

# R para Ciência de Dados I

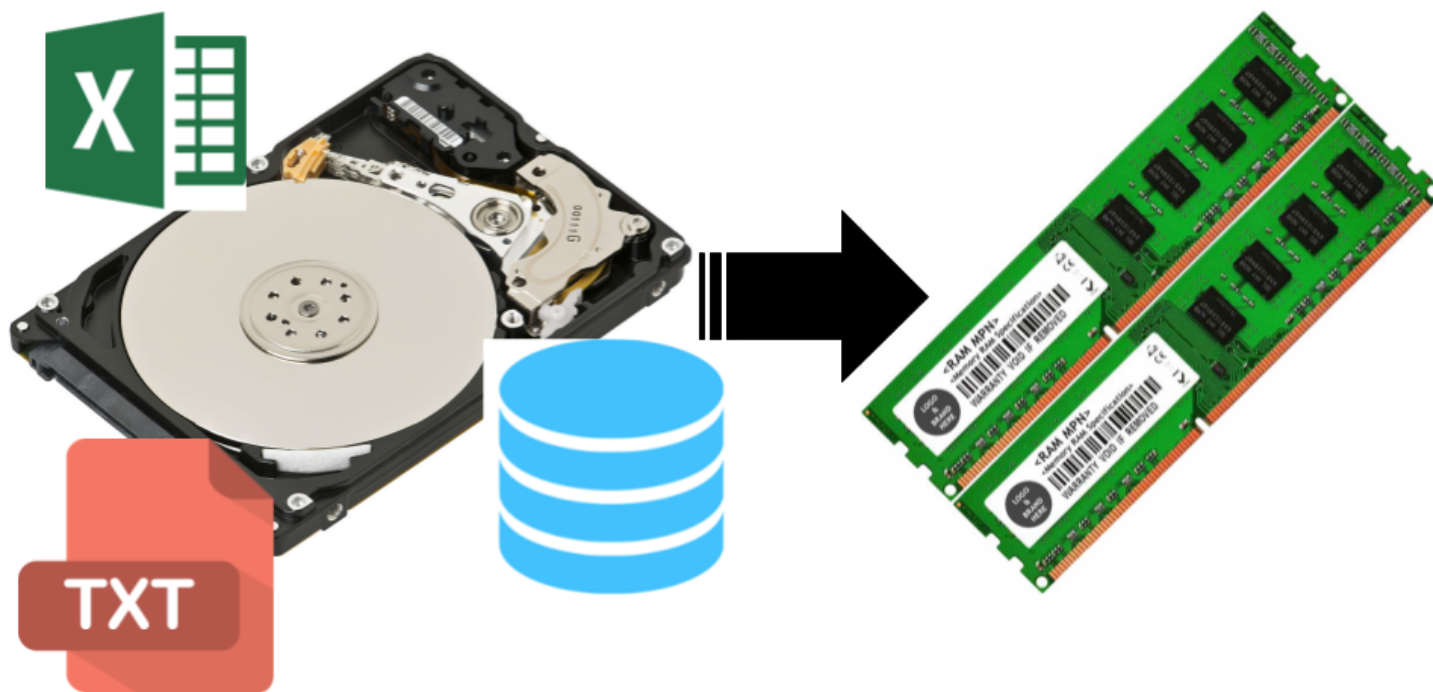
## Importação de dados



# Importação

# O que é?

Importar uma base de dados para o R significa levar a informação contida no disco rígido (HD) para a memória RAM.



# Caminhos

Um passo importante na tarefa de importação de dados para o R é saber onde está o arquivo que queremos importar.

Toda função de importação vai exigir um **caminho**, uma string que representa o endereço do arquivo no computador.

Há duas formas de passarmos o caminho de arquivo: usar o **caminho absoluto** ou usar o **caminho relativo**.

Antes de falarmos sobre a diferença dos dois, precisamos definir o que é o **diretório de trabalho**.

# Diretório de trabalho

O diretório de trabalho (*working directory*) é a pasta em que o R vai procurar arquivos na hora de ler informações ou gravar arquivos na hora de salvar objetos.

Se você está usando um projeto, o diretório de trabalho da sua sessão será, por padrão, a pasta raiz do seu projeto (é a pasta que contém o arquivo com extensão `.Rproj`).

Se você não estiver usando um projeto ou não souber qual é o seu diretório de trabalho, você pode descobri-lo usando a seguinte função `getwd()`.

Ela vai devolver uma string com o caminho do seu diretório de trabalho.

A função `setwd()` pode ser utilizada para mudar o diretório de trabalho. Como argumento, ela recebe o caminho para o novo diretório.

# Caminhos absolutos

Caminhos absolutos são aqueles que tem início na pasta raiz do seu computador/usuário. Por exemplo:

```
D:/GitHub/main-r4ds-1/slides
```

Esse é o caminho absoluto para a pasta onde esses slides foram produzidos.

Na grande maioria dos casos, caminhos absolutos são uma **má prática**, pois deixam o código irreprodutível. Se você trocar de computador ou passar o script para outra pessoa rodar, o código não vai funcionar, pois o caminho absoluto para o arquivo muito provavelmente será diferente.

# Caminhos relativos

Caminhos relativos são aqueles que tem início no diretório de trabalho da sua sessão.

O diretório de trabalho da sessão utilizada para produzir esses slides é a pasta `intro-programacao-em-r-mestre`. Veja o caminho absoluto no slide anterior. Então, o caminho relativo para a pasta onde esses slides foram produzidos seria apenas `slides/`.

**Trabalhar com projetos no RStudio ajuda bastante o uso de caminhos relativos**, pois nos incentiva a colocar todos os arquivos da análise dentro da pasta do projeto.

Assim, se você usar apenas caminhos relativos e compartilhar a pasta do projeto com alguém, todos os caminhos existentes nos códigos continuarão a funcionar em qualquer computador!

# Tibbles

Tibbles são uma evolução da classe *data frame*, que herdou apenas os comportamentos desejáveis dessa classe. As funções do **tidyverse** para importação e manipulação de bases devolvem sempre tibbles em vez de *data frames*.

Embora existam outras diferenças entre as classes, trataremos aqui apenas de uma: a forma como a tabela é mostrada no Console.

Quanto a isso, as diferenças são:

- tibbles mostram o número linhas, colunas e a classe das variáveis;
- tibbles só mostram as primeiras 10 linhas;
- tibbles só mostram o número de colunas que couber na tela;
- e tibbles não nomeiam linhas (`row.names`).



Veja a diferença com a base `mtcars`. Primeiro, como *data frame* (apenas as 20 primeiras linhas para caber no slide).

##	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
## Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
## Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
## Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
## Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
## Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
## Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
## Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
## Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
## Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
## Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
## Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
## Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
## Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
## Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
## Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
## Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
## Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4
## Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
## Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
## Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1

Agora como tibble.

```
tibble::as_tibble(mtcars)
```

```
## # A tibble: 32 x 11
##   mpg   cyl  disp    hp  drat    wt   qsec    vs  am  gear  carb
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1  21     6  160   110  3.9   2.62  16.5    0    1     4     4
## 2  21     6  160   110  3.9   2.88  17.0    0    1     4     4
## 3 22.8     4  108    93  3.85  2.32  18.6    1    1     4     1
## 4 21.4     6  258   110  3.08  3.22  19.4    1    0     3     1
## 5 18.7     8  360   175  3.15  3.44  17.0    0    0     3     2
## 6 18.1     6  225   105  2.76  3.46  20.2    1    0     3     1
## 7 14.3     8  360   245  3.21  3.57  15.8    0    0     3     4
## 8 24.4     4  147.    62  3.69  3.19  20      1    0     4     2
## 9 22.8     4  141.    95  3.92  3.15  22.9    1    0     4     2
## 10 19.2     6  168.   123  3.92  3.44  18.3    1    0     4     4
## # ... with 22 more rows
```

# Lendo arquivos de texto

Para ler arquivos de texto, como arquivos `.csv` ou `.txt`, utilizaremos funções do pacote `readr`.

Como exemplo, vamos utilizar uma base de filmes do IMDB. Essa base contém informações de 3713 filmes lançados entre 1916 e 2016.

Vamos então importar essa base para o R lendo o arquivo `imdb.csv` que está dentro da pasta "dados". Para isso, utilizamos a função `read_csv()`. Se o arquivo estiver bem formatado, a função só precisa do caminho até o arquivo para funcionar.

```
library(readr)

imdb_csv <- read_csv("dados/imdb.csv")
```

A mensagem devolvida pela função indica qual classe foi atribuída para cada coluna da base.

```
## Parsed with column specification:
## cols(
##   titulo = col_character(),
##   ano = col_double(),
##   diretor = col_character(),
##   duracao = col_double(),
##   cor = col_character(),
##   generos = col_character(),
##   pais = col_character(),
##   classificacao = col_character(),
##   orcamento = col_double(),
##   receita = col_double(),
##   nota_imdb = col_double(),
##   likes_facebook = col_double(),
##   ator_1 = col_character(),
##   ator_2 = col_character(),
##   ator_3 = col_character()
## )
```

O objeto resultante é uma tibble:

```
imdb_csv
```

```
## # A tibble: 3,713 x 15
##   titulo    ano diretor duracao cor   generos pais classificacao orcamento receita
##   <chr>    <dbl> <chr>      <dbl> <chr> <chr>   <chr>   <chr>          <dbl>    <dbl>
## 1 Avata~   2009 James ~    178 Color Action~ USA   A partir de ~ 2370000000  7.61e8
## 2 Pirat~   2007 Gore V~    169 Color Action~ USA   A partir de ~ 3000000000  3.09e8
## 3 The D~   2012 Christ~   164 Color Action~ USA   A partir de ~ 2500000000  4.48e8
## 4 John ~   2012 Andrew~   132 Color Action~ USA   A partir de ~ 2637000000  7.31e7
## 5 Spide~   2007 Sam Ra~    156 Color Action~ USA   A partir de ~ 2580000000  3.37e8
## 6 Tangl~   2010 Nathan~   100 Color Advent~ USA   Livre          2600000000  2.01e8
## 7 Aveng~   2015 Joss W~    141 Color Action~ USA   A partir de ~ 2500000000  4.59e8
## 8 Batma~   2016 Zack S~    183 Color Action~ USA   A partir de ~ 2500000000  3.30e8
## 9 Super~   2006 Bryan ~    169 Color Action~ USA   A partir de ~ 2090000000  2.00e8
## 10 Pirat~  2006 Gore V~    151 Color Action~ USA   A partir de ~ 2250000000  4.23e8
## # ... with 3,703 more rows, and 5 more variables: nota_imdb <dbl>,
## #   likes_facebook <dbl>, ator_1 <chr>, ator_2 <chr>, ator_3 <chr>
```

Em alguns países, como o Brasil, as vírgulas são utilizadas para separar as casas decimais dos números, inviabilizando os arquivos `.csv`.

Nesses casos, os arquivos `.csv` são na verdade separados por ponto-e-vírgula. Para ler bases separadas por ponto-e-vírgula no R, basta usar a função `read_csv2()`.

```
imdb_csv2 <- read_csv2("dados/imdb.csv")
```

Arquivos `.txt` podem ser lidos com a função `read_delim()`. Além do caminho até o arquivo, você também precisa indicar qual é o caractere utilizado para separar as colunas da base.

Um arquivo separado por tabulação, por exemplo, pode ser lido utilizando a o código abaixo. O código `\t` é uma forma textual de representar a tecla TAB.

```
imdb_txt <- read_delim("dados/imdb.txt", delim = "\t")
```

# Arquivos Excel

Para ler planilhas do Excel (arquivos `.xlsx` ou `.xls`), basta utilizarmos a função `read_excel()` do pacote `readxl`. Instale o pacote antes caso você ainda não o tenha instalado.

```
install.packages("readxl")  
  
library(readxl)  
  
imdb_xlsx <- read_excel("dados/imdb.xlsx")
```

# Argumentos úteis

Como planilhas do Excel são facilmente editáveis, é muito comum recebermos bases de dados desconfiguradas, isto é, em um formato que o R não consegue importar.

Para não precisarmos arrumar o arquivo na mão, correndo o risco de cometermos algum erro e alterarmos algum dado, a função `read_excel()` tem alguns argumentos muito úteis para lidarmos com essa situação.

Listamos abaixo os principais argumentos:

- `sheet=` para definir em qual aba estão os dados
- `col_names` indica se a primeira linha representa o nome das colunas
- `col_types=` para definir a classe das colunas
- `skip=` para pular linhas
- `na=` indica quais strings devem ser interpretadas como NA



# Gravando arquivos

Exportar objetos do R significa pegar uma informação que está na memória RAM e gravá-la em um arquivo no disco rígido (HD).

Em geral, para cada função `read_()` existe uma função `write_()`. As funções de escrita são bem simples: tudo o que você precisa passar para elas é o objeto que quer escrever e o caminho/nome do arquivo que será criado (ou sobrescrito). **O nome do arquivo deve conter a extensão.**

```
# CSV (vírgula) e CSV2 (ponto-e-vírgula)
readr::write_csv(mtcars, "mtcars.csv")
readr::write_csv2(mtcars, "mtcars.csv")

# TXT por tabulação, escrevendo dentro da pasta dados
readr::write_delim(mtcars, "dados/mtcars.txt", delim = "\t")

# Excel
writexl::write_xlsx(mtcars, "mtcars.xlsx")
```

Nos códigos acima, os pacotes estão explícitos apenas para ficar claro a qual pacote cada função pertence. Na prática, você pode usar `library(nome_do_pacote)` e retirar o `nome_do_pacote::` do começo de cada função.

# A extensão .rds

A extensão `.rds` representa uma estrutura binária de arquivos nativa do R. Ela pode ser utilizada para salvarmos no disco rígido **qualquer objeto do R**, não só bases de dados (*data frames*) como nos exemplos anteriores.

Quando usamos para gravar base de dados, por ser binária, essa estrutura pode ser compactada para gerar arquivos muito menores.

Para criar e ler arquivos `.rds`, utilizamos as funções `write_rds()` e `read_rds()` do pacote `readr`.

```
# Escrevendo sem compactação
write_rds(mtcars, path = "mtcars.rds")

# Escrevendo com compactação
write_rds(mtcars, path = "mtcars.rds", compress = "gz")

# Lendo
read_rds("mtcars.rds")
```

# Conexão com bancos de dados

Em seguida, mostramos um exemplo de como conectar o R com um banco de dados. Por pragmatismo, utilizamos um banco SQLite, que funciona a partir de um arquivo `.sqlite` no HD.

O primeiro passo é conectar o R com o banco de dados.

```
conexao <- RSQLite::dbConnect(  
  RSQLite::SQLite(),  
  dbname = "dados/imdb.sqlite"  
)
```

O primeiro argumento indica qual é o driver SQL a ser utilizado. O segundo indica o caminho até o arquivo `.sqlite` que contém a base.

Como estamos usando um banco SQLite, utilizaremos funções do pacote `RSQLite`. Cada sabor de SQL terá um pacote exclusivo dentro do R. Se você estiver utilizando MySQL, por exemplo, utilize o pacote `RMariaDB`. Neste caso, também precisará passar os argumentos `username`, `password` e `host`.

```
# Apenas exemplo de código, não rode
conexao <- RMariaDB::(
  RMariaDB::MariaDB(),
  dbname = "db_imdb",
  username = "seu_usuario",
  password = "sua_senha",
  host = "IP_ou_localhost"
)
```

Para mais informações sobre outros tipos de SQL e seus pacotes no R, acesse: [db.rstudio.com](https://db.rstudio.com).

# Acessando uma tabela

Você pode ver as tabelas dentro de um banco utilizando a função `dbListTables()`.

```
RSQLite::dbListTables(conexao)
```

Feita a conexão com o banco de dados, podemos acessar tabelas dentro desse banco utilizando a função `tbl()` do pacote `dplyr`.

```
imdb_sqlite <- dplyr::tbl(conexao, "imdb")
```

Essa operação **não traz** a tabela para a memória RAM. O objeto `imdb_sqlite` funciona como uma tabela temporária.

Também podemos passar instruções SQL ao criar esses objetos temporários

```
instrucao <- dplyr::sql("SELECT titulo, ano, diretor FROM imdb")  
imdb_select <- dplyr::tbl(conexao, instrucao)
```

# Trazendo para a memória RAM

Para trazer uma tabela para a memória RAM, utilizamos a função `collect()` do pacote `dplyr`.

```
imdb_db <- dplyr::collect(imdb_sqlite)
imdb_db_select <- dplyr::collect(imdb_select)
```

Agora, esses objetos são como qualquer outro objeto do R.

# Escrevendo tabelas no banco

Para escrever tabelas no banco de dados utilizamos a função `dbWriteTable()`.

```
RSQLite::dbWriteTable(conexao, "mtcars", mtcars)
```

```
RSQLite::dbListTables(conexao)
```