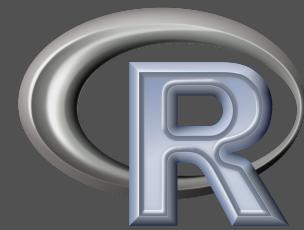




ME115 - Linguagem R

Parte 06

1º semestre de 2023



Arquivos e Diretórios no R

Arquivos e Diretórios no R

Antes de começarmos a importar e exportar dados usando o R, você precisa saber como localizar arquivos e diretórios.

Diretório de trabalho (*working directory*): é a pasta em que o R está trabalhando no momento. Onde ele vai procurar arquivos ou salvar objetos.

A função `getwd()` diz qual é o seu diretório de trabalho:

```
getwd()  
#> [1] "/volumes/GoogleDrive/Documents/Unicamp/2022S1/ME115/Slides"
```

Experimentem essa função no seu computador também.

Fonte: <https://r-coder.com/working-directory-r/>



Arquivos e Diretórios no R

Você pode modificar o seu diretório de trabalho, usando a função `setwd()`, cujo argumento é o caminho (*path*) para o novo diretório:

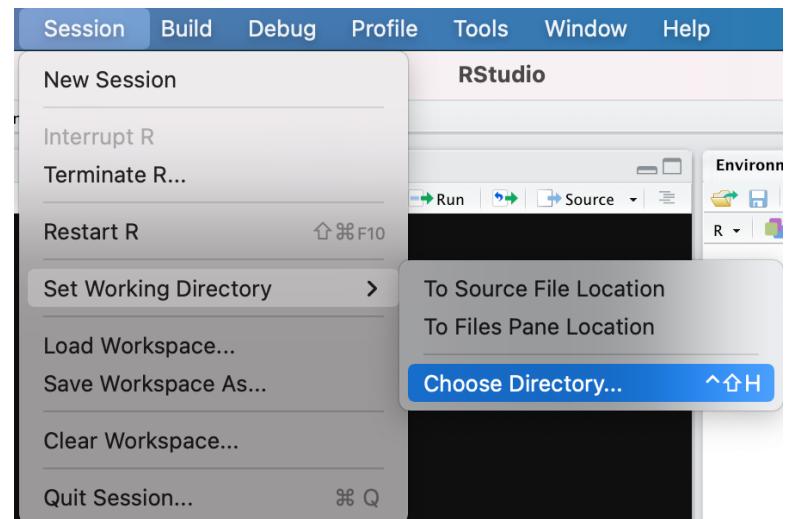
```
setwd("/Users/tatiana/Documents/Unicamp/2022S1/ME115/")
```

Mac ou Linux: o separador entre diretórios é "/"

Windows: o separador é "\".

R irá sempre imprimir o resultado usando "/".

No **RStudio** você pode ir diretamente em **Session > Set Working Directory** e escolher a opção desejada, como na figura ao lado.



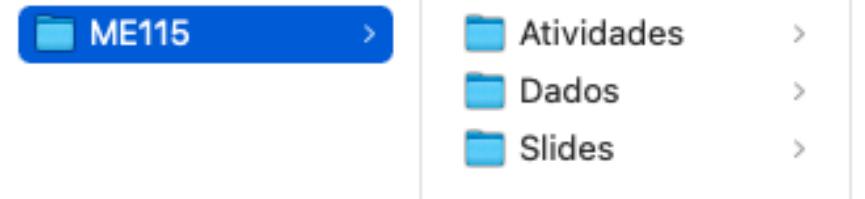
Caminhos Absolutos e Relativos

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

- **Caminho Absoluto:** tem início na pasta raiz do seu computador/usuário.
- **Caminho Relativo:** tem início no diretório de trabalho atual.

```
getwd()  
#> [1] "/Volumes/GoogleDrive/Documents/Unicamp/2022S1/ME115/Slides"
```

Exemplo: O caminho acima é o nosso diretório de trabalho atual e queremos mudá-lo para o diretório “Dados”, como indicado na figura ao lado.



```
setwd("/Volumes/GoogleDrive/Documents/Unicamp/2022S1/ME115/Dados/") # Caminho Absoluto  
setwd("../Dados/") # Caminho Relativo
```

Compartilhando seu Código

Prefira caminhos relativos: se você trocar de computador ou compartilhar o seu código com outra pessoa, pense que os caminhos (*paths*) devem funcionar também. Nesse caso, os caminhos relativos são preferíveis.

Trabalhar com Projetos no RStudio: você pode criar um R Project (com extensão .Rproj) e ao compartilhar o projeto, toda a estrutura de pastas é mantida.

Diferentes sistemas operacionais: para evitar problemas com o código ao mudar de sistema operacional, você pode usar a função `file.path()` para construir o caminho correto.

```
caminho <- file.path("Users", "tatiana", "Desktop")
#> [1] "Users/tatiana/Desktop"

setwd(caminho)
```

Arquivos em um Diretório

Para listar os arquivos contidos no diretório de trabalho, usamos as funções `list.files()` ou `dir()`.

Para listar os arquivos contidos em um diretório específico, por exemplo em "Dados", basta especificar o caminho:

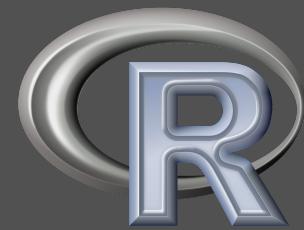


```
list.files(path = "../Dados/")

#> [1] "flights.csv.zip"  "Icon\r"          "iris.csv"
#> [4] "iris.txt"         "iris.xlsx"       "irisEspecie.xls"
```

Listar todos os arquivos que tenham ".xls" no nome:

```
list.files(path = "../Dados/", pattern = "*.xls")
#> [1] "iris.xlsx"        "irisEspecie.xls"
```



Importação e Exportação de Dados

Importação de Dados

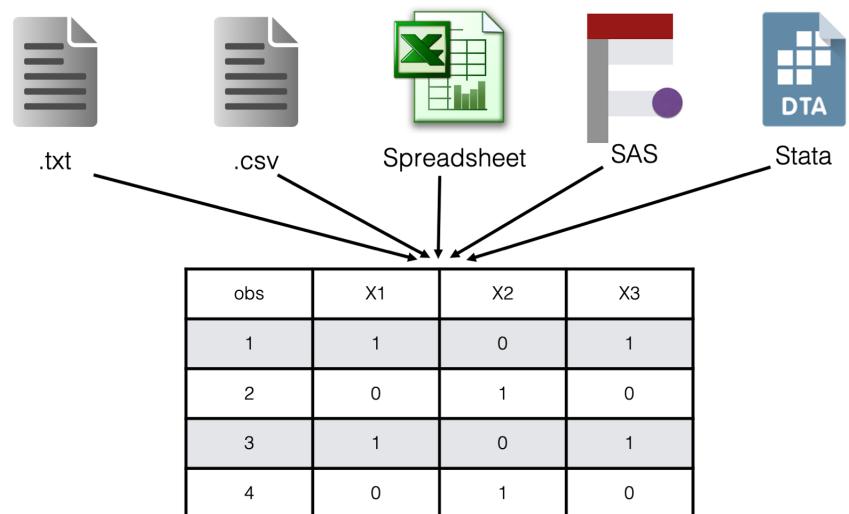
Até agora, usamos conjuntos de dados já disponibilizados como objetos do R.

```
data("iris")  
data("mtcars")
```

Entretanto, o mais comum é que esses dados estejam armazenados em arquivos, bases de dados e outras fontes e precisam ser importados para o R.

Fonte da imagem:

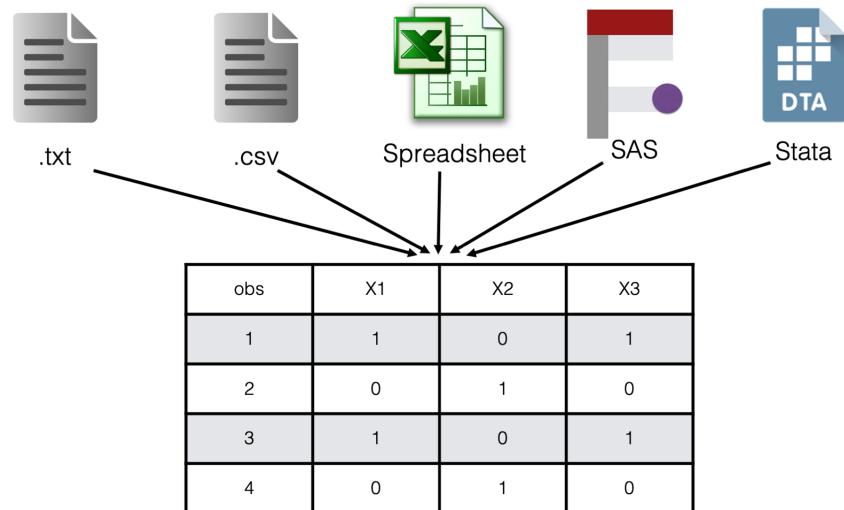
<https://www.r4epi.com/importing-binary-files.html>



Importação de Dados

A forma de importar os dados no R depende do formato que esses dados estão armazenados, podendo ser:

- Arquivos texto: TXT
- Arquivos do Excel: XLS ou CSV
- Arquivos de softwares estatísticos: SAS, SPSS, Stata, etc
- Arquivos da Web: HTML
- Databases: SQL



Existem inúmeros pacotes para nos ajudar com a importação.

Na aula de hoje, focaremos principalmente em arquivos tabulares salvos como texto (TXT) e arquivos do Excel (XLS e CSV).

Arquivos Tabulares

- Arquivos tabulares têm forma retangular (linhas e colunas). Exemplo clássico: planilha Excel.
- Colunas costumam representar variáveis e linhas, observações.
- Podem ser apresentados ao usuário/analista em diferentes versões:

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

table1

country	year	key	value
Afghanistan	1999	cases	745
Afghanistan	1999	population	19987071
Afghanistan	2000	cases	2666
Afghanistan	2000	population	20595360
Brazil	1999	cases	37737
Brazil	1999	population	172006362
Brazil	2000	cases	80488
Brazil	2000	population	174504898
China	1999	cases	212258
China	1999	population	1272915272
China	2000	cases	213766
China	2000	population	1280428583

table2

country	year	population
Afghanistan	1999	745 / 19987071
Afghanistan	2000	2666 / 20595360
Brazil	1999	37737 / 172006362
Brazil	2000	80488 / 174504898
China	1999	212258 / 1272915272
China	2000	213766 / 1280428583

table3

Fonte: <https://garrettgman.github.io/tidying/>

Arquivos Tabulares

O R armazena dados tabulares como *data frames*.

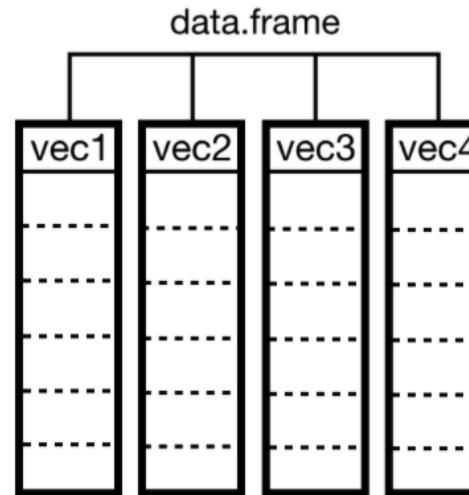


table1

country	year	cases	pop
Afghan	1999	745	19987071
Afghan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

Data frames: lista de vetores de mesmo comprimento dispostos lado a lado para parecer uma tabela.

Fonte: <https://garrettgman.github.io/tidying/>

Formato *Tidy*

- Anteriormente conhecido como formato longo (SAS);
- É o melhor formato para análises estatísticas;
- Pode não ser o formato mais compacto, mas é o mais versátil;
- Métodos comumente implementados para ciência de dados costumam utilizar como entrada dados no formato *tidy*;
- Colunas representam variáveis e as linhas correspondem a observações;
- O formato *tidy* funciona bem no R, pois é uma linguagem de programação vetorizada: estruturas de dados são construídas de vetores e operações são otimizadas para vetores.

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

table1

Formato *Tidy*

- Cada observação é uma linha;
- Cada variável é uma coluna;
- Cada valor é armazenado em uma célula.

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

table1

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

variables

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

observations

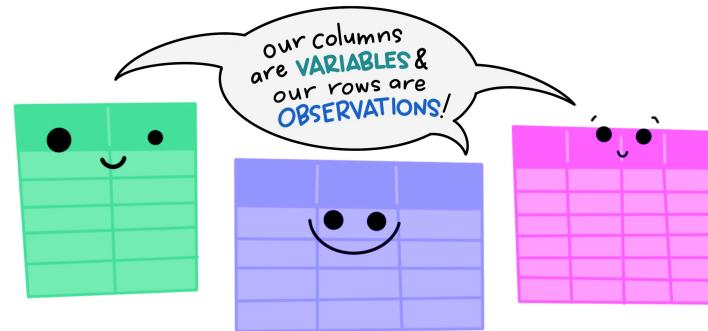
country	year	cases	population
Afghanistan	99	745	19987071
Afghanistan	00	2666	20595360
Brazil	99	37737	172006362
Brazil	00	80488	174504898
China	99	212258	1272915272
China	00	213766	1280428583

values

Fonte: <https://garrettgman.github.io/tidying/>

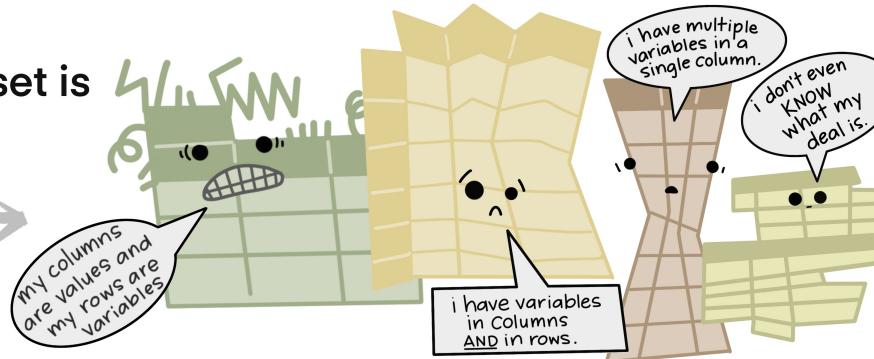
Formato *Tidy*

The standard structure of
tidy data means that
“tidy datasets are all alike...”



“...but every messy dataset is
messy in its own way.”

—HADLEY WICKHAM



Fonte: <https://cfss.uchicago.edu/notes/tidy-data/>

Formato não *Tidy*

Por que esses exemplos não estão no formato *tidy*?

country	year	key	value
Afghanistan	1999	cases	745
Afghanistan	1999	population	19987071
Afghanistan	2000	cases	2666
Afghanistan	2000	population	20595360
Brazil	1999	cases	37737
Brazil	1999	population	172006362
Brazil	2000	cases	80488
Brazil	2000	population	174504898
China	1999	cases	212258
China	1999	population	1272915272
China	2000	cases	213766
China	2000	population	1280428583

table2

country	year	population
Afghanistan	1999	745 / 19987071
Afghanistan	2000	2666 / 20595360
Brazil	1999	37737 / 172006362
Brazil	2000	80488 / 174504898
China	1999	212258 / 1272915272
China	2000	213766 / 1280428583

table3

- Tabela 2: os valores de **cases** e **population** estão intercalados na mesma coluna;
- Tabela 3: combina os valores **cases** e **population** na mesma célula.

Arquivos Delimitados (CSV, CSV2 ou TSV)

Arquivos no formato texto que, de acordo com o separador definido pelo criador do arquivo, temos os diferentes formatos: CSV, CSV2 ou TSV.

	CSV	CSV2	TSV
Formato	texto	texto	texto
Cabeçalho	opcional	opcional	Optional
Separador	, (vírgula)	; (ponto-e-vírgula)	tab (espaço em branco)
Separado Decimal	.	,	. ou ;
Exemplo	Produto,Dia,Valor Gasolina,Segunda,4.19 Gasolina,Terça,4.19 Gasolina,Quarta,4.09 Etanol,Segunda,3.39 Etanol,Terça,3.39 Etanol,Quarta,3.09	Produto;Dia;Valor Gasolina;Segunda;4,19 Gasolina;Terça;4,19 Gasolina;Quarta;4,09 Etanol;Segunda;3,39 Etanol;Terça;3,39 Etanol;Quarta;3,09	Produto Dia Valor Gasolina Segunda 4.19 Gasolina Terça 4.19 Gasolina Quarta 4.09 Etanol Segunda 3.39 Etanol Terça 3.39 Etanol Quarta 3.09

Arquivos de Largura Fixa

- Representação relativamente compacta de dados;
- A largura de cada campo é pré-especificada;
- Muito rápidos de serem importados, visto que a posição de cada campo é sempre fixa;
- Usuário precisa entender o posicionamento de cada campo;
- Essencialmente, é preciso conhecer as posições de início e fim de cada campo;

Arquivos XLS/XLSX

- Arquivos binários ou XML;
- Também conhecidos como “arquivos Excel”;
- Versões antigas do Excel, restringem arquivos a terem, no máximo, 65.535 linhas;
- Versões recentes do Excel, restringem arquivos a terem, no máximo, 1 milhão de linhas;
- Pela restrição no número de linhas, esse é um formato problemático para dados volumosos;



Sugestões para Criação de Arquivos Tabulares

Se você quer organizar seus dados em uma planilha, aqui estão algumas recomendações:

- Quando possível, prefira salvar sua planilha em formatos mais simples como arquivos texto ou outros binários;
- Você pode compactar o seu arquivo, pois não é preciso descompactá-lo antes da importação;
- A primeira linha é reservada para os nomes das colunas e a primeira coluna é usada para identificar a unidade amostral;
- Prefira nomes curtos;



Fonte da imagem: <https://r-coder.com/read-excel-r/>

Sugestões para Criação de Arquivos Tabulares

Continuação...

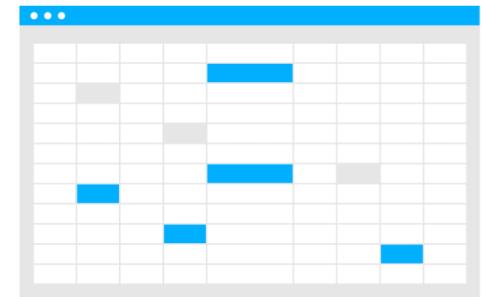
- Evite nomear colunas com expressões:
 - Iniciadas por números;
 - Que contenham valores com espaços em branco. Se quiser concatenar palavras, use (.) ou (_) em vez de espaço;
 - Que contenham caracteres especiais (? \$ % ^ & * () - # ? , < > / | \ [] { }) ou letras acentuadas;
- Apague qualquer comentário que você tenha feito no arquivo em Excel;
- Tenha certeza de que qualquer valor faltante está indicado com **NA**.



Salvando sua Planilha

Para salvar seu arquivo do Excel, prefira o formato CSV (*comma separated values*).

- Vá em **File > Save As** ou use o atalho **Ctrl + Shift + S**.
- Escolha um nome para o seu arquivo, digamos **MeusDados**.
- Antes de clicar para salvar, certifique-se de selecionar arquivo no formato **.csv** e verifique novamente o diretório onde o arquivo está sendo salvo.
- Quando salvo, você terá então o arquivo “**MeusDados.csv**” gravado no diretório escolhido.



Antes da Importação ...

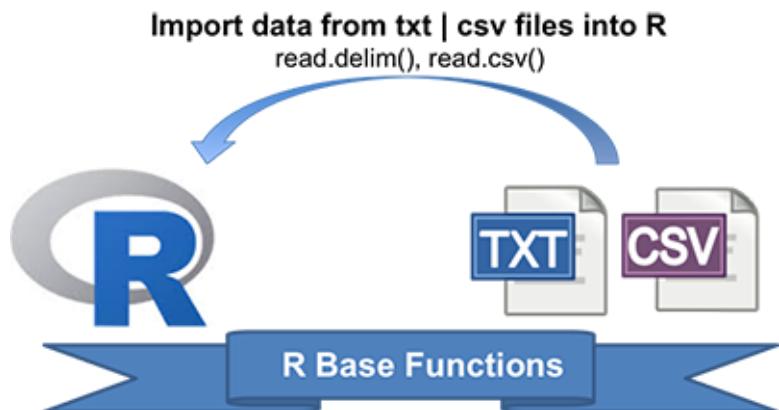
- Espie o conteúdo do arquivo. A função `readr::read_lines()` lê as `n_max` primeiras linhas do arquivo.
- O que separa uma coluna da outra?
- Qual é o separador decimal utilizado?
- Qual é o separador de milhar utilizado?
- O arquivo possui cabeçalho?
- Que *string* define o que é um valor faltante?
- Existem linhas de comentário dentro do arquivo?
- Existem linhas no início do arquivo que devem ser puladas no momento da importação?
- Quantas linhas devem ser importadas?
- Quais são os tipos de cada coluna a ser lida?

Importando Dados Tabulares

A base do R oferece várias funções para importação de dados tabulares, dependendo do tipo de arquivo.

Essas funções são do pacote `utils`:

- CSV: `read.csv()`
- CSV2: `read.csv2()`
- TSV: `read.delim()`
- Delimitados: `read.table()`
- Largura fixa: `read.fwf()`



A função principal é a `read.table()` e as demais são casos especiais desta.

```
read.table(file, header = FALSE, sep = "", quote = "\t",
          dec = ".", stringsAsFactors = TRUE)
```

Importando Arquivos TXT

Dados: Faça o download do arquivo `iris.zip` disponível no Moodle. Ele contém os dados `iris` salvos em vários formatos que serão usados nos exemplos a seguir.

Vamos importar o conjunto de dados `iris.txt`:

```
my.iris <- read.table("../Dados/iris.txt")
```

Veja o que acontece com os nomes das colunas:

```
colnames(my.iris)
```

```
## [1] "V1" "V2" "V3" "V4" "V5"
```

Isso acontece porque o argumento `header = FALSE` por *default*. Então, faça:

```
my.iris <- read.table("../Dados/iris.txt", header = TRUE)  
colnames(my.iris)
```

```
## [1] "Sepal.Length" "Sepal.Width"   "Petal.Length"  "Petal.Width"   "Species"
```

Importando Arquivos TXT

Vamos verificar a classe cada coluna dos dados importados:

```
sapply(my.iris, class)
```

```
## Sepal.Length  Sepal.Width  Petal.Length  Petal.Width      Species  
##   "numeric"    "numeric"    "numeric"    "numeric"    "character"
```

Veja que a coluna **Species** está como **character**. É isso mesmo que você quer?

Em alguns casos, caracteres são convertidos automaticamente para fatores e isso nem sempre é conveniente! Para controlar essa opção veja o argumento **stringAsFactors**:

```
my.iris <- read.table("../Dados/iris.txt", header = TRUE, stringsAsFactors = TRUE)  
sapply(my.iris, class)
```

```
## Sepal.Length  Sepal.Width  Petal.Length  Petal.Width      Species  
##   "numeric"    "numeric"    "numeric"    "numeric"    "factor"
```

Importando Arquivos CSV

Agora iremos importar o conjunto de dados `iris.csv` que também está salvo no seu computador:

```
my.iris <- read.csv("~/Documents/Unicamp/ME115/Dados/iris.csv")
```

Funções `read.csv()` e `read.csv2()` são comumente usada para arquivos .csv e são idênticas à função `read.table()`, exceto pelos *defaults* de alguns argumentos.

Alternativamente, em vez de indicar o caminho completo para o arquivo, podemos definir o caminho em um objeto separado com a função `file.path()`:

```
fullpath <- file.path("~/Documents/Unicamp/ME115/Dados/iris.csv")
fullpath
#> [1] "~/Documents/Unicamp/ME115/Dados/iris.csv"

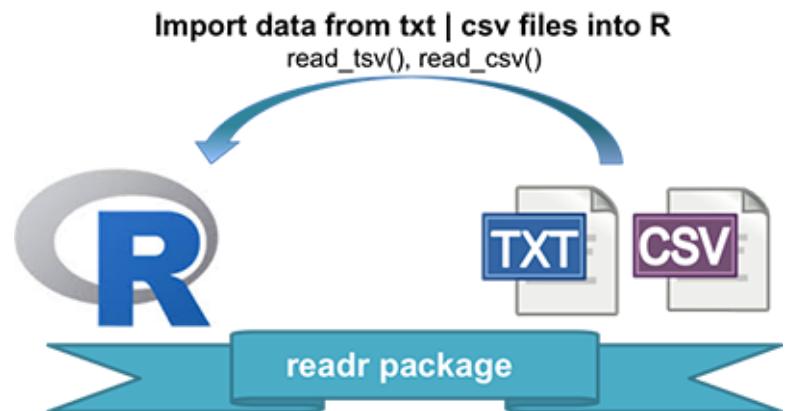
my.iris <- read.csv(fullpath)
```

Importando Dados Tabulares

Existem também pacotes mais atuais para importação de dados tabulares, como o pacote `readr`, parte do `tidyverse`.

Funções disponíveis no pacote `readr`:

- CSV: `read_csv()`
- CSV2: `read_csv2()`
- TSV: `read_tsv()`
- Delimitados: `read_delim()`
- Largura fixa: `read_fwf()`



Análogo à `read.table()` da base do R, aqui a função principal é `read_delim()`. As demais são casos especiais desta.

```
read_delim(file, delim, col_types = NULL, na = c("", "NA"), ...)
```

Pacote `readr`

Comparado com as funções base, as funções do `readr` são 10x mais rápidas e são mais fáceis de usar.

A saída foi construída para facilitar sua vida:

- Caracteres nunca são convertidos automaticamente para fatores (não precisa usar `stringsAsFactors = FALSE`)
- Nomes das colunas são deixadas como são. Para chamar variáveis com nomes não usuais ou até com espaço em branco, use crase. **Exemplo:** `df$`Income ($000)``
- O objeto criado é um *tibble*, classe `c("tbl_df", "tbl", "data.frame")`. Veja a diferença ao imprimir quando comparado a um `data.frame`.
- Não são usados nomes das linhas.



Importando Arquivos CSV - `readr`



Para ler um arquivo CSV, usamos a função `read.csv()` da base.
No pacote `readr`, a função equivalente é `read_csv()`:

```
library(readr)
my_iris <- read_csv("../Dados/iris.csv")
head(my_iris, 1)

## # A tibble: 1 × 5
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##       <dbl>      <dbl>       <dbl>       <dbl> <chr>
## 1       5.1       3.5        1.4        0.2  setosa
```

```
class(my_iris)
```

```
## [1] "spec_tbl_df" "tbl_df"        "tbl"          "data.frame"
```

Pacote `readr`



Para especificar o tipo de cada coluna, pode-se usar uma string compacta, na qual cada caracter representa uma coluna, ou através de uma lista:

```
my_iris <- read_csv("../Dados/iris.csv", col_types = "dddf")  
my_iris <- read_csv("../Dados/iris.csv", col_types = list(Species = col_factor()))  
head(my_iris, 4)
```

```
## # A tibble: 4 × 5  
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species  
##       <dbl>      <dbl>      <dbl>      <dbl> <fct>  
## 1         5.1        3.5        1.4        0.2  setosa  
## 2         4.9        3.0        1.4        0.2  setosa  
## 3         4.7        3.2        1.3        0.2  setosa  
## 4         4.6        3.1        1.5        0.2  setosa
```

Base do R vs `readr`

Aqui resumimos as principais funções para leitura de dados tabulares tanto na base do R (pacote `utils`) quanto no pacote `readr`:

<code>utils</code>	<code>readr</code>	Formato	Extensão
<code>read.table</code>	<code>read_table</code>	valores separados por espaços em branco	<code>txt</code>
<code>read.csv</code>	<code>read_csv</code>	valores separados por vírgula	<code>csv</code>
<code>read.csv2</code>	<code>read_csv2</code>	valores separados por ponto e vírgula	<code>csv</code>
<code>read.delim</code>	<code>read_tsv</code>	valores separados por tabulação	<code>tsv</code>
<code>read.table</code>	<code>read_delim</code>	formato geral de arquivo de texto, você deve definir delimitador	<code>txt</code>
<code>read.fwf</code>	<code>read_fwf</code>	formato de largura fixa	<code>txt</code>

Importação de Dados de uma Webpage

É comum que conjuntos de dados estejam armazenados em uma página da internet e não no seu computador.

Quando esses dados estão em arquivos, podemos baixá-los e importá-los, ou mesmo lê-los diretamente da web.

Exemplo: como o pacote `dslabs` está no GitHub, o arquivo que baixamos com o pacote tem uma URL:

```
url <- "https://raw.githubusercontent.com/rafalab/dslabs/master/inst/extdata/murders.csv"
```

E então, a função `read_csv()` pode ler esses arquivos diretamente:

```
murders <- read_csv(url)
```

Fazer o download de arquivos

Se deseja ter uma cópia local do arquivo, você pode usar `download.file()`:

```
download.file(url, "my_murders.csv")
```

Isso fará o download do arquivo e o salvará no diretório atual com o nome `my_murders.csv`.

Cuidado: arquivos existentes com mesmo nome serão substituídos **sem aviso!**

Uma função que pode ser útil ao baixar dados da internet é `tempfile()`. Ela cria um nome aleatório de arquivo, que pode ser excluído posteriormente.

```
tmp_filename <- tempfile()  
download.file(url, tmp_filename)  
murders <- read_csv(tmp_filename)  
file.remove(tmp_filename)
```

Atrasos em Vôos nos EUA

Esse conjunto de dados possui informação sobre atrasos de voos domésticos nos EUA realizados por grandes empresas aéreas.

Os dados estão disponíveis no link: <https://www.kaggle.com/usdot/flight-delays>

- `airlines.csv` tem 359 B
- `airports.csv` tem 23.3 KB
- `flights.csv.zip` tem 201.6 MB (zipped), 564.9 MB (unzipped).

Importando os dados sem descompactar o arquivo:

```
library(readr)
flights <- read_csv("../Dados/flights.csv.zip")
```

ATENÇÃO: o comando acima pode levar um tempo para rodar!

Atrasos em Vôos nos EUA

```
head(flights, 3)

#> # A tibble: 3 x 31
#>   YEAR MONTH   DAY DAY_OF_WEEK AIRLINE FLIGHT_NUMBER TAIL_NUMBER
#>   <dbl> <dbl> <dbl>        <dbl> <chr>          <dbl> <chr>
#> 1  2015     1     1           4  AS             98  N407AS
#> 2  2015     1     1           4  AA            2336  N3KUAA
#> 3  2015     1     1           4  US            840  N171US
#> # ... with 24 more variables: ORIGIN_AIRPORT <chr>, DESTINATION_AIRPORT <chr>,
```

- Dados tabulares podem ser bem grandes;
- **Questão:** Como trabalhar com um arquivo que possui 20GB de dados em um computador com 4GB de RAM?
- Calma!!! Soluções para isso serão discutidas lá em ME315 =)

Readr Cheat Sheet

Data import with the tidyverse :: CHEAT SHEET



Read Tabular Data with readr

```
read_*(file, col_names = TRUE, col_types = NULL, col_select = NULL, id = NULL, locale, n_max = Inf,  
skip = 0, na = c("", "NA"), guess_max = min(1000, n_max), show_col_types = TRUE) See ?read_delim
```

A B C 1 2 3 4 5 NA	→	A B C 1 2 3 4 5 NA	read_delim("file.txt", delim = " ") Read files with any delimiter. If no delimiter is specified, it will automatically guess. To make file.txt, run: write_file("A B C\n1 2 3\n4 5 NA", file = "file.txt")
A,B,C 1,2,3 4,5,NA	→	A B C 1 2 3 4 5 NA	read_csv("file.csv") Read a comma delimited file with period decimal marks. write_file("A,B,C\n1,2,3\n4,5,NA", file = "file.csv")
A;B;C 1,5;2,3 4,5;NA	→	A B C 1.5 2 3 4.5 5 NA	read_csv2("file2.csv") Read semicolon delimited files with comma decimal marks. write_file("A;B;C\n1,5;2,3\n4,5;NA", file = "file2.csv")
A B C 1 2 3 4 5 NA	→	A B C 1 2 3 4 5 NA	read_tsv("file.tsv") Read a tab delimited file. Also read_table() , read_fwf("file.tsv", fwf_widths(c(2, 2, NA))) Read a fixed width file. write_file("A\tB\tC\n1\t2\t3\n4\t5\tNA", file = "file.tsv")

USEFUL READ ARGUMENTS

A B C 1 2 3 4 5 NA	No header read_csv("file.csv", col_names = FALSE)	1 2 3 4 5 NA	Skip lines read_csv("file.csv", skip = 1)
x y z A B C 1 2 3 4 5 NA	Provide header read_csv("file.csv", col_names = c("x", "y", "z"))	1 2 3 NA 2 3 4 5 NA	Read a subset of lines read_csv("file.csv", n_max = 1)
Read multiple files into a single table	read_csv("f1.csv", "f2.csv", "f3.csv", id = "origin_file")	A B C 1,5;2,3,0	Read values as missing read_csv("file.csv", na = c("1"))

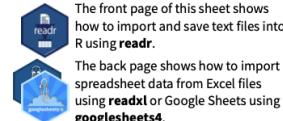
Save Data with readr

```
write_*(x, file, na = "NA", append, col_names, quote, escape, eol, num_threads, progress)
```

A B C 1 2 3 4 5 NA	→	A,B,C 1,2,3 4,5,NA	write_delim(x, file, delim = "") Write files with any delimiter. write_csv(x, file) Write a comma delimited file. write_csv2(x, file) Write a semicolon delimited file. write_tsv(x, file) Write a tab delimited file.
--------------------------------------	---	--------------------------	---



One of the first steps of a project is to import outside data into R. Data is often stored in tabular formats, like csv files or spreadsheets.



OTHER TYPES OF DATA
Try one of the following packages to import other types of files:

- **haven** - SPSS, Stata, and SAS files
- **DBI** - databases
- **jsonlite** - json
- **xml2** - XML
- **httr** - Web APIs
- **rvest** - HTML (Web Scraping)
- **readr::read_lines()** - text data

Column Specification with readr

Column specifications define what data type each column of a file will be imported as. By default readr will generate a column spec when a file is read and output a summary.

spec(x) Extract the full column specification for the given imported data frame.

```
spec(x)  
# cols(  
#   age = col_integer(),  
#   sex = col_character(),  
#   earn = col_double()  
# )
```

age is an integer
sex is a character
earn is a double (numeric)

COLUMN TYPES

Each column type has a function and corresponding string abbreviation.

- **col_logical()** - "l"
- **col_integer()** - "i"
- **col_double()** - "d"
- **col_number()** - "n"
- **col_character()** - "c"
- **col_factor()**(levels, ordered = FALSE) - "f"
- **col_datetime()**(format = "") - "T"
- **col_date()**(format = "") - "D"
- **col_time()**(format = "") - "L"
- **col_skip()** - "!"
- **col_guess()** - "?"

DEFINE COLUMN SPECIFICATION

Set a default type

```
read_csv(  
  file,  
  col_type = list(default = col_double()))  
)
```

Use column type or string abbreviation

```
read_csv(  
  file,  
  col_type = list(x = col_double(), y = "l", z = " "))  
)
```

Use a single string of abbreviations

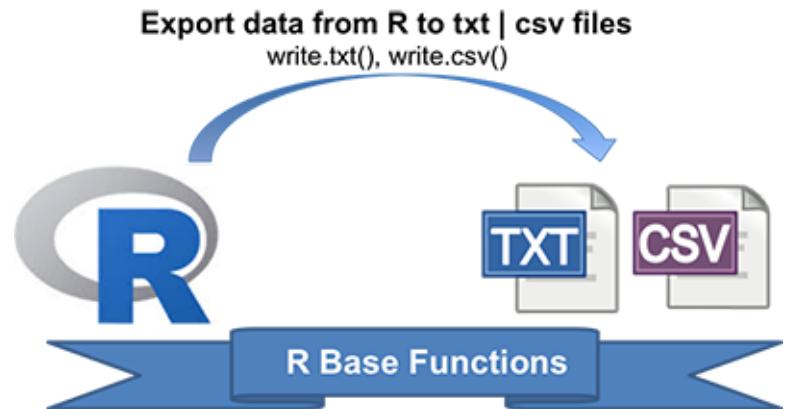
```
# col_types: skip, guess, integer, logical, character  
read_csv(  
  file,  
  col_type = "_?ilc")  
)
```

Fonte: [Readr Cheat Sheet](#)

Exportação de Dados

Além de importar dados para o R, você pode querer exportar dados que estão no seu ambiente de trabalho para arquivos .txt ou .csv, por exemplo.

Na base do R, o equivalente das funções de importação `read.table()` e `read.csv()` são as funções `write.table()` e `write.csv()`.



Exemplo: exportar o conjunto de dados `iris` para um arquivo texto:

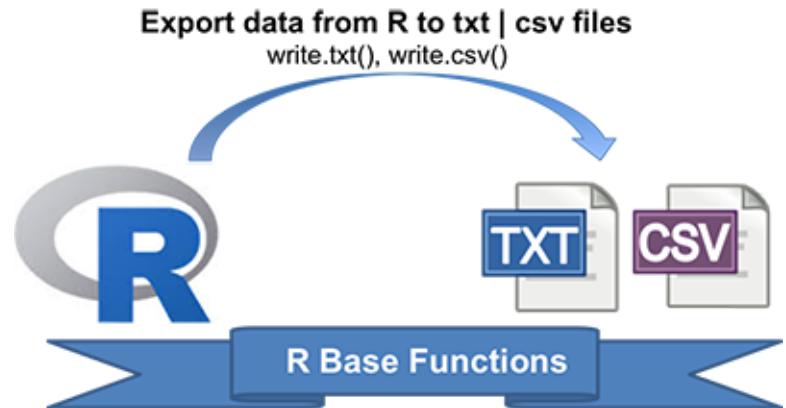
```
write.table(iris, "iris.txt") # salva no diretório atual  
  
list.files(pattern = "iris*")  
#> [1] "iris.txt"
```

Exportação de Dados

Por default, as funções `write.table()` e `write.csv()` criam uma coluna extra no arquivo contendo o número das linhas. Para evitar isso, modifique o argumento `row.names`.

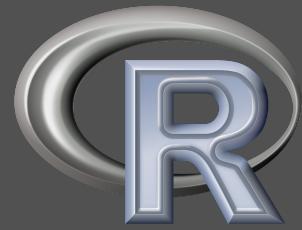
Vamos agora salvar os dados `iris` em um arquivo .csv, sem os números das linhas.

```
write.csv(iris, "iris.csv", row.names = FALSE) # salva no diretório atual  
write.csv(iris, "~/Documents/Unicamp/ME115/Dados/iris.csv") # salva no diretório Dados
```



Veja que você pode especificar que o arquivo seja salvo em qualquer diretório.

Cuidado: Se você escolher um nome de arquivo já existente, o R sobrescreve o arquivo original sem que apareça nenhum aviso!!!!



Importação e Exportação de Dados - Arquivos Excel

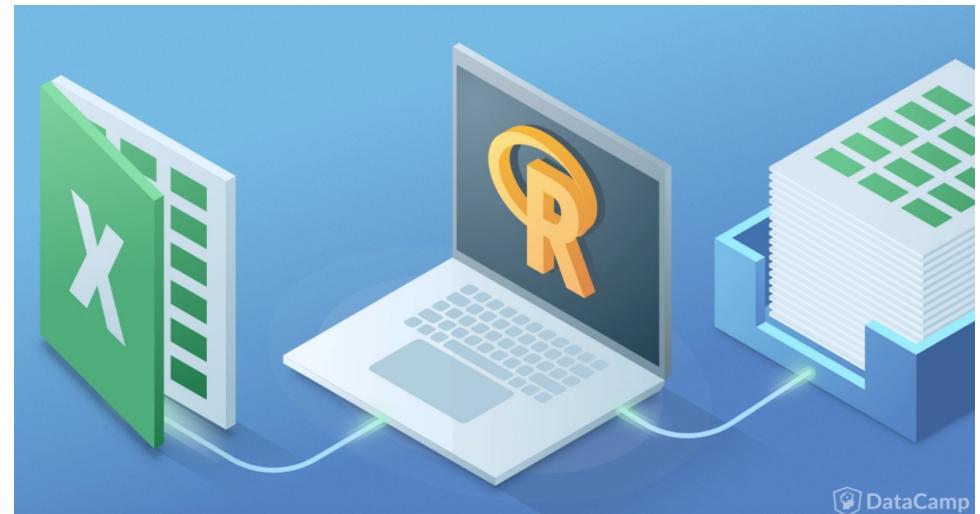
Importando Arquivos do Excel

Quando falamos em dados tabulares, pensamos logo em arquivos do Excel.

Então, vamos apresentar algumas ferramentas de importação e exportação de dados para esse tipo de arquivo (extensões .xls ou .xlsx).

Existem inúmeros pacotes específicos para leitura e/ou escrita de arquivos Excel:

- `writexl`
- `readxl`
- `XLConnect`
- `xlsx`
- `openxlsx`



Exportando Dados para Excel - `writexl`

O pacote `writexl` tem uma única função, a `write_xlsx()`, que exporta um *data frame* para um arquivo Excel com extensão .xlsx com uma ou mais planilhas.

```
library(writexl)
write_xlsx(iris, path = "../Dados/iris.xlsx")
write_xlsx(list(Planilha1 = iris), path = "../Dados/iris.xlsx")

list.files(pattern = "*.xlsx", path = "../Dados")

## [1] "iris.xlsx"
```

O desempenho dessa função é melhor que outras com a mesma finalidade: mais rápida e produzindo arquivos menores.

```
file.info("../Dados/iris.xlsx")$size

## [1] 8503
```

Pacote `readxl`

- Escrito por Hadley Wickham.
- Mais rápido e fácil para carregar arquivos Excel no R.
- Parte do pacote `tidyverse`.
- Pode ser instalado diretamente ou através do pacote `tidyverse`.
- Deve ser explicitamente carregado, ou seja, não basta carregar o `tidyverse`.



Você pode carregar o pacote `readxl` usando:

```
library(readxl)
```

Importando Dados do Excel - `readxl`



O pacote oferece as seguintes funções para a leitura nos formatos do Microsoft Excel de acordo com a extensão:

Função	Formato	Extensão
<code>read_excel</code>	detecta automaticamente o formato	xls, xlsx
<code>read_xls</code>	formato original	xls
<code>read_xlsx</code>	novo formato	xlsx

Os formatos do Microsoft Excel permitem que você tenha mais de uma planilha em um arquivo.

As funções listadas acima leem por padrão a primeira planilha, mas também podemos ler as outras, como veremos a seguir.



Pacote `readxl`

Os argumentos da função são simples e diretos:

```
library(readxl)  
my_iris <- read_excel("../Dados/iris.xlsx")
```

Esse comando, por *default*, irá ler a primeira planilha do seu arquivo.

Se o arquivo possui mais que uma planilha, basta extrair os nomes das planilhas usando a função `excel_sheets()` e adicionar a planilha desejada na função `read_excel()`.

```
excel_sheets("../Dados/iris.xlsx")
```

```
## [1] "Planilha1"
```

```
read_excel("../Dados/iris.xlsx", sheet="Planilha1")
```

Pacote `readxl`

Se o argumento `col_names` não é especificado, seu default é `TRUE` e a função `read_excel()` irá importar a primeira linha da planilha como sendo os nomes das colunas.

Alinhado com os padrões do `tidyverse`, os nomes das colunas são deixadas exatamente como elas estão no arquivo em Excel.

Se o argumento `col_types` não for especificado, a função irá obter uma amostra de 10 linhas e assinalar a essa coluna o tipo de variável com maior ocorrência.

Caso contrário, você pode assinalar o tipo de coluna manualmente.

```
read_excel("../Dados/iris.xlsx",
          col_types = c(rep("numeric", 4), "text"))
```



Pacote `readxl`



É possível limitar o número de linhas a serem lidas com o argumento `n_max` ou especificando o argumento `range`.

Nesse exemplo, estamos lendo partes do arquivo `iris.xlsx` referentes a cada espécie e salvando em novos *data frames*:

```
iris_setosa <- read_excel("../Dados/iris.xlsx", n_max=50)
iris_versicolor <- read_excel("../Dados/iris.xlsx", range="A52:E101",
                               col_names=colnames(iris))
iris_virginica <- read_excel("../Dados/iris.xlsx", range="A102:E151",
                               col_names=colnames(iris))
head(iris_virginica, 2)
```

```
## # A tibble: 2 × 5
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##       <dbl>      <dbl>      <dbl>      <dbl> <chr>
## 1       6.3       3.3         6        2.5 virginica
## 2       5.8       2.7        5.1        1.9 virginica
```

Pacote XLConnect

XLConnect: “comprehensive and cross-platform R package for manipulating Microsoft Excel files from within R”.

É possível usar as funções para criar arquivos em Excel, inclusive com várias planilhas, e adicionar dados a elas.

```
library(XLConnect)
df <- readWorksheetFromFile("nome do arquivo e extensão",
                            sheet=1,
                            startRow = 4,
                            endCol = 2)
```



Outros argumentos: `startRow`, `startCol`, `endRow`, `endCol`, `region`.

Importando Dados do Excel - **XLConnect**



Alternativamente, é possível carregar o arquivo inteiro (*workbook*) e então ler a planilha específica (*worksheet*).

```
# Load in Workbook  
wb <- loadWorkbook("../Dados/iris.xlsx")  
# Load in Worksheet  
df <- readWorksheet(wb, sheet="Planilha1")
```

Além de ler arquivos, é possível também exportar *data frames* do R para arquivos de Excel usando a função `writeWorksheetToFile()`.

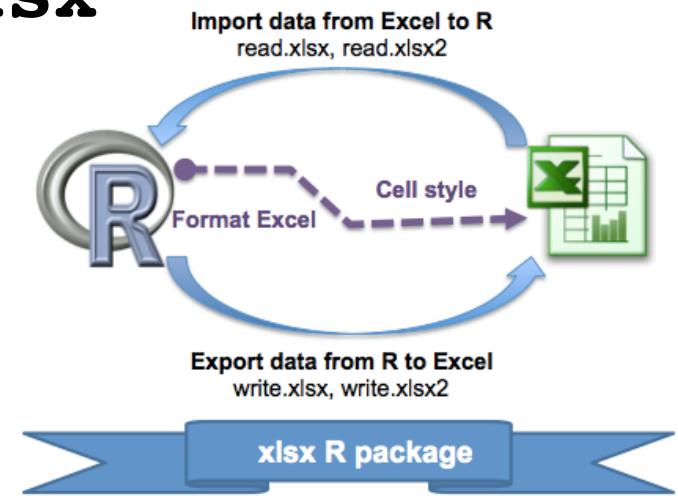
```
writeWorksheetToFile("irisSpecies.xls",  
                    data = list(i1=iris[1:50, ], i2=iris[51:100, ], i3=iris[101:150, ]),  
                    sheet = c("Especie1", "Especie2", "Especie3"),  
                    startRow = c(1, 10, 20), startCol = c(1, 6, 11))
```

Importando Dados do Excel - `xlsx`

No pacote `xlsx`, a função para ler os arquivos é bem semelhante à função `read.table()` e suas variantes:

Vamos ler os dados `iris.xlsx`, especificando a “Planilha1”:

```
library(xlsx)
df <- read.xlsx("../Dados/iris.xlsx", sheetIndex = "Planilha1")
```



Se você tem um conjunto de dados maior, você terá uma melhor performance usando a função `read.xlsx2()`:

```
df <- read.xlsx2("<nome e extensão do seu arquivo>",
                 sheetIndex = 1, startRow=2, colIndex = 2)
```

Exportando Dados para Excel - `xlsx`

Assim como o pacote `XLConnect`, o pacote `xlsx` permite exportar *data frames* do R para arquivos de Excel, usando as funções `write.xlsx()` e `write.xlsx2()`, semelhante à função `write.table()`.

```
write.xlsx(df, "df.xlsx", sheetName="New Data Frame")
```

Se você quiser adicionar um *data frame* (planilha) a um arquivo que já existe como uma nova planilha:

```
write.xlsx(df,
           "<nome e extensão do seu arquivo existente>",
           sheetName="New Data Frame",
           append=TRUE)
```

Fique atento para o local onde o arquivo será gravado!

Dados no Formato *R*

Dados formato *.RData*

Vantagens

É um formato nativo do R, o que significa que pode ser lido e escrito facilmente pelo R. Isso pode acelerar o tempo de leitura e gravação dos dados, em comparação com outros formatos que requerem pacotes adicionais para serem lidos pelo R.

Os objetos no formato .RData são compactos, o que significa que podem ser armazenados e transportados facilmente. Isso pode ser útil quando se trabalha com grandes conjuntos de dados e se precisa movê-los entre diferentes computadores ou servidores.

Desvantagem

O formato .RData é específico do R e não pode ser facilmente lido por outras linguagens de programação. Isso pode ser um problema se você precisa compartilhar seus dados com outras pessoas que não usam o R.

Exportando e Importando Dados

Para exportar um conjunto de dados no formato com extensão *.RData* utilizamos a função *save()*.

```
save(df, file = "nome.RData")
```

Para importarmos ou “carregarmos” um conjunto de dados no format *.RData* utilizamos a função *load()*

```
load(file = "nome.RData")
```



Importação de Dados de outros softwares

Importação de Dados

É possível ler um arquivo de dados binário escrito por outro software estatístico.

Geralmente, é melhor evitar isso, mas pode ser inevitável se o sistema de origem não está disponível.

O pacote `foreign` (<https://CRAN.R-project.org/package=foreign>) fornece recursos de importação de dados armazenados por algumas versões dos softwares estatísticos *EpiInfo*, *Minitab*, *S*, *SAS*, *SPSS*, *STATA*, *Systat*, *Weka*.

Em alguns casos, as funções desse pacote podem exigir substancialmente menos memória do que `read.table`.

```
library(foreign)
```

Pacote `haven`

Para ler arquivos gerados por outros softwares, como *SPSS*, *SAS* e *STATA*, você pode usar também as funções do pacote `haven`.

Este pacote faz parte do `tidyverse` e é um wrapper da biblioteca `ReadStat`, escrita em C.

- `read_sas`: leitura de arquivos *SAS*;
- `read_spss`: leitura de arquivos *SPSS*;
- `read_dta` leitura de arquivos *STATA*.



Além disso é possível salvar ou escrever bases em *SAS* e *STATA* com as funções `write_sas` e `write_dta`.

Referências

Algumas referências utilizadas para a construção desse material:

- [Introdução à Ciência de Dados - Capítulo 5](#)
- [R for Data Science - Chapter 11](#)
- [R Programming for Data Science - Chapter 5](#)
- [Ciência de Dados em R - Capítulo 5](#)
- [Readr Cheat Sheet](#)



Slides produzidos pela profa. Tatiana Benaglia e atualizados pela profa. Larissa Matos e pelo prof. Rafael Maia.