afinal, você aprendeu na aula 1 que número separado por vírgula é lido como texto.

```
x <- "3,2"
class(x)
```

```
## [1] "character"
```

```
y <- 3.2
class(y)
```

```
## [1] "numeric"
```

Por fim, o encoding é uma fonte de problemas quando temos variáveis de texto com acentos (item 5). Algumas das funções abaixo, especialmente do tidyverse, foram desenhadas para lidar com dados de texto em inglês (na verdade, UTF-8), que não contem acento. Abaixo, ensinaremos como lidar com esse tipo de problema.

3.2 Pacotes para a leitura de dados no R

Apesar dos diversos tipos de dados, a leitura deles no programa é bastante simples. Aqui, o foco será a leitura de dados em csv, txt, excel e dbc.

Há inúmeros pacotes desenhados para essa função:

- pacote de base do sistema, com *read.table()* e seus derivados;
- o pacote *tidyverse*, com o *read_table()* e seus derivados - note a presença do subscrito _ .
- o pacote *data.table*, com o *fread()*
- o pacote *foreign*, com *read.dta()*, *read.spss()*, entre outros;
- o pacote *read_dbc* com a função *read.dbc* para a leitura de dados do DATASUS.

```r
# instalar os pacotes
install.packages(readr) # ou install.packages(tidyverse)
install.packages(data.table)
install.packages(foreign)
install.packages(read.dbc)
```

3.3 Lendo .csv

Antes de executar os comandos, é importante verificar a pasta em que está o arquivo (lembre do passo 1). Depois, faça uma inspeção visual nos dados do arquivo *exemplo1.csv*. Como o csv é um arquivo de texto, você pode abrí-lo no bloco de notas ou no excel.

- identifique o separador do arquivo;
- identifique o separador dos decimais.

Atenção: o separador das colunas do arquivo *exemplo1.csv* é uma **,**

- Esse formato é muito comum nos dados americanos

3.3.1 Lendo .csv separado por vírgulas

Abaixo, apresentamos algumas abordagens para a leitura dos dados separados por vírgula.

```r
# Base do sistema
ex1_base <- read.csv("C:/curso_r/exemplo1.csv")
```

- Os comandos do read.csv e read_csv acima assumem que:
  - o **sep** = “,”
  - o **dec** = “.”
  - o **header** == **TRUE**

3.3.2 Lendo .csv separado por ponto e vírgula

Agora, vamos inspecionar o arquivo **exemplo2.csv**, no qual as colunas são separadas por **;**.

```r
# Base do sistema
ex2_base <- read.csv2("C:/curso_r/dados/exemplo2.csv")
```

Veja que há uma diferença em relação a forma de leitura do arquivo anterior: read_csv**2**. Esse **2** indica uma segunda forma de leitura em que:

- o **sep** = “;”
- o **dec** = “,”
- o **header** == **TRUE**

Na prática, a maior parte dos dados produzidos no Brasil têm separador **;** e dec = **.**. Então, na dúvida, utilize a segunda forma de ler csv: **read.csv2**.

3.4 Lendo txt

- o **R** fornece o **read.csv** como programa de base para ler arquivos de texto (*.txt*). Na verdade, os arquivos txt são lidos da mesma forma que os arquivos csv. Portanto, não há diferenças concretas, exceto a presença de vírgula ou ponto e vírgula no separador e vírgula ou ponto nos decimais.

```
# Base
ex3_base <- read.csv("C:/curso_r/dados/exemplo1a.txt")
```

3.5 Lendo com readr(): mais rápido e com melhor formatação

Apesar das funções constantes na base do sistema do **R**, há outras formas mais eficientes para a leitura dos dados. Uma delas é apresentada pelo readr(), do pacote tidyverse. A sintaxe é muito próxima à apresentada anteriormente e se diferencia apenas pela presença do _ no comando. Por isso, indicamos a leitura por meio do readr em detrimento da base do sistema.

```
# Tidyverse - readr
ex1_tdv  <- read_csv("exemplo1.csv")

# Ou ainda
ex1_tdv <- read_csv(file.choose())

# Tidyverse
ex2_tdv  <- read_csv2("exemplo2.csv")
```

A abordagem do pacote *readr* é preferível, pois permite importar dados no formato tbl_df (tiblle), cuja estrutura traz vantagens para leitura de dados. Além disso, é muito mais rápido que a base do sistema quando lidamos com bases de dados grandes (arquivos com mais de 100 mb, por exemplo) e bastante intuitivo. Como as atuais bases de dados são grandes, sugerimos o uso da abordagem do **readr**.

No quadro abaixo, identificamos outras formas de leitura contidas no pacote readr. Note as inúmeras possibilidades de leitura.

função	o que processa
read_csv()	arquivos csv separados por ,
read_tsv()	arquivos separados por tab
read_delim()	arquivos separados por delimitador geral (espaço ou tab)
read_fwf()	arquivos com largura fixa
read_table()	arquivos separados por um espaço em branco

3.5.0.1 readr: lendo apenas algumas linhas

A leitura dos dados pode envolver questões práticas como importar apenas algumas linhas dos dados - no caso de arquivos grandes, por exemplo, 1 milhão de linhas - com a extensão **n_max**. Essa é uma estratégia útil para dar uma rápida olhada nas informações do banco de dados.

```
ex1_tdv  <- read_csv("exemplo1.csv",
                     n_max =  2)
```

3.5.0.2 readr: pulando linhas

- Também é possível pular algumas linhas. Porém, com o readr os nomes das colunas não serão importados:

```
ex1_tdv  <- read_csv("exemplo1.csv", skip = 3,
                     n_max =  2)
```

3.5.0.3 readr: selecionando colunas

Ainda, há casos em que não se quer importar todas as colunas, apenas algumas.

```
ex1_tdv  <- read_csv("C:/curso_r/dados/exemplo1.csv", skip = 3,
                     col_names =  c("uf", "ano", "cod_ibge"))
```

3.6 fread(): velocidade

As funções do pacote readr são versáteis e produzem tbl_df, mas não são as mais rápidas diante de grandes bases de dados (> 1Gb). Sem dúvida, na presença de grandes bases de dados em txt ou csv a função fread(), do pacote **data.table**, tem a melhor performance. Além disso, o *fread()* é extramemente útil quando não sabe-se qual o separador do arquivo, pois a função foi construída para deduzi-lo.

Do mesmo modo que os outras funções de leitura, permite:

- ler apenas algumas linhas (**nrow = 10**, por exemplo) do arquivo para uma rápida inspeção
- delimitar os decimais com o comando **dec = “,”**
- também, é possível ler apenas algumas das colunas desejadas (**select(c(“col1”, “col5”))**), por exemplo))

Embora eficiente, às vezes apresenta erros para a leitura dos decimais. Por isso, sempre após ler os dados recomenda-se observar a estrutura do arquivo baixado para verificar se as colunas (vetores) correspondem aos tipos desejados.

```
library(data.table)
ex1_fread  <- fread("C:/curso_r/dados/exemplo1.csv",
                    nrows = 5)
ex1_fread
```

```
##     ano uf cod_ibge             municipio esp_vida tx_analf_15m   pop
## 1: 1991 11  1100015 ALTA FLORESTA D'OESTE    62.01        23.55 23417
## 2: 1991 11  1100023              ARIQUEMES    66.02        17.18 56061
## 3: 1991 11  1100031                 CABIXI    63.16        24.57  7601
## 4: 1991 11  1100049                 CACOAL    65.03        21.41 69173
## 5: 1991 11  1100056            CEREJEIRAS    62.73        20.26 19451
##      rdpc
## 1: 198.46
## 2: 319.47
## 3: 116.38
## 4: 320.24
## 5: 240.10
```

3.7 O problema do encoding

Ao baixar dados no R no Windows, muitas vezes a leitura de dados de texto pode ser problemática, especialmente com as soluções do **readr**. Isso porque essas funções foram concebidas para a leitura de dados americanos - sem acentos ou caracteres especiais.

Esse é um problema comum. Veja abaixo como o nome dos municípios brasileiros acabam distorcidos quando utilizamos o **read_csv**.

Note que a importação do **readr** retorna quais as classes dos vetores que conformam banco de dados.

```
# leitura com problema
problema <- read_csv("C:/curso_r/dados/exemplo1a.txt")
```

```
## Parsed with column specification:
## cols(
##   ano = col_integer(),
```

```
##   uf = col_integer(),
##   cod_ibge = col_integer(),
##   municipio = col_character(),
##   esp_vida = col_double(),
##   tx_analf_15m = col_double(),
##   pop = col_integer(),
##   rdpc = col_double()
## )
```

```
head(problema, 10)
```

```
## # A tibble: 10 x 8
##      ano    uf cod_ibge municipio        esp_vida tx_analf_15m   pop  rdpc
##    <int> <int>    <int> <chr>               <dbl>        <dbl> <int> <dbl>
##  1  1991    11  1100015 ALTA FLORESTA D~     62.0         23.6 23417 198.
##  2  1991    11  1100023 ARIQUEMES            66.0         17.2 56061 319.
##  3  1991    11  1100031 CABIXI               63.2         24.6  7601 116.
##  4  1991    11  1100049 CACOAL               65.0         21.4 69173 320.
##  5  1991    11  1100056 CEREJEIRAS           62.7         20.3 19451 240.
##  6  1991    11  1100064 COLORADO DO OES~     64.5         25.4 25441 225.
##  7  1991    11  1100072 CORUMBIARA           59.3         30.5 11968  81.4
##  8  1991    11  1100080 COSTA MARQUES        62.8         19.2  7418 250.
##  9  1991    11  1100098 "ESPIG\xc3O D'O~     64.2         22.1 22877 263.
## 10  1991    11  1100106 "GUAJAR\xc1-MIR~     64.7         16.7 32289 391.
```

A leitura dos acentos é problemática, mas é um problema facilmente solucionável. Mas perceba que o problema de leitura ocorrerá com texto. Se o banco tem apenas números, não há problema. Esse problema leitura de dados no windows pode ser resolvido com a especificação do encoding no corpo da função **read_csv**.

```
resolvido  <- read_csv("C:/curso_r/dados/exemplo1a.txt",
                       locale = locale(encoding = "Latin1"))
head(resolvido, 15)
```

3.8 Lendo Excel (xls e xlsx)

- Há duas formas práticas de ler dados do *Excel* no **R**.
- Vamos observar o arquivo **exemplo4.xlsx** e aplicar a leitura por comando.

```
library(readxl)
# Lendo os dados do excel
ex4  <- read_excel("C:/curso_r/dados/exemplo4.xlsx")

# Importar a segunda planilha
ex4_s2  <- read_excel("C:/curso_r/dados/exemplo4.xlsx", sheet = 2)
```

Há também a possibilidade de ler os dados com o cursor.Vá em *File* -> *Import Dataset* -> *From Excel*.

3.9 Lendo outros tipos de arquivo

- É possível ler outros arquivos, como os de SPSS, SAS ou STATA. Também, dados *.jason*, *.xml*, entre outros.
- Todos os comandos acima podem ser executados facilmente com o uso do mouse.
- Siga as instruções:
  - *File*
  - *Import Dataset*
  - *From ...*
- Apesar disso, recomendamos aprender os comandos. É mais rápido!! Caso as dúvidas persitam, acesse os links no final do arquivo.

3.10 Leituras adicionais

Mais uma vez, indico a leitura desse tutorial em caso de dúvida.

- **Tutorial de importação de dados do Datacamp**
- **Segunda parte do tutoria de importação do Datacamp**
- **Abordagem Tidyverse**
- **fread() do data.table: foco na eficiência**

Seção 4

Exportando dados

4.1 Exportando dados

- É possível utilizar as funções **write** para exportar dados do **R** para a sua pasta.
- Como os dados de texto podem ser lidos em qualquer programa (EXCEL,SPSS, SAS, STATA, entre outros), vamos aprender a exportar arquivos nessa forma.

```
# Você pode determinar a pasta na qual o arquivo ser? salvo
setwd("C:/curso_r/dados/")

# Exportando arquivos de texto
write.table(exemplo1_base, "meu_exemplo.csv")
```

4.2 Exportando dados

- Essa função permite determinar:

```
# Separador
write.table(exemplo1_base, "meu_exemplo2.csv",
          sep = ";")

# Decimal
write.table(exemplo1_base, "meu_exemplo2.csv",
          sep = ";", dec = ",")

# Omitir o número das linhas
write.table(exemplo1_base, "meu_exemplo2.csv",
          sep = ";", dec = ",", row.names = FALSE)
```

```r
# mais?
?write.table; help(write.table)
```

4.3 Exportando dados com readr e fread

Assim como na importação, os pacotes **readr** e **fread** trazem em seu bojo funções de exportação incorporando rapidez ao processo. No caso da exportação dos dados, não há maior mistério, sendo o pacote da base do sistema bastante satisfatório, mesmo com arquivos grandes. De qualquer modo, recomendo o uso da função do **data.table**, o *fwrite*.

```r
# com a função fwrite do data.table
fwrite(ex1_fread, "meu_exemplo_com_fread.csv",
          sep = ";", dec = ",", row.names = FALSE)
```

Caso queira exportar com a abordagem do readr, busque: **?write_csv**.

Seção 5

Exercícios: Importando dados com R

5.1 Exercício 1

Para esse exercício, utilizaremos o banco *dados_sociais.csv* na pasta dados.

5.1.1 Utilizando o pacote readr

1. Carregue o pacote readr
2. Baixe o arquivo *dados_sociais.csv* atribuindo a ele um objeto com mesmo nome.
   - verifique se os nomes dos municípios foram importados corretamente.
3. Baixe o arquivo *dados_sociais.csv* ajustando o encoding
4. Baixe o arquivo *dados_sociais.csv* ajustando o encoding, pulando as primeira 100 linhas e importando apenas 50 observações.
   - os nomes das colunas foram importados corretamente?
5. Baixe o arquivo *dados_sociais.csv* ajustando o encoding, pulando as primeiras 100 linhas e importando apenas 50 observações. Ajuste o código para baixar os nomes das colunas corretamente.
   - **?read_csv**

5.1.2 Utilizando fread()

1. Carregue o pacote data.table.
2. Baixe o arquivo dados_sociais.csv e atribua o nome **ds**
   - é preciso colocar o separador das colunas?
3. Observe a estrutra do objeto **ds**. As classes das colunas foram importadas corretamente?
4. Baixe o arquivo *dados_sociais.csv* e atribua o nome **ds**. Dessa vez, corrija os erros de importação.
5. Baixe o arquivo *dados_sociais.csv* e atribua o nome **ds**. Pule as 200 primeiras linhas e importe apenas 20.

5.1.3 Lendo em Excel

1. Carregue o pacote readxl.
2. Leia as quatro planilhas do arquivo **exemplo4.xlsx** de duas formas:
   - com o número da pasta de trabalho do excel.
   - com o nome da pasta de trabalho do excel.

- Nomeie cada uma das planilhas com os nomes das abas.

5.2 Exercício 2

- Com a base dados_sociais corretamente importada, verifique:
  - Qual a classe do banco de dados exemplo1_base? E do exemplo1_tdv?
  - Quais as classes das variáveis do banco?
  - Como visualizar esses dados?
  - Quais os nomes das variáveis do banco? Crie um vetor com esses nomes.
  - Como é possível extrair a média da expectativa de vida?
  - Qual o máximo e o mínimo dessa variável?
  - Crie um data.frame com cada um desses valores.

5.3 Exercício 3

1. Crie um tibble 10 x 4, sendo:
   - 1 coluna com um vetor numérico
   - 1 coluna com um vetor lógico
   - 1 coluna com um vetor character
   - 1 coluna com um vetor de texto
2. Crie um data.frame iris. Acesse **data()**. Procure o banco de dados *iris*. Atribua o banco a um objeto.
3. Exporte os dois bancos de dados em formato *csv* para a sua pasta de preferência.
4. Tente exportar cada arquivo com o um separador de coluna distinto.
5. Tente exportar cada arquivo com um separador decimal distinto.
6. Verifique se o número das linhas estão no arquivo exportado. Como resolver esse problema?