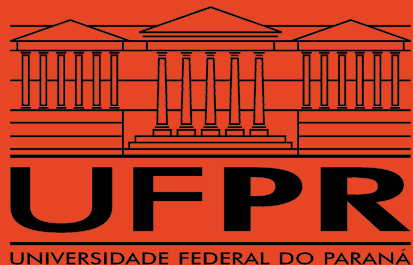


# Conhecendo o Tidyverse

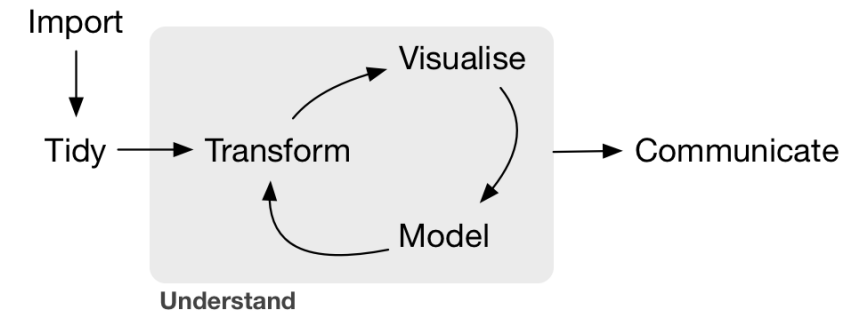
*tibble, readr, e readxl*

Fernando de Pol Mayer (LEG/DEST/UFPR)  
2022-03-08



# Manipulação e visualização de dados

- ▶ Manipular e visualizar dados (MVD) são atividades **obrigatórias** em Data Science (DS).
- ▶ A MVD **determina o sucesso** de uma série de etapas.
  - ▶ Entendimento dos dados.
  - ▶ Limpeza e conciliação de dados.
  - ▶ Engenharia de características.
  - ▶ Especificação de modelos.
  - ▶ Comunicação de resultados, etc.
- ▶ Fazer MVD de forma **eficiente** requer:
  - ▶ Conhecer o processo e suas etapas.
  - ▶ Dominar a tecnologia para execução.
- ▶ **Linguagens de programação** oferecem uma série de vantagens: reproduzível, extensível, escalonável, integrável, portátil, etc.

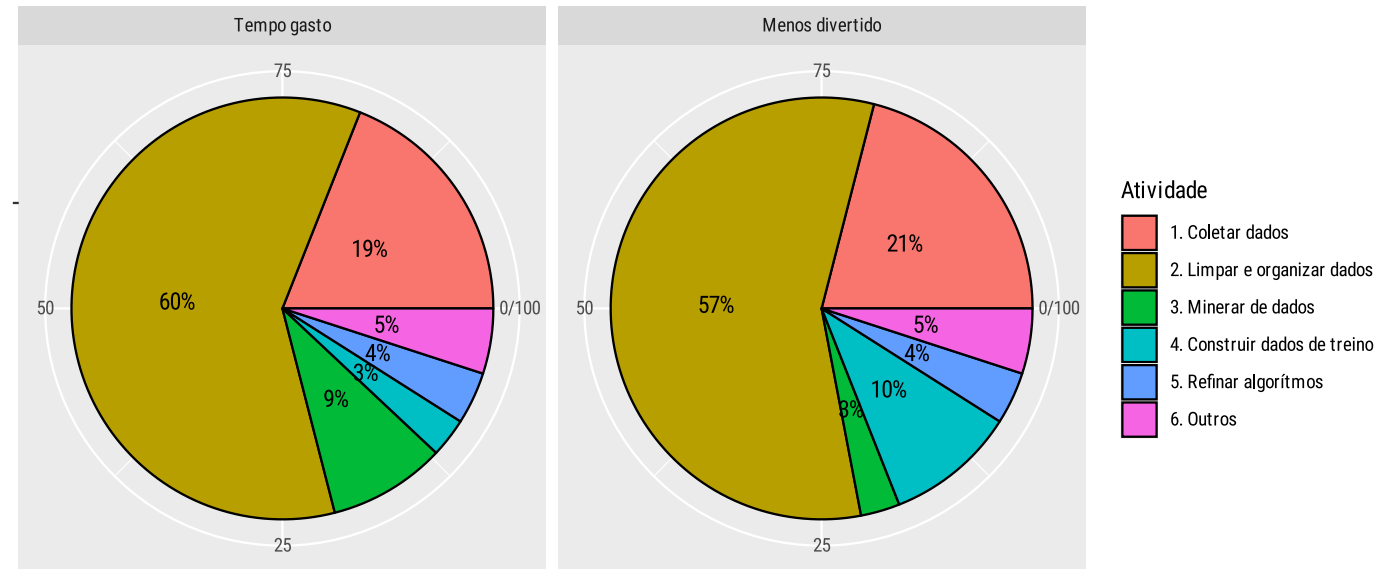


Ciclo de vida da ciência de dados. Fonte da imagem:

<https://bookdown.org/fjmcgrade/ismaykim/#intro-for-students>

# O tempo gasto nas tarefas em DS

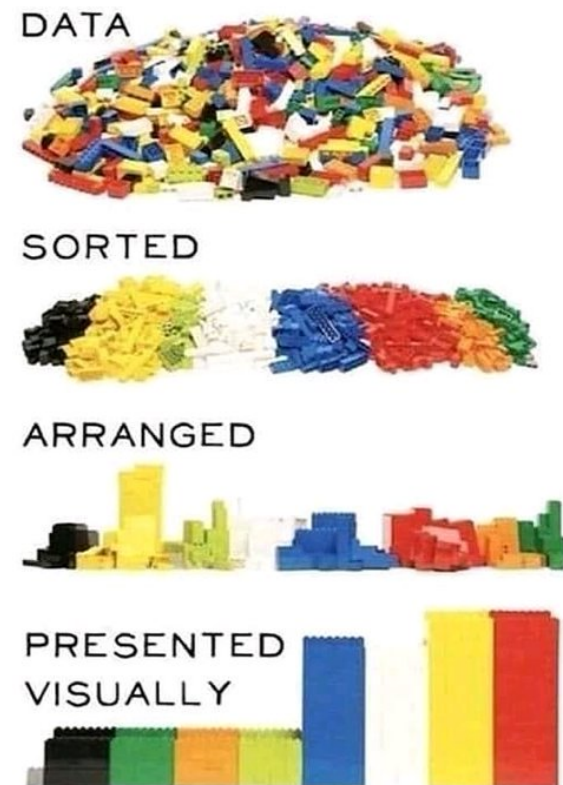
O que os cientistas de dados mais gastam tempo fazendo e como gostam disso?



Tempo gasto e diversão em atividades. Fonte: [Gil Press, 2016](#).

# O ambiente R para manipulação de dados

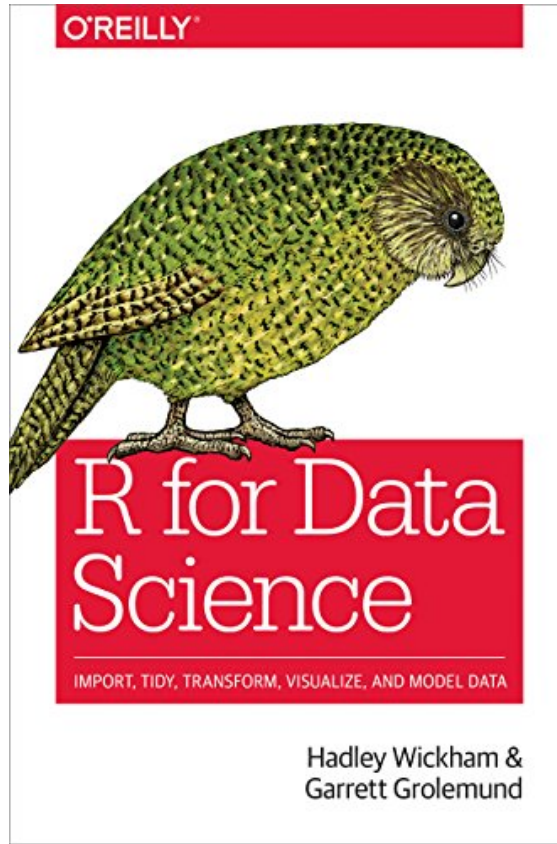
- ▶ O R é a lingua franca da Estatística.
- ▶ Desde o princípio oferece recursos para manipulação de dados.
  - ▶ O `data.frame` é a estrutura base para dados tabulares.
  - ▶ `base`, `utils`, `stats`, `reshape`, etc com recursos para importar, transformar, modificar, filtrar, agregar, `data.frames`.
- ▶ Porém, existem "algumas imperfeições" ou espaço para melhorias:
  - ▶ Coerções indesejadas de `data.frame`/matriz para vetor.
  - ▶ Ordem/nome irregular/inconsistente dos argumentos nas funções.
  - ▶ Dependência de pacotes apenas em cascata.



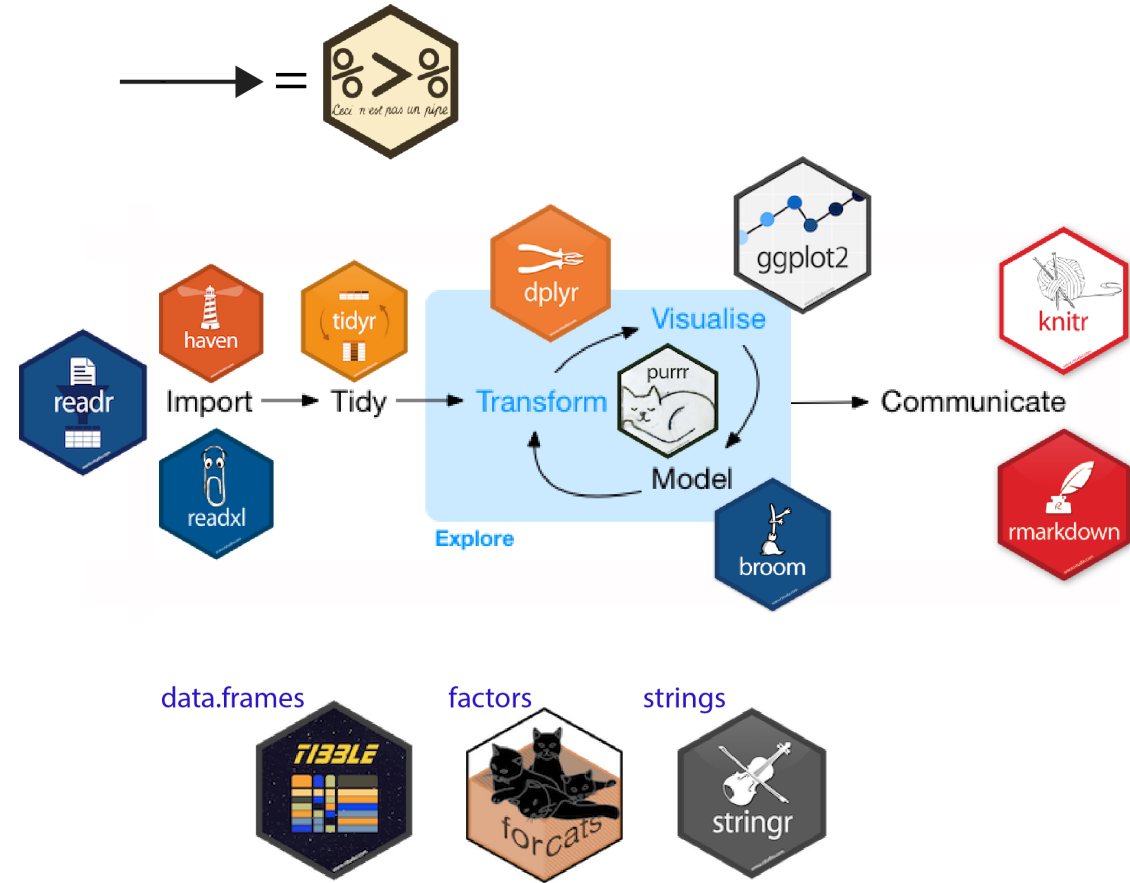
Uma ilustração do processo de transformação de dados em informação. Fonte:

<https://twitter.com/africadatasch/status/1301026743516639>

# R for Data Science



R for Data Science, a principal referência sobre o emprego da linguagem R em ciência de dados.



Workflow de ciência de dados com o {tidyverse}. Fonte:  
[https://oliviergimenez.github.io/intro\\_tidyverse/#7](https://oliviergimenez.github.io/intro_tidyverse/#7)

0 framework do {tidyverse}

# O {tidyverse}

- ▶ Oferece uma **reimplementação e extensão** das funcionalidades para manipulação e visualização.
- ▶ É uma coleção **8 de pacotes** R que operam em harmonia.
- ▶ Eles foram planejados e construídos para trabalhar em conjunto.
- ▶ Possuem gramática, organização, filosofia e estruturas de dados mais clara.
- ▶ Maior facilidade de desenvolvimento de código e portabilidade.
- ▶ Outros pacotes acoplam muito bem com o {tidyverse}.
- ▶ Pacotes: <https://www.tidyverse.org/packages/>.
- ▶ **R4DS**: <https://r4ds.had.co.nz/>.
- ▶ Cookbook: <https://rstudio-education.github.io/tidyverse-cookbook/program.html>.

- ▶ Para instalar o `tidyverse` (apenas uma vez)

```
install.packages("tidyverse")
```

- ▶ Isso instalará todos os pacotes que fazem parte dele
- ▶ Você pode carregar cada pacote separadamente
- ▶ Ou carregar direto o `tidyverse`, que carregará automaticamente os 8 principais de uma só vez

```
library(tidyverse)
```

```
tidyverse_packages() # Lista todos os pacotes
```

```
# [1] "broom"      "cli"
# [3] "crayon"     "dbplyr"
# [5] "dplyr"      "dtplyr"
# [7] "forcats"    "googledrive"
# [9] "googlesheets4" "ggplot2"
# [11] "haven"      "hms"
# [13] "httr"       "jsonlite"
# [15] "lubridate"  "magrittr"
# [17] "modelr"     "pillar"
# [19] "purrr"      "readr"
# [21] "readxl"     "reprex"
# [23] "rlang"      "rstudioapi"
# [25] "rvest"      "stringr"
# [27] "tibble"     "tidyr"
# [29] "xml2"       "tidyverse"
```

## Os pacotes do {tidyverse}



Pacotes que fazem parte do {tidyverse}.



# A anatomia do {tidyverse}

## {tibble}

- ▶ O `data.frame` é a estrutura nativa (primitiva) para representar tabelas de dados.
- ▶ O `tibble` é uma reimplementação da estrutura com melhorias.
  - ▶ Método `print` mais enxuto e informativo.
  - ▶ Mais consistente para seleção e modificação de conteúdo.
  - ▶ Mais fácil conversão de outros formatos para `tibble`.
  - ▶ Colunas/cédulas podem representar objetos mais complexos.
- ▶ Documentação:
  - ▶ <https://tibble.tidyverse.org/>.
  - ▶ <https://r4ds.had.co.nz/tibbles.html>.

## {readr}

- ▶ O `readr` tem recursos para importação de dados retangulares na forma de texto pleno.
- ▶ 10x mais rápido que R básico e 1.2-2x mais lento de `data.table`.
- ▶ Leitura/escrita de dados tabulares: `csv`, `tsv`, `fwf`.
  - ▶ Funções de importação: `read_*()`.
  - ▶ Funções de escrita: `write_*()`.
  - ▶ Funções de *parsing*: `parse_*`.
- ▶ Recursos "inteligentes" que determinam tipo de variável, como importar campos de datas como datas!
- ▶ Documentação:
  - ▶ <https://readr.tidyverse.org/>.
  - ▶ <https://r4ds.had.co.nz/data-import.html>.

# A anatomia do {tidyverse}

## {tidyr}

- ▶ Formato `tidy`:
  - ▶ Cada variável está em uma coluna.
  - ▶ Cada observação é uma linha.
  - ▶ Cada valor é uma célula.
- ▶ Principais recursos:
  - ▶ Mudar disposição dos dados: long/empilhar  $\rightleftharpoons$  wide/esparramar.
  - ▶ Lidar com valores ausentes.
  - ▶ Partir/concatenar variáveis.
  - ▶ Aninhar/desaninhar listas.
- ▶ Documentação:
  - ▶ <https://tidyr.tidyverse.org/>.
  - ▶ <https://r4ds.had.co.nz/tidy-data.html>.

## {dplyr}

- ▶ O `dplyr` é a **gramática** para manipulação de dados.
- ▶ Tem um conjunto **consistente** de verbos para atuar sobre tabelas.
  - ▶ Verbos: `mutate()`, `select()`, `filter()`, `arrange()`, `summarise()`, `slice()`, `rename()`, etc.
  - ▶ Sufixos: `_at()`, `_if()`, `_all()`, etc.
  - ▶ Extratificação: `group_by()` e `ungroup()`.
  - ▶ Junções: `inner_join()`, `full_join()`, `left_join()` e `right_join()`.
- ▶ Documentação:
  - ▶ <https://dplyr.tidyverse.org/>.
  - ▶ <https://r4ds.had.co.nz/relational-data.html>.

# A anatomia do {tidyverse}

## {ggplot2}

- ▶ Criação de gráficos baseado no *The Grammar of Graphics* Wilkinson et. al (2013).
- ▶ Claro mapeamento das variáveis do BD em variáveis visuais e construção baseada em camadas.
- ▶ Documentação: <https://ggplot2.tidyverse.org/>.
- ▶ Wickham (2016): `ggplot2` - Elegant Graphics for Data Analysis.
- ▶ Teutonico (2015): `ggplot2` Essentials.

## {purrr}

- ▶ O `purrr` fornece um conjunto **completo e consistente** para **programação funcional**.
- ▶ São uma sofisticação da família `apply`.
- ▶ Funções que aplicam funções em lote varrendo objetos: vetores, listas, etc.
- ▶ Várias função do tipo `map( )` para cada tipo de input/output.
- ▶ Percorrem vetores, listas, colunas, linhas, etc.
- ▶ Permitem filtrar, concatenar, parear listas, etc.
- ▶ Além disso, permite:
  - ▶ Chamar funções de forma não tradicional.
  - ▶ Aplicar funções para tratamento de excessões.
  - ▶ Operar de forma a acumular e reduzir recursivamente.
  - ▶ Aninhar e aplanar objetos.
- ▶ Documentação: <https://purrr.tidyverse.org/>.

# A anatomia do {tidyverse}

## {forcats}

- ▶ Para manipulação de variáveis categóricas/fatores.
- ▶ As principais operação são:
  - ▶ Renomenar: manualmente, programaticamente (truncar, abreviar, etc.).
  - ▶ Reordenar: manualmente, por frequência, por alguma variável.
  - ▶ Aglutinar: combinar níveis menos frequentes, etc.
- ▶ Documentação:
  - ▶ <https://forcats.tidyverse.org/>.
  - ▶ <https://peerj.com/preprints/3163/>.

## {stringr}

- ▶ Recursos coesos construídos para manipulação de *strings*.
- ▶ Feito sobre o pacote `stringi`.
- ▶ Praticamente tudo que envolva aplicação de expressões regulares.
  - ▶ Detectar.
  - ▶ Contar.
  - ▶ Partir.
  - ▶ Extrair.
  - ▶ Substituir.
  - ▶ etc.
- ▶ Documentação:
  - ▶ <https://stringr.tidyverse.org/>.

# Harmonizam bem com o {tidyverse}

## {lubridate} e {hms}

- ▶ Recursos para manipulação de dados *date*, *time* e *date-time*.
- ▶ Fácil decomposição de datas: dia, mês, semana, dia da semana, etc.
- ▶ Lida com fusos horários, horários de verão, etc.
- ▶ Estende para outras classes de dados baseados em *date-time*: duração, período, intervalos.
- ▶ Mas **não** é carregado junto com o `tidyverse`.

## {magrittr}

- ▶ O operador permite expressar de forma mais direta as operações.
- ▶ É uma ideia inspirada no Shell e usada em várias linguagens.
- ▶ A lógica é bem simples:
  - ▶ `x %>% f` é o mesmo que `f(x)`.
  - ▶ `x %>% f(y)` é o mesmo que `f(x, y)`.
  - ▶ `x %>% f %>% g %>% h` é o mesmo que `h(g(f(x)))`.
- ▶ Existem outros operadores *pipe* para situações específicas.

# Sobre os operadores pipe

## Pipe do `magrittr` (`%>%`)

- ▶ Criado em 2014, antes mesmo do `tidyverse`
- ▶ Foi incorporado ao `tidyverse` pela sua versatilidade
- ▶ Para usá-lo, é necessário carregar o pacote `magrittr` (ou o `tidyverse`)

```
x <- 0:10  
x %>% mean
```

```
# [1] 5
```

```
x %>% mean()
```

```
# [1] 5
```

```
## Possui um placeholder (.)  
mtcars %>% lm(mpg ~ disp, data = .) %>% coef()
```

```
# (Intercept)      disp  
# 29.59985476 -0.04121512
```

## Pipe nativo (`|>`)

- ▶ Lançado na versão 4.1.0 do R (2021)
- ▶ Não precisa carregar nenhum pacote adicional

```
x <- 0:10  
## x |> mean      # não pode ser usado sem ()  
x |> mean()
```

```
# [1] 5
```

```
## Não possui placeholder  
mtcars |>  
  (function(x) lm(mpg ~ disp, data = x))() |> coef()
```

```
# (Intercept)      disp  
# 29.59985476 -0.04121512
```

```
mtcars |> (\(x) lm(mpg ~ disp, data = x))() |> coef()
```

```
# (Intercept)      disp  
# 29.59985476 -0.04121512
```

# Mais sobre manipulação de dados no R

- ▶ `{tidyverse}`:

- ▶ É uma coleção de pacotes.
- ▶ Principais: `tibble`, `readr`, `dplyr`, `tidyr`.
- ▶ Acessórios: `ggplot2`, `purrr`, `stringr`, `forcats`.

- ▶ R básico.

- ▶ Classe `data.frame`.
- ▶ Métodos disponíveis no pacote `base`, `reshape`, `reshape2`, etc.
- ▶ Resolve muitos problemas mas tem espaços para melhorias.
- ▶ `{data.table}`: número 1 em performance para operações de manipulação de dados.
- ▶ `{sqldf}`: manipulação de `data.frames` com instruções SQL.
- ▶ Além disso, muitos drives para bancos de dados e frameworks.

# 0 {data.table}

## DATA TABLES

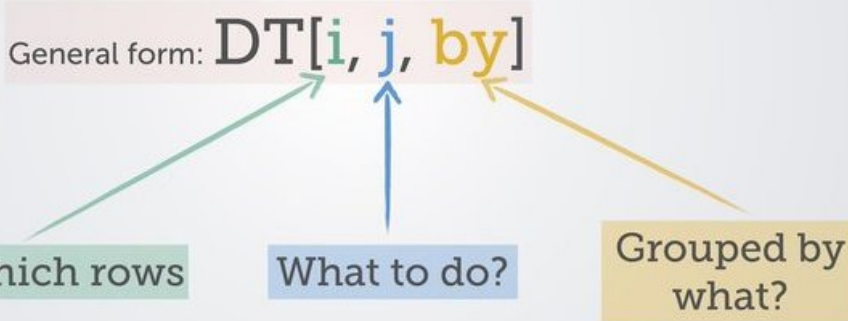
- think in terms of basic units — **rows**, **columns** and **groups**
- data.table syntax provides *placeholder* for each of them

General form: **DT[i, j, by]**

On which rows

What to do?

Grouped by what?



```
library(data.table)
```

```
iris_dt <- as.data.table(iris)
iris_dt[,
  list(SPmean = mean(Sepal.Length),
       PLmean = mean(Petal.Length)),
  by = Species]
```

#	Species	SPmean	PLmean
# 1:	setosa	5.006	1.462
# 2:	versicolor	5.936	4.260
# 3:	virginica	6.588	5.552

Sintaxe do {data.table}. Para uma visão geral, visite a wiki do projeto:

<https://github.com/Rdatatable/data.table/wiki>.



## 0 {sqldf}

- ▶ Permite uso de expressões SQL para operar em `data.frames`.
- ▶ Cria um banco de dados e faz as operações lá, trazendo os resultados como `data.frames`.
- ▶ Suporta vários backends: `RSQLite`, `RH2`, `RMySQL` and `RPostgreSQL`.
- ▶ Tutoriais rápidos:
  - ▶ [https://jasminedaly.com/tech-short-papers/sqldf\\_tutorial.html](https://jasminedaly.com/tech-short-papers/sqldf_tutorial.html).
  - ▶ <http://dept.stat.lsa.umich.edu/~jerrick/courses/stat701/notes/sql.html>.
  - ▶ <https://anythingbutrbitrary.blogspot.com/2012/08/manipulating-data-frames-using-sqldf.html>.
- ▶ Recomendado para:
  - ▶ Conhecedores de SQL.
  - ▶ Situações em que exigem maior desempenho.

```
library(sqldf)

sqldf(paste("SELECT Species,",
            "AVG(`Sepal.Length`) AS SLmean,",
            "AVG(`Petal.Length`) AS PLmean",
            "FROM iris",
            "GROUP BY Species"))
```

#	Species	SLmean	PLmean
# 1	setosa	5.006	1.462
# 2	versicolor	5.936	4.260
# 3	virginica	6.588	5.552

# Considerações finais

## Sobre o {tidyverse}

- ▶ É um framework para manipulação de dados no R.
- ▶ Torna mais fácil, ágil e portátil a escrita de código para manipulação de dados no R.
- ▶ Os pacotes foram feitos para trabalhar em harmonia.
- ▶ Não é a única forma de trabalhar com dados no R, mas é a mais popular em data science.

## O que vem agora?

- ▶ Uso cada pacote do tidyverse.
- ▶ Exemplos didáticos seguidos de desafios práticos.
- ▶ *Happy coding.*

Dados tabulares com `tibble`

## Um overview do {tibble}

- ▶ O `data.frame` é a estrutura nativa (primitiva) para representar tabelas de dados.
- ▶ O `tibble` é uma reimplementação da estrutura com melhorias.
  - ▶ Método `print` mais enxuto e informativo.
  - ▶ Mais consistente para seleção e modificação de conteúdo.
  - ▶ Mais fácil conversão de outros formatos para `tibble`.
  - ▶ Colunas/cédulas podem representar objetos mais complexos.
- ▶ Documentação:
  - ▶ <https://tibble.tidyverse.org/>.
  - ▶ <https://r4ds.had.co.nz/tibbles.html>.
  - ▶ <https://cran.r-project.org/package=tibble>

# Formas de criar um tibble

## Por colunas (*column-by-column*)

```
library(tidyverse)
```

```
tb <- tibble(x = 1:3,  
            y = letters[1:length(x)],  
            z = x^2)  
tb
```

```
# # A tibble: 3 × 3  
#       x y     z  
#   <int> <chr> <dbl>  
# 1     1 a     1  
# 2     2 b     4  
# 3     3 c     9
```

```
class(tb)
```

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

Com `data.frame` isso não seria possível

```
ta <- data.frame(x = 1:3,  
                y = letters[1:length(x)],  
                z = x^2)
```

```
# Error in data.frame(x = 1:3, y = letters[1:length(x)], z = x^2): arguments imply differing number of rows: 3, 11
```

## Por linhas (*row-by-row*)

```
tr <- tribble(  
  ~x, ~y, ~z,  
  "a", 2, 3.6,  
  "b", 1, 8.5  
)  
tr
```

```
# # A tibble: 2 × 3  
#       x     y     z  
#   <chr> <dbl> <dbl>  
# 1 a     2     3.6  
# 2 b     1     8.5
```

```
class(tr)
```

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

# Formas de criar um tibble

## Coerção de vetores nomeados

```
notas <- c("André" = 6,  
          "Larissa" = 9,  
          "Mariana" = 8,  
          "Tobias" = 3)
```

```
notas
```

```
#   André Larissa Mariana Tobias  
#     6       9       8     3
```

```
notas <- notas %>%  
  enframe(name = "aluno", value = "nota")  
notas
```

```
# # A tibble: 4 × 2  
#   aluno    nota  
#   <chr>   <dbl>  
# 1 André     6  
# 2 Larissa   9  
# 3 Mariana   8  
# 4 Tobias    3
```

```
class(notas)
```

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

## Coerção de matrizes

```
matrix(1:12, ncol = 3) %>%  
  as_tibble()
```

```
# # A tibble: 4 × 3  
#       V1     V2     V3  
#   <int> <int> <int>  
# 1     1     5     9  
# 2     2     6    10  
# 3     3     7    11  
# 4     4     8    12
```

## Coerção de data.frames

```
iris %>% as_tibble() %>% print(n = 5)
```

```
# # A tibble: 150 × 5  
#   Sepal.Length Sepal.Width Petal.Length  
#         <dbl>         <dbl>         <dbl>  
# 1         5.1           3.5           1.4  
# 2         4.9           3             1.4  
# 3         4.7           3.2           1.3  
# 4         4.6           3.1           1.5  
# 5           5           3.6           1.4  
# # ... with 145 more rows, and 2 more  
# #   variables: Petal.Width <dbl>,  
# #   Species <fct>
```

# Operações com tibble

## Seleção

```
tb <- iris[1:4, ] %>%  
  as_tibble()  
## Resulta em vetor  
tb$Petal.Length
```

```
# [1] 1.4 1.4 1.3 1.5
```

```
tb[["Petal.Length"]]
```

```
# [1] 1.4 1.4 1.3 1.5
```

```
tb[[3]]
```

```
# [1] 1.4 1.4 1.3 1.5
```

```
## Resulta em `tibble`!  
tb[, 3]
```

```
# # A tibble: 4 × 1  
#   Petal.Length  
#         <dbl>  
# 1         1.4  
# 2         1.4  
# 3         1.3  
# 4         1.5
```

```
tb[, "Petal.Length"]
```

```
# # A tibble: 4 × 1  
#   Petal.Length  
#         <dbl>  
# 1         1.4  
# 2         1.4  
# 3         1.3  
# 4         1.5
```

```
tb[, c(3:4)]
```

```
# # A tibble: 4 × 2  
#   Petal.Length Petal.Width  
#         <dbl>         <dbl>  
# 1         1.4         0.2  
# 2         1.4         0.2  
# 3         1.3         0.2  
# 4         1.5         0.2
```

```
tb[1, ]
```

```
# # A tibble: 1 × 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
```

# Operações com tibble

## Adicionar colunas

```
notas <- c("André" = 6,  
          "Larissa" = 9,  
          "Mariana" = 8,  
          "Tobias" = 3)  
notas <- notas %>%  
  enframe(name = "aluno", value = "nota")  
  
notas <- notas %>%  
  add_column(faltas = c(12, 8, 0, 18))  
notas
```

```
# # A tibble: 4 × 3  
#   aluno      nota faltas  
#   <chr>    <dbl> <dbl>  
# 1 André         6     12  
# 2 Larissa       9      8  
# 3 Mariana       8      0  
# 4 Tobias        3     18
```

## Adicionar linhas

```
notas <- notas %>%  
  add_row(aluno = c("Roberto", "Sabrina"),  
          nota = c(8, 7))  
notas
```

```
# # A tibble: 6 × 3  
#   aluno      nota faltas  
#   <chr>    <dbl> <dbl>  
# 1 André         6     12  
# 2 Larissa       9      8  
# 3 Mariana       8      0  
# 4 Tobias        3     18  
# 5 Roberto       8     NA  
# 6 Sabrina       7     NA
```



## Exercícios para usar o `tibble`

Criar um `tibble` a partir:

1. do objeto `precip`.
2. do objeto `cars`.
3. do objeto `mtcars`.
4. do objeto `WorldPhones`.
5. do objeto `HairEyeColor`.

Importação de dados com `readr` e `readxl`

## Um overview do {readr}

- ▶ O processo de análise de dados começa com a importação dos dados para o ambiente de manipulação.
- ▶ Existem várias meios para armazenar dados.
  - ▶ Arquivos de texto plano (tsv, txt, csv, etc).
  - ▶ Planilhas eletrônicas.
  - ▶ Bancos de dados relacionais.
  - ▶ Etc.
- ▶ O `readr` tem recursos para importação de dados retangulares na forma de texto plano.
- ▶ Documentação:
  - ▶ <https://readr.tidyverse.org/>.
  - ▶ <https://r4ds.had.co.nz/data-import.html>.
  - ▶ <https://cran.r-project.org/package=readr>.

# Funções para importação e exportação

## Funções de leitura

```
ls("package:readr") %>%  
  str_subset("^read_")
```

```
# [1] "read_builtin"  
# [2] "read_csv"  
# [3] "read_csv_chunked"  
# [4] "read_csv2"  
# [5] "read_csv2_chunked"  
# [6] "read_delim"  
# [7] "read_delim_chunked"  
# [8] "read_file"  
# [9] "read_file_raw"  
# [10] "read_fwf"  
# [11] "read_lines"  
# [12] "read_lines_chunked"  
# [13] "read_lines_raw"  
# [14] "read_lines_raw_chunked"  
# [15] "read_log"  
# [16] "read_rds"  
# [17] "read_table"  
# [18] "read_table2"  
# [19] "read_tsv"  
# [20] "read_tsv_chunked"
```

## Funções de escrita

```
ls("package:readr") %>%  
  str_subset("^write_")
```

```
# [1] "write_csv"  
# [2] "write_csv2"  
# [3] "write_delim"  
# [4] "write_excel_csv"  
# [5] "write_excel_csv2"  
# [6] "write_file"  
# [7] "write_lines"  
# [8] "write_rds"  
# [9] "write_tsv"
```

# Principais argumentos

```
## Argumentos da `read_csv2()`.  
args(read_csv2)
```

```
# function (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), progress = show_progress(),  
#   name_repair = "unique", num_threads = readr_threads(), show_col_types = should_show_types(),  
#   skip_empty_rows = TRUE, lazy = should_read_lazy())  
# NULL
```

```
## Argumentos da `write_csv2()`.  
args(write_csv2)
```

```
# function (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), progress = show_progress(),  
#   name_repair = "unique", num_threads = readr_threads(), show_col_types = should_show_types(),  
#   skip_empty_rows = TRUE, lazy = should_read_lazy())  
# NULL
```

```
## Argumentos da `locale()`.  
args(locale)
```

```
# function (date_names = "en", date_format = "%AD", time_format = "%AT",  
#   decimal_mark = ".", grouping_mark = ",", tz = "UTC", encoding = "UTF-8",  
#   asciify = FALSE)  
# NULL
```

# As funções de parsing

- ▶ Uma das maiores vantagens do `readr` é a maior flexibilidade para determinar o tipo de valor dos campos.
- ▶ As funções para o *parsing* (exame) são usadas para atribuir o tipo de valor apropriado durante a importação.
- ▶ Isso é economia de tempo.

```
ls("package:readr") %>%  
  str_subset("^parse_")
```

```
# [1] "parse_character" "parse_date"  
# [3] "parse_datetime" "parse_double"  
# [5] "parse_factor"    "parse_guess"  
# [7] "parse_integer"   "parse_logical"  
# [9] "parse_number"    "parse_time"  
# [11] "parse_vector"
```

```
# Conversão para data e data-tempo.  
parse_date("2018/12/25")
```

```
# [1] "2018-12-25"
```

```
parse_datetime("20181225")
```

```
# [1] "2018-12-25 UTC"
```

```
parse_datetime("2018-12-25T12:10:00")
```

```
# [1] "2018-12-25 12:10:00 UTC"
```

```
# Número com separador de milhar.  
guess_parser("1,234,566")
```

```
# [1] "number"
```

```
parse_guess("1,234,566")
```

```
# [1] 1234566
```

```
# Datas.  
guess_parser(c("2010-10-10"))
```

```
# [1] "date"
```

```
parse_guess(c("2010-10-10"))
```

```
# [1] "2010-10-10"
```

## Importação de dados com {readr}

- ▶ O argumento obrigatório é o caminho para o arquivo.
- ▶ Argumentos opcionais existem pra um controle detalhado das opções de importação.

```
url <- "http://leg.ufpr.br/~walmes/data/anovareg.txt"
## Default
tb <- read_tsv(file = url)
str(tb)
## Tipo de valores para cada variável.
tb <- read_tsv(file = url, col_types = "cicd")
## Renomeia os campos.
tb <- read_tsv(file = url, skip = 1,
               col_names = c("clt", "ntr", "blc", "index"))
str(tb)
```

- ▶ A lista de especificações de tipo de valor garante que variáveis com certos nomes serão importadas com o tipo de valor definido.
- ▶ Ou seja, em todo arquivo que houver **bloco**, ele será importado como fator.
- ▶ Não precisa estar na ordem e nem conter todas as variáveis.

```
## Lista de especificações.
specs <- cols(
  cultivar = col_factor(levels = NULL),
  bloco = col_factor(levels = NULL)
  # dose = col_integer(),
  # indice = col_integer()
)
## Usando a lista.
tb <- read_tsv(file = url, col_types = specs)
str(tb)
```

## Dados da aula passada

```
## Com read.table
da <- read.table("dados/JF_Secao_25_Ago_2020.csv", sep = ";",
                 dec = ",", header = TRUE, encoding = "latin1")
str(da)
## Com read.csv2
da <- read.csv2("dados/JF_Secao_25_Ago_2020.csv",
                encoding = "latin1")
str(da)
## Com readr::read_csv2
da <- read_csv2("dados/JF_Secao_25_Ago_2020.csv")
str(da)
## Especificando o encoding
da <- read_csv2("dados/JF_Secao_25_Ago_2020.csv",
                locale = locale(encoding = "latin1"))
str(da)
## Para tirar o spec e problems
attr(da, "spec") <- NULL
attr(da, "problems") <- NULL
## OU, de uma só vez (um subset vazio)
da <- da[]
str(da)
```



# Um overview do {readxl}

- ▶ O `readxl` funciona basicamente igual ao `readr`
  - ▶ Retorna um `tibble`
- ▶ A diferença é que arquivos do Excel podem ser lidos diretamente
- ▶ Suporta tanto arquivos `.xls` quanto `.xlsx`
- ▶ Não possui dependências externas como outros pacotes (`gdata`, `xlsx`, etc)
- ▶ Funciona com dados **tabulares**
- ▶ Para dados não tabulares, o pacote `tidyxl` pode ser usado
- ▶ **Não** tem funções para exportação. Veja
  - ▶ `openxlsx`
  - ▶ `writexl`
- ▶ Documentação:
  - ▶ <https://readxl.tidyverse.org>
  - ▶ <https://github.com/rstudio/cheatsheets/blob/main/data-import.pdf>

## Funções de leitura

```
library(readxl)
ls("package:readxl") %>%
  str_subset("^read_")

# [1] "read_excel" "read_xls"
# [3] "read_xlsx"
```

# Dados novos

```
library(readxl)
url <- "dados/Base de Dados Justiça Estadual 2020 CNJ.xlsx"
da <- read_excel(url)
## Abra o arquivo para ver que a planilha não tem apenas a tabela
str(da)

## Especifica o range de linhas e colunas a serem lidas
da <- read_excel(url, range = "A3:Q31")
str(da)
```