Importação de Dados Locais

Análise de Dados Financeiros e Econômicos com o R

Marcelo S. Perlin

EA-UFRGS

27/08/2026

Sumário

Importação de dados locais no R

Formatos disponíveis

> Dados são serializados ou salvos em texto em diferentes tipos de arquivos.

Em ordem de popularidade:

- Dados delimitados em texto (*csv*, *tsv*);
- Planilhas do Microsoft Excel (*xls*, *xlsx*);
- Arquivos de dados nativos do R (*RData* e *rds*)
- SQLite (*SQLITE*) e outros bancos de dados
- Texto não estruturado (*txt*)
- Formato parquet
- Formato `fst` (*fst*)
- Outros notáveis: Feather

Pontos importantes

- o local do arquivo no sistema deve ser indicado explicitamente no código. **Use o autocomplete com tecla *tab***

```
1 my_file ← 'C:/Data/MyData.csv'
```

- Note o uso de barras (/) para designar o diretório do arquivo. Referências relativas também funcionam, tal como em:

```
1 my_file ← 'Data/MyData.csv'
```

- os dados serão importados e exportados no R como **objetos do tipo** `dataframe`.

Pacote `afedR3`

> Pacote `afedR3` (Perlin 2023) é o pacote oficial do livro, contendo diversos arquivos de dados de exemplo.

Passos:

1. Instale o pacote `remotes` (caso ainda não o tenha feito) e depois instale `afedR3` com comando `remotes::install_github("msperlin/afedR3")`
2. Após instalação, verifique os dados disponíveis com o comando abaixo:

```
1 afedR3::data_list()
```

- Para ter o caminho completo de cada arquivo, basta usar função `afedR3::data_path` tendo o nome do arquivo como entrada:

```
1 # get location of file
2 my_f ← afedR3::data_path('CH11_grunfeld.csv')
3
```

```
4 # print it
5 print(my_f)
```

```
/home/msperlin/R/x86_64-pc-linux-gnu-
library/4.6/afedR3/extdata/data/CH11_grunfeld.csv
```

Arquivos *csv*

Introdução

- formato mais popular em análise de dados
- arquivos texto que podem ser abertos por qualquer editor/plataforma
- comportam dados volumosos e podem ser facilmente compactados para .zip ou .tar.gz
- funciona apenas para objetos do tipo `dataframes`
- a importação pode ser problemática, dependendo de onde o arquivo é gerado

Argumentos da função `readr::read_csv`

```
1 help("read_csv", package = 'readr')
```

```
read_csv(
  file,
  col_names = TRUE,
  col_types = NULL,
  col_select = NULL,
  id = NULL,
  locale = default_locale(),
  na = c("", "NA"),
  quoted_na = TRUE,
  quote = "\"",
  comment = "",
  trim_ws = TRUE,
  skip = 0,
  n_max = Inf,
  guess_max = min(1000, n_max),
  name_repair = "unique",
  num_threads = readr_threads(),
  progress = show_progress(),
  show_col_types = should_show_types(),
  skip_empty_rows = TRUE,
  lazy = should_read_lazy()
)
```

Importação

```r
# get location of file
my_f ← afedR3::data_path('CH04_ibovespa.csv')

df ← readr::read_csv(my_f)
print(df)
```

```
# A tibble: 3,215 × 2
   ref_date   price_close
   <date>           <dbl>
 1 2010-01-04       70045
 2 2010-01-05       70240
 3 2010-01-06       70729
 4 2010-01-07       70451
 5 2010-01-08       70263
 6 2010-01-11       70433
 7 2010-01-12       70076
```

Algumas dicas para lidar com csv

Um arquivo problemático (1/2)

Agora, vamos estudar um caso mais anormal de arquivo *.csv*. No pacote do livro temos um arquivo chamado `CH04_funky-csv-file.csv` onde:

- o cabeçalho possui texto com informações dos dados;
- o arquivo usará a vírgula como decimal;
- o texto do arquivo conterá caracteres latinos.

As primeiras 10 linhas dos arquivos contém o seguinte conteúdo:

```
1 my_f ← afedR3::data_path('CH04_funky-csv-file.csv')
2
3 readr::read_lines(my_f, n_max = 10)
```

```
[1] "Example of funky file:"
[2] "- columns separated by \";\""
[3] "- decimal points as \",\""
[4] ""
[5] "Data build in 2022-12-28"
```

```
 [6] "Origin: www.funkysite.com.br"
 [7] ""
 [8] "ID;Race;Age;Sex;Hour;IQ;Height;Died"
 [9] "001;White;80;Male;00:00:00;92;68;FALSE"
[10] "002;Hispanic;25;Female;00:00:00;99;68;TRUE"
```

Um arquivo problemático (2/2)

```r
my_locale ← readr::locale(decimal_mark = ',')

df_not_funky ← readr::read_delim(
  file = my_f,
  skip = 7, # how many lines do skip
  delim = ';', # column separator
  col_types = readr::cols(), # column types
  locale = my_locale # locale
)

dplyr::glimpse(df_not_funky)
```

```
Rows: 100
Columns: 8
$ ID      <chr> "001", "002", "003", "004", "005", "006",
"007", "008", "009", …
$ Race    <chr> "White", "Hispanic", "Asian", "White",
```

```
"White", "White", "White…
$ Age    <dbl> 80, 25, 25, 64, 76, 89, 33, 61, 23, 59,
80, 31, 83, 68, 67, 60,…
$ Sex    <chr> "Male", "Female", "Male", "Male",
"Female", "Female", "Female",…
```

Exportação de Dados

```r
library(readr)

# set number of observations
N ← 100

# create dataframe with random data
my_df ← data.frame(y = runif(N),
                   z = rep('a', N))

# write to file
f_out ← tempfile(fileext = '.csv')
readr::write_csv(x = my_df, file = f_out)
```

Arquivos *Excel* (*xls* e *xlsx*)

Introdução

- formato popular na indústria por razões de praticidade
- baixa portabilidade, arquivo pode corromper com o tempo
- muito difícil de trabalhar com dados volumosos
- funciona apenas para objetos `dataframes`
- EVITEM usar o excel no R!
- ótima alternativa: Google Sheets

Lendo arquivos *Excel*

```r
# set file
my_f ← afedR3::data_path("CH04_ibovespa-Excel.xlsx")

# read xlsx into dataframe
my_df ← readxl::read_excel(my_f, sheet = 'Sheet1')

# glimpse contents
dplyr::glimpse(my_df)
```

```
Rows: 3,215
Columns: 2
$ ref_date    <dttm> 2010-01-04, 2010-01-05, 2010-01-06,
2010-01-07, 2010-01-0…
$ price_close <dbl> 70045, 70240, 70729, 70451, 70263,
70433, 70076, 70385, 69…
```

Exportação de Dados

```r
# set number of rows
N ← 50

# create random dataframe
my_df ← data.frame(y = seq(1,N),
                   z = rep('a',N))

# write to xlsx
f_out ← tempfile(fileext = '.xlsx')
writexl::write_xlsx(
  my_df,
  f_out
)
```

Arquivos *.rds*

Introdução

- RDS = R Data Serialization
- formato nativo do R (só é usado no R!)
- rápido acesso e arquivo resultante é relativamente compacto
- funciona em qualquer tipo de objeto no R (`listas` são ótimas para salvar em .rds!)
- ótima opção para projetos exclusivos do R!
- péssima opção para compartilhar dados

Importação de Dados

```r
# set file path
my_file ← afedR3::data_path('CH04_example-rds.rds')

# load content into workspace
my_df ← readr::read_rds(file = my_file)
dplyr::glimpse(my_df)
```

```
Rows: 3,215
Columns: 2
$ ref_date    <dttm> 2010-01-04, 2010-01-05, 2010-01-06,
2010-01-07, 2010-01-0…
$ price_close <dbl> 70045, 70240, 70729, 70451, 70263,
70433, 70076, 70385, 69…
```

Exportação de Dados

```r
# set data and file
df ← data.frame(
  x = 1:100
)

my_file ← fs::file_temp(ext = '.rds')

# save as .rds
readr::write_rds(df, my_file)
```

Arquivos *fst* (pacote `fst`)

Introdução

- formato *fst* foi especialmente desenhado para possibilitar a gravação e leitura de dados tabulares de forma rápida e com mínimo uso do espaço no disco.
- usa todos os **núcleos** da máquina na importação e exportação (computação paralela)
- baixa portabilidade (funciona apenas no R)
- funciona apenas para objetos `dataframes`
- ótima opção para trabalhar com volumosas bases de dados em computadores potentes.

Importação de Dados

O uso do formato *fst* é bastante simples. Utilizamos a função `read_fst` para ler arquivos:

```
1 my_file ← afedR3::data_path('CH04_example-fst.fst')
2 my_df ← fst::read_fst(my_file)
3
4 dplyr::glimpse(my_df)
```

```
Rows: 100
Columns: 8
$ ID      <chr> "001", "002", "003", "004", "005", "006",
"007", "008", "009", …
$ Race    <fct> Black, White, Hispanic, Black, White,
White, White, White, Hisp…
$ Age     <int> 33, 35, 23, 87, 65, 51, 58, 67, 22, 52,
52, 71, 38, 23, 81, 31,…
$ Sex     <fct> Male, Female, Male, Female, Male, Male,
Female, Female, Male, F…
```

Exportação de Dados

```
1 library(fst)
2
3 # create dataframe
4 N ← 1000
5 my_file ← tempfile(fileext = '.fst')
6 my_df ← data.frame(x = runif(N))
7
8 # write to fst
9 write_fst(x = my_df, path = my_file)
```

Arquivos *SQLite*

Introdução

- alta portabilidade (SQLITE é uma tecnologia de banco de dados)
- permite leituras parciais dos dados (ex. ler apenas uma parte específica dos dados)
- permite *queries* personalizadas com a linguagem SQL, e de rápida execução no banco de dados (ex. contar número de casos)
- permite a criação e acesso a múltiplas tabelas
- ótima para casos onde o banco de dados fica mudando com novas inserções

Importação de Dados

Assumindo a existência de um arquivo com formato SQLite, podemos importar suas tabelas com o pacote `RSQLite`:

```
# set name of SQLITE file
f_sqlite ← afedR3::data_path('CH04_example-sqlite.SQL

# open connection
my_con ← RSQLite::dbConnect(drv = RSQLite::SQLite(),
                            f_sqlite)

# list tables
RSQLite::dbListTables(my_con)
```

```
[1] "MyTable1" "MyTable2"
```

```
# read table
my_df ← RSQLite::dbReadTable(conn = my_con,
                             name = 'MyTable1') # nam
```

```
4
5 # print with str
6 dplyr::glimpse(my_df)
```

```
Rows: 1,000
Columns: 2
$ x <dbl> 0.24645265, 0.47972926, 0.36782192, 0.09451063,
0.15353447, 0.014591…
$ G <chr> "B", "B", "A", "A", "A", "A", "A", "B", "B",
"A", "A", "A", "A", "A"…
```

```
1 # disconnect
2 RSQLite::dbDisconnect(my_con)
```

```
1 # open connection
2 my_con ← RSQLite::dbConnect(drv = RSQLite::SQLite(),
3                              f_sqlite)
4
5 # set sql statement
6 my_SQL_statement ← "select * from myTable2 where G='A
7
```

```r
# get query
my_df_A ← RSQLite::dbGetQuery(conn = my_con,
                              statement = my_SQL_stat

# disconnect from db
```

```
            x G
1  0.926352099 A
2  0.130461409 A
3  0.175863242 A
4  0.232315670 A
5  0.847271047 A
6  0.300454626 A
7  0.505549132 A
8  0.612706532 A
9  0.973640711 A
```

Exportação de Dados

```
1  library(RSQLite)
2
3  # set number of rows in df
4  N = 10^6
5
6  # create simulated dataframe
7  my_large_df_1 ← data.frame(x=runif(N),
8                             G= sample(c('A','B'),
9                                       size = N,
10                                      replace = TRUE))
11
12 my_large_df_2 ← data.frame(x=runif(N),
13                            G = sample(c('A','B'),
```

Dados Não-Estruturados e Outros Formatos

Introdução

- dados podem ser armazenados de forma não-estruturada, tal como um texto qualquer.
- tornar uma base de dados não-estruturada em estruturada pode ser um trabalho significativo
- o caso clássico de base de dados não-estruturada remete a importação de texto bruto

O livro *Pride and Prejudice*

```
# set file to read
my_f ← afedR3::data_path('CH04_price-and-prejudice.tx

# read file line by line
my_txt ← read_lines(my_f)

# print 50 characters of first fifteen lines
print(stringr::str_sub(string = my_txt[1:15],
                       start = 1,
                       end = 50))
```

```
 [1] "                                        [Illustration:"
 [2] ""
 [3] "                                         GEORGE ALLEN"
 [4] "                                           PUBLISHER"
 [5] ""
 [6] "                                  156 CHARING CROSS ROAD"
```

```
 [7] "                                                    LONDON"
 [8] ""
 [9] "                                                RUSKIN HOUSE"
[10] "                                                       ]"
```

Exportação de Dados Não-Estruturados

```r
# set file
my_f ← tempfile(fileext = '.txt')

# set some string
my_text ← paste0('Today is ', Sys.Date(), '\n',
                 'Tomorrow is ', Sys.Date()+1)

# save string to file
readr::write_lines(x = my_text, file = my_f, append =
```

```r
print(read_lines(my_f))
```

```
[1] "Today is 2026-08-27"    "Tomorrow is 2026-08-28"
```

Selecionando o Formato

Qual formato selecionar?

O usuário deve levar em conta três pontos na decisão de formato de arquivos de dados:

- velocidade de importação e exportação;
- tamanho do arquivo resultante;
- compatibilidade com outros programas e sistemas.

Na grande maioria das situações, o uso de arquivos *csv* satisfaz esses quesitos!

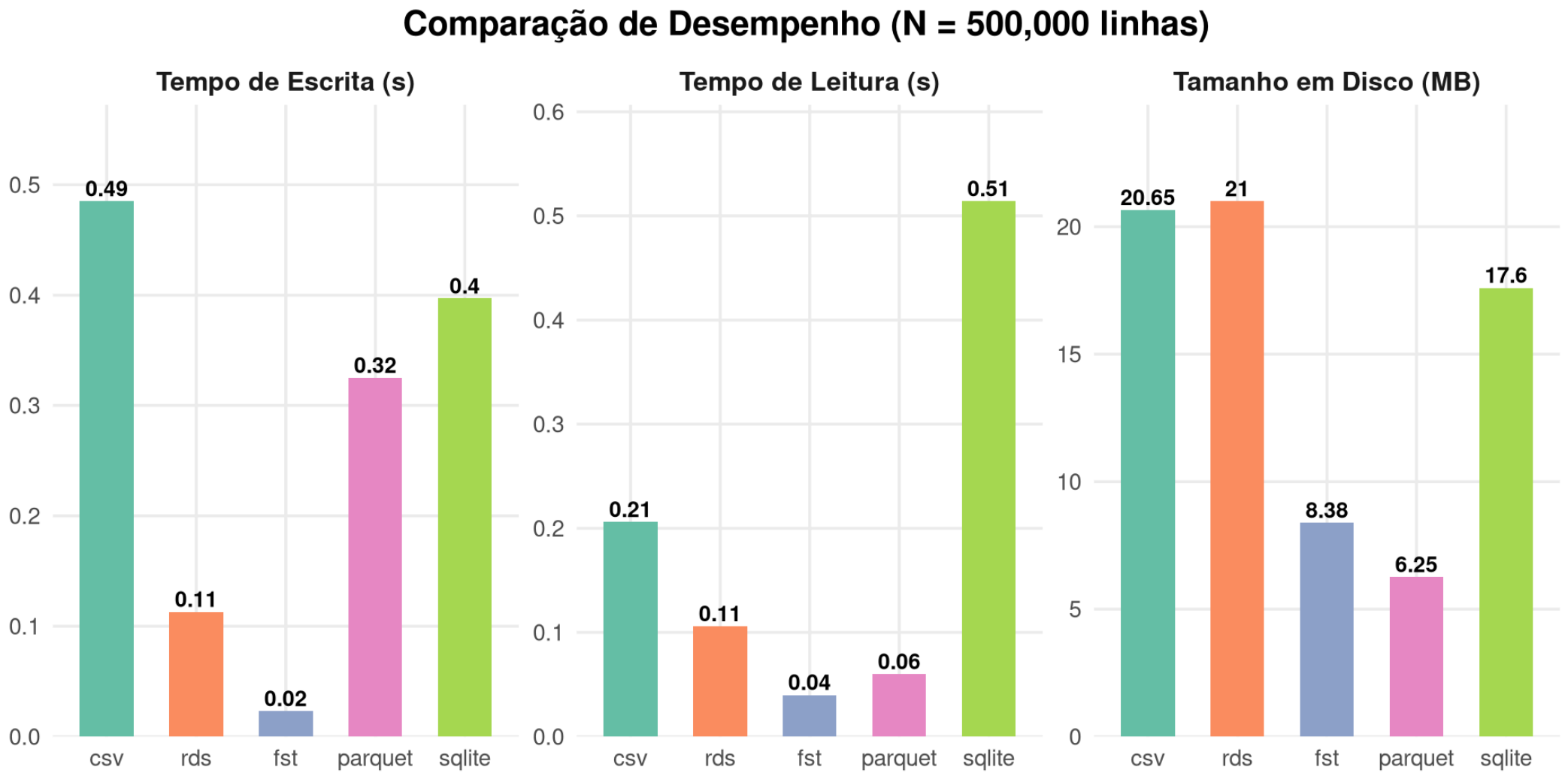
Caso abrir mão de portabilidade não faça diferença ao projeto, o formato *rds* é ótimo e prático. Se este não foi suficiente para dados maiores, as melhores alternativas são ***fst*** e ***parquet*** (via pacote `arrow`), os quais usam o máximo do *hardware* do computador e oferecem compressão e velocidade excepcionais. Como sugestão, caso puder, apenas **evite o formato do Excel**, o qual é o menos eficiente de todos.

Comparando o Desempenho dos Formatos (gravação, leitura e tamanho do arquivo)

Código

```
 1 library(fst)
 2 library(readr)
 3 library(arrow)
 4 library(RSQLite)
 5 library(wakefield)
 6 library(dplyr)
 7 library(purrr)
 8
 9 options(scipen = 999)
10
11 # create simulated dataset
12 set.seed(123)
13 n_row ← 500000
```

Resultado


Comparação de Desempenho (N = 500,000 linhas)
Tempo de Escrita (s)
0.49
0.11
0.02
0.32
0.4
Tempo de Leitura (s)
0.21
0.11
0.04
0.06
0.51
Tamanho em Disco (MB)
20.65
21
8.38
6.25
17.6
csv
rds
fst
parquet
sqlite

Big Data no R

ADFER - UFRGS

Introdução

> O R carrega todos os dados **diretamente na memória RAM** (*in-memory*).

- Quando o tamanho dos dados excede a RAM disponível, o computador trava (ou desliga sozinho).
- Na prática, raramente precisamos de todos os dados brutos de uma vez: apenas colunas, filtros ou agregações específicas, o que permite processamento mais eficiente.
- **Pacotes Recomendados:**
  - `arrow` (Apache Arrow): manipulação de arquivos colunares (`.parquet`) e datasets particionados em disco
  - `duckdb`: motor SQL analítico e de altíssimo desempenho
  - `sparklyr` / `dbplyr`: delegação de processamento pesado para bancos de dados SQL remotos ou clusters Apache Spark.

Exercícios

1. Crie um dataframe artificial utilizando o pacote `wakefield`, permitindo definir a quantidade de linhas, o número de colunas e diferentes tipos de variáveis (ex.: texto, numérico, fator, data, booleano):

```
library(wakefield)

n_row ← 10000
df ← r_data_frame(
  n = n_row,
  id,
  age,
  sex
)
```

Exporte o dataframe resultante para cada um dos formatos destacados a seguir:

- csv
- rds

Qual dos formatos ocupou maior espaço em disco? Podes verificar o tamanho dos arquivos no Windows Explorer ou com a função `file.size()` no R.

2. Experimente modificar o código anterior alterando:

   - O número de linhas (ex.: `n = 1000000`);

   Como o tamanho do arquivo e o tempo de gravação/leitura de cada formato respondem a diferentes tipos e volumes de dados?

3. No material do livro (pacote `afedR3`) existe um arquivo de dados chamado `CH08_some-stocks-SP500.csv`. Usando a função `afedR3::data_path`, utilize a função `readr::read_csv` para carregar o seu conteúdo. Utilize a função `glimpse` para verificar o conteúdo dos dados importados. Quantas colunas e qual o nome de cada coluna da tabela?

4. Na página da CVM é possível obter informações de todas as empresas atualmente listadas na bolsa em um arquivo disponível no link https://dados.cvm.gov.br/dados/CIA_ABERTA/CAD/DADOS/cad_cia_aberta. Utilizando o R, baixe o arquivo em seu computador, importe os dados diretamente usando `readr::read_csv` (a função importa diretamente de arquivos compactados, sem necessidade de descompactação explícita). Quantas empresas fazem parte da bolsa atualmente (verifique o número de linhas do dataframe com a função `nrow`)?

Referências

Perlin, Marcelo S. 2023. *afedR3: Data and Functions for Third Edition of Book "Analyzing Financial and Economical Data with r"*. https://github.com/msperlin/afedR3/.