8 Data Import

Introdução à ciência de dados

Daniel Brito dos Santos

8.1 Introdução

- Usar os dados fornecidos por pacotes de R é uma ótima forma de aprender as ferramentas da ciência de dados
- Mas você vai querer aplicar o que aprendeu nos seus próprios dados!
- Na aula de hoje veremos o básico da leitura de arquivos no R

Especificamente:

- vamos focar em arquivos **tabulares de texto** (csv)
- veremos dicas práticas de como lidar com **colunas**, **nomes**, **tipos** e **dados faltantes**.
- depois veremos como ler dados de **arquivos múltiplos** e **salvar** arquivos
- finalmente veremos como **criar datasets** manualmente

8.1.1 Prerequisitos

- Vamos utilizar o pacote **readr** que é parte do tidyverse

. . .

```
library(tidyverse)
```

- Também vamos mencionar o pacote **janitor** para facilitar certas manipulações de dados

. . .

```
install.packages("janitor")
library(janitor)
```

8.2 Lendo dados de um arquivo

CSV

- formato mais famoso de dados retangulares
- primeira linha normalmente é **header**
  - nomeia as colunas
- linhas seguintes apresenta os dados

Exemplo

. . .

```
#> Student ID,Full Name,favourite.food,mealPlan,AGE
#> 1,Sunil Huffmann,Strawberry yoghurt,Lunch only,4
#> 2,Barclay Lynn,French fries,Lunch only,5
#> 3,Jayendra Lyne,N/A,Breakfast and lunch,7
#> 4,Leon Rossini,Anchovies,Lunch only,
#> 5,Chidiegwu Dunkel,Pizza,Breakfast and lunch,five
#> 6,Güvenç Attila,Ice cream,Lunch only,6
```

Tabela

Student ID	Full Name	favourite.food	mealPlan	AGE
1	Sunil Huffmann	Strawberry yoghurt	Lunch only	4
2	Barclay Lynn	French fries	Lunch only	5
3	Jayendra Lyne	N/A	Breakfast and lunch	7
4	Leon Rossini	Anchovies	Lunch only	
5	Chidiegwu Dunkel	Pizza	Breakfast and lunch	five
6	Güvenç Attila	Ice cream	Lunch only	6

Função `read_csv()`

- Como o nome sugere, é a função para ler CSVs no R
- Seu primeiro argumento é o caminho do arquivo

. . .

```
students <- read_csv("data/students.csv")
```

```
students <- read_csv("data/students.csv")
#> Rows: 6 Columns: 5
#>   Column specification
#> Delimiter: ","
#> chr (4): Full Name, favourite.food, mealPlan, AGE
#> dbl (1): Student ID
#>
#>  Use `spec()` to retrieve the full column specification for this data.
#>  Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

- Quando você roda a função ela apresenta uma mensagem com:
  - número de linhas e colunas
  - o delimitador utilizado
  - especificações das colunas organizdas pelo tipo de dados que a coluna contém
  - informações sobre como recuperar a especificação da completa da coluna (já vamos falar disso)
  - como fazer essa mensagem sumir

8.2.1 Conselho prático

- O primeiro passo após a leitura dos dados é deixá-los de um jeito que facilite o resto da análise
- Com isso em mente:
  - Vamos dar mais uma olhada no `students`

```
# A tibble: 6 × 5
  `Student ID` `Full Name`      favourite.food     mealPlan             AGE
         <dbl> <chr>            <chr>              <chr>                <chr>
1            1 Sunil Huffmann   Strawberry yoghurt Lunch only           4
2            2 Barclay Lynn     French fries       Lunch only           5
3            3 Jayendra Lyne    N/A                Breakfast and lunch  7
4            4 Leon Rossini     Anchovies          Lunch only           NA
5            5 Chidiegwu Dunkel Pizza              Breakfast and lunch  five
6            6 Güvenç Attila    Ice cream          Lunch only           6
```

- Na coluna `favourite.food` tem alguns alimentos e uma string "N/A" que deveria ser um `NA` que o R entende como "não disponível"
- Pedemos resolver isso definindo quais stings devem ser interpretadas como `NA`

. . .

```
students <- read_csv("data/students.csv", na = c("N/A", ""))
```

```
# A tibble: 6 × 5
  `Student ID` `Full Name`      favourite.food     mealPlan             AGE
         <dbl> <chr>            <chr>              <chr>                <chr>
1            1 Sunil Huffmann   Strawberry yoghurt Lunch only           4
2            2 Barclay Lynn     French fries       Lunch only           5
3            3 Jayendra Lyne    NA                 Breakfast and lunch  7
4            4 Leon Rossini     Anchovies          Lunch only           NA
5            5 Chidiegwu Dunkel Pizza              Breakfast and lunch  five
6            6 Güvenç Attila    Ice cream          Lunch only           6
```

- Perceberam que `Student ID` e `Full Name` estão entre crases (‘) ?
- Isso é porque elas contém espaço, o que viola a regra para nomes de variáveis
- Como resolver?

. . .

```
students |>
  rename(
    student_id = `Student ID`,
    full_name = `Full Name`
```

```
  )
```

- Alternativamente podemos usar o`janitor::clean_names()`

. . .

```
students |> janitor::clean_names()
#> # A tibble: 6 × 5
#>   student_id full_name        favourite_food     meal_plan           age
#>        <dbl> <chr>            <chr>              <chr>               <chr>
#> 1          1 Sunil Huffmann   Strawberry yoghurt Lunch only          4
#> 2          2 Barclay Lynn     French fries       Lunch only          5
#> 3          3 Jayendra Lyne    <NA>               Breakfast and lunch 7
#> 4          4 Leon Rossini     Anchovies          Lunch only          <NA>
#> 5          5 Chidiegwu Dunkel Pizza              Breakfast and lunch five
#> 6          6 Güvenç Attila    Ice cream          Lunch only          6
```

- Essa função automaticamente transforma os nomes para “snake_case”

```
# A tibble: 6 × 5
  student_id full_name        favourite_food     meal_plan           age
       <dbl> <chr>            <chr>              <chr>               <chr>
1          1 Sunil Huffmann   Strawberry yoghurt Lunch only          4
2          2 Barclay Lynn     French fries       Lunch only          5
3          3 Jayendra Lyne    NA                 Breakfast and lunch 7
4          4 Leon Rossini     Anchovies          Lunch only          NA
5          5 Chidiegwu Dunkel Pizza              Breakfast and lunch five
6          6 Güvenç Attila    Ice cream          Lunch only          6
```

- Outra tarefa comum após a leitura é considerar os tipos de variáveis.
- A variável `meal_type` é categórica com um conjunto conhecido de valores possíveis
- Esse tipo de variável deve ser representada como factor:

```
students |>
  janitor::clean_names() |>
  mutate(
    meal_plan = factor(meal_plan)
  )
#> # A tibble: 6 × 5
#>   student_id full_name        favourite_food     meal_plan           age
#>        <dbl> <chr>            <chr>              <fct>               <chr>
#> 1          1 Sunil Huffmann   Strawberry yoghurt Lunch only          4
#> 2          2 Barclay Lynn     French fries       Lunch only          5
#> 3          3 Jayendra Lyne    <NA>               Breakfast and lunch 7
#> 4          4 Leon Rossini     Anchovies          Lunch only          <NA>
#> 5          5 Chidiegwu Dunkel Pizza              Breakfast and lunch five
#> 6          6 Güvenç Attila    Ice cream          Lunch only          6
```

- Perceba que o `meal_type` mudou de caráctere (`<chr>`) para factor (`<fct>`)

```
# A tibble: 6 × 5
  student_id full_name        favourite_food     meal_plan           age
       <dbl> <chr>            <chr>              <fct>               <chr>
1          1 Sunil Huffmann   Strawberry yoghurt Lunch only          4
2          2 Barclay Lynn     French fries       Lunch only          5
3          3 Jayendra Lyne    NA                 Breakfast and lunch 7
4          4 Leon Rossini     Anchovies          Lunch only          NA
5          5 Chidiegwu Dunkel Pizza              Breakfast and lunch five
6          6 Güvenç Attila    Ice cream          Lunch only          6
```

- Você também vai querer consertar a coluna `age`
- Atualmente é uma variável de caracteres porque um dos valores está `five` ao invés do número `5`:

. . .

```
students <- students |>
  janitor::clean_names() |>
  mutate(
    meal_plan = factor(meal_plan),
    age = parse_number(if_else(age == "five", "5", age))
  )
```

```
# A tibble: 6 × 5
  student_id full_name        favourite_food     meal_plan             age
       <dbl> <chr>            <chr>              <fct>                 <dbl>
1          1 Sunil Huffmann   Strawberry yoghurt Lunch only              4
2          2 Barclay Lynn     French fries       Lunch only              5
3          3 Jayendra Lyne    NA                 Breakfast and lunch     7
4          4 Leon Rossini     Anchovies          Lunch only             NA
5          5 Chidiegwu Dunkel Pizza              Breakfast and lunch     5
6          6 Güvenç Attila    Ice cream          Lunch only              6
```

8.2.2 Outros argumentos importantes

Antes vamos aprender um truque

- A função `read_csv()` pode ler uma string no próprio console

. . .

```
read_csv(
  "a,b,c
  1,2,3
  4,5,6"
)
#> # A tibble: 2 × 3
#>       a     b     c
#>   <dbl> <dbl> <dbl>
#> 1     1     2     3
#> 2     4     5     6
```

- `read_csv()` utiliza por padrão a primeira linha para os nomes das colunas
- Mas as vezes nosso csv tem metadados antes
- Podemos pular n linhas (`skip = n`)
- Ou definir comentários (`comment = "#"`)

```
read_csv(
  "The first line of metadata
  The second line of metadata
  x,y,z
  1,2,3",
  skip = 2
)
#> # A tibble: 1 × 3
#>       x     y     z
#>   <dbl> <dbl> <dbl>
#> 1     1     2     3
```

. . .

```
read_csv(
  "# A comment I want to skip
  x,y,z
  1,2,3",
  comment = "#"
)
#> # A tibble: 1 × 3
#>       x     y     z
#>   <dbl> <dbl> <dbl>
#> 1     1     2     3
```

- Às vezes não temos nenhuma linha com nomes de coluna
- Podemos utilizar `col_names = FALSE` ou definir manualmente com um vetor

. . .

```
read_csv(
  "1,2,3
  4,5,6",
  col_names = FALSE
)
#> # A tibble: 2 × 3
#>      X1    X2    X3
#>   <dbl> <dbl> <dbl>
#> 1     1     2     3
#> 2     4     5     6
```

. . .

```
read_csv(
  "1,2,3
  4,5,6",
  col_names = c("x", "y", "z")
)
#> # A tibble: 2 × 3
#>       x     y     z
#>   <dbl> <dbl> <dbl>
#> 1     1     2     3
#> 2     4     5     6
```

Pronto!

- O que vimos até aqui te permite ler a grande maioria dos CSVs que vai encontrar por aí
- As excessões você tem que inspecionar com cuidado o arquivo e ler a documentação da `read_csv()`

8.2.3 Outros formatos de arquivo

- Depois de compreender o `read_csv()` é fácil utilizar as outras funções do **readr**
  - `read_csv2()` - lê os CSVs separados por ; ao invés de ,
  - `read_tsv()` - lê arquivos separados por tabs
  - `read_delim()` - lê arquivos separados por qualquer delimitador
  - `read_fwf()` - lê arquivos com largura fixa
  - `read_table()` - lê arquivos com separação fixa de espaços em branco
  - `read_log()` - lê logs estilo Apache

8.3 Controlando os tipos de dados das colunas

- Arquivos CSV não contém **nenhuma informação** sobre o **tipo** de cada variável
  - (se ela é lógica, numérica, string, etc)
- O **readr** vai tentar advinhar o tipo de dado

  - vamos ver como esse processo funciona
  - aprender a resolver falhas comuns
  - como nós mesmos deinirmos os tipos
  - Estratégias do que fazer quando o a leitura falha catastróficamente

8.3.1 Advinhando tipos de dados

- readr usa uma **heurística** para atribuir tipo as colunas
- Para cada coluna o readr:

- seleciona os valores 1 000 linhas igualmente espaçados, ignorando NAs
- a partir dessa amostra ele responde essas perguntas:
  - os valores contém apenas `F`, `T`, `FALSE`, ou `TRUE` (case insentive)? Se sim: tipo lógico
  - Os valores contém apenas números (`1`, `-4.5`, `5e6`, `Inf`)? Então: tipo numérico
  - Os valores correspondem ao padrão ISO8601? Então é data ou date-time
  - Do contrário é uma string
- Essa heurística funciona bem se tiver um dataset limpo, mas na vida real …

8.3.2 Valores faltantes, tipos de colunas e problemas

- A falha mais comum na detecção de colunas é quando ela contém valores inesperados
- O caso mais comum é registrar um valor faltante utilizando algo diferente de `NA`

. . .

```
csv <- "
  x
  10
  .
  20
  30"

df <- read_csv(csv)
```

```
#> Rows: 4 Columns: 1
#>   Column specification
#> Delimiter: ","
#> chr (1): x
#>
#>  Use `spec()` to retrieve the full column specification for this data.
#>  Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

- Nesse exemplo a coluna x é interpretada como caractere
- Como poderíamos resolver?
- Isso, `na = "."`
- Mas e se fossem milhares de linhas com poucos valores faltantes?

- Uma forma de resolver é dizer pra readr que a coluna é numérica e ver onde a leitura falha
- Podemos utilizar o `col_types` para isso:

. . .

```
df <- read_csv(csv, col_types = list(x = col_double()))
#> Warning: One or more parsing issues, call `problems()` on your data frame for
#> details, e.g.:
#>   dat <- vroom(...)
#>   problems(dat)
```

- Observem a mensagem avisando que houve um problema
- Vamos investigar!

Investigando problemas em datasets

```
problems(df)
#> # A tibble: 1 × 5
#>     row   col expected actual file
#>   <int> <int> <chr>    <chr>  <chr>
#> 1     3     1 a double .      /tmp/Rtmps4rcCC/file1fd82862f609
```

- Quem gostaria de tentar interpretar?
- Essa função nos mostra que houve um problema na linha 3, coluna 1
- O readr esperava um valor do tipo `double`, mas encontrou o "."
- Isso nos sugere que o dataset utiliza "." para valores faltantes
- Agora sim podemos informar ao readr pra fazer o chute correto:

```
df <- read_csv(csv, na = ".")
#> Rows: 4 Columns: 1
#>   Column specification
#> Delimiter: ","
#> dbl (1): x
#>
#>  Use `spec()` to retrieve the full column specification for this data.
#>  Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

8.3.3 Tipos de colunas

- readr oferece nove tipos de colunas para utilizarmos na leitura:
  - `col_logical()` e `col_double()` para tipos lógicos e números reais. São raramente usadas porque o readr costuma acertar esse chute
  - `col_integer()` para inteiros. Funcionalmenete idênticos aos reais mas ocupam metade da memória.
  - `col_character()` para strings. É útil para explicitar uma coluna com identifiador numérico. Por exemplo, não faz sentido seu número de matrícula por dois, ser uma string garante isso.

- `col_factor`, `col_data()`, `col_datetime()` para fatores, datas e date-times.
- `col_number()` é um parser numérico que ignora componentes não numéricos, particularmente útil para moedas.
- `col_skip()` para pular a leitura de determinada coluna.

- Outra função útil é `cols_only` para ler apenas as colunas que você especificar:

. . .

```
read_csv(
  "x,y,z
  1,2,3",
  col_types = cols_only(x = col_character())
)
#> # A tibble: 1 × 1
#>   x
#>   <chr>
#> 1 1
```

8.4 Lendo dados de múltiplos arquivos

- As vezes nossos dados estão separados em vários arquivos
- Um exemplo são dados de vendas em múltiplos meses:
  - `01-sales.csv` para janeiro
  - `02-sales.csv` para fevereiro e
  - `03-sales.csv` para março
- você pode usar o `read_csv()` para empilhar todos esses dados em apenas um dataframe

---

. . .

```
sales_files <- c("data/01-sales.csv", "data/02-sales.csv", "data/03-sales.csv")
read_csv(sales_files, id = "file")
#> Rows: 19 Columns: 6
#>   Column specification
#> Delimiter: ","
#> chr (1): month
#> dbl (4): year, brand, item, n
#>
#>  Use `spec()` to retrieve the full column specification for this data.
#>  Specify the column types or set `show_col_types = FALSE` to quiet this message.
#> # A tibble: 19 × 6
#>   file              month    year brand  item     n
#>   <chr>             <chr>   <dbl> <dbl> <dbl> <dbl>
#> 1 data/01-sales.csv January  2019     1  1234     3
#> 2 data/01-sales.csv January  2019     1  8721     9
#> 3 data/01-sales.csv January  2019     1  1822     2
```

```
#> 4 data/01-sales.csv January  2019     2  3333     1
#> 5 data/01-sales.csv January  2019     2  2156     9
#> 6 data/01-sales.csv January  2019     2  3987     6
#> # … with 13 more rows
```

- Com o parâmetro `id` nós definimos a coluna `files` para abrigar o nome do arquivo de origem de cada linha
- Isso pode ser útil para localizar uma observação na sua fonte original

---

- Se forem muitos arquivos você pode usar a função `list.files()` para gerar a lista dos nomes pra você a partir de algum padrão

. . .

```
sales_files <- list.files("data", pattern = "sales\\.csv$", full.names = TRUE)
sales_files
#> [1] "data/01-sales.csv" "data/02-sales.csv" "data/03-sales.csv"
```

- O capítulo 17 do livro e a documentação dessa função são ótimas referências caso você precise dela

8.5 Salvando em um arquivo

- readr oferece duas funções para salvar dados no disco:
  - `write_csv()` e `write_tsv()`
- Ambas utilizam os padrões UTF-8 para strings e ISO8601 para date-times
- Os principais argumentos são o dataframe a ser salvo e a localização do arquivo
- Você também pode especificar como escrever os `na`
- e acrescentar o dataset a um arquivo existente com `append`

- Vamos salvar o `student` e ler de volta:

. . .

```
students
#> # A tibble: 6 × 5
```

```
#>   student_id full_name        favourite_food     meal_plan             age
#>        <dbl> <chr>            <chr>              <fct>                 <dbl>
#> 1          1 Sunil Huffmann   Strawberry yoghurt Lunch only              4
#> 2          2 Barclay Lynn     French fries       Lunch only              5
#> 3          3 Jayendra Lyne    <NA>               Breakfast and lunch     7
#> 4          4 Leon Rossini     Anchovies          Lunch only             NA
#> 5          5 Chidiegwu Dunkel Pizza              Breakfast and lunch     5
#> 6          6 Güvenç Attila    Ice cream          Lunch only              6
write_csv(students, "students-2.csv")
read_csv("students-2.csv")
#> # A tibble: 6 × 5
#>   student_id full_name        favourite_food     meal_plan             age
#>        <dbl> <chr>            <chr>              <chr>                 <dbl>
#> 1          1 Sunil Huffmann   Strawberry yoghurt Lunch only              4
#> 2          2 Barclay Lynn     French fries       Lunch only              5
#> 3          3 Jayendra Lyne    <NA>               Breakfast and lunch     7
#> 4          4 Leon Rossini     Anchovies          Lunch only             NA
#> 5          5 Chidiegwu Dunkel Pizza              Breakfast and lunch     5
#> 6          6 Güvenç Attila    Ice cream          Lunch only              6
```

- Perceba que a informação de tipo se **perde** quando você salva em CSV
- É necessário **recriar** a especificação das colunas **sempre** que carregar um CSV
- Isso torna o CSV incerto e pouco confiável para armazenar resultados interinos
- Temos duas alternativas:
  - Salvar os dados em RDS, ou parquet

RDS

- Formato de binário nativo de R
- Não consegui o significado da sigla
  - O melhor chute que eu encontrei foi “R data serialized”

. . .

```
write_rds(students, "students.rds")
read_rds("students.rds")
#> # A tibble: 6 × 5
#>   student_id full_name        favourite_food     meal_plan             age
```

```
#>        <dbl> <chr>            <chr>              <fct>               <dbl>
#> 1          1 Sunil Huffmann   Strawberry yoghurt Lunch only              4
#> 2          2 Barclay Lynn     French fries       Lunch only              5
#> 3          3 Jayendra Lyne    <NA>               Breakfast and lunch     7
#> 4          4 Leon Rossini     Anchovies          Lunch only             NA
#> 5          5 Chidiegwu Dunkel Pizza              Breakfast and lunch     5
#> 6          6 Güvenç Attila    Ice cream          Lunch only              6
```

parquet

- parquet files é o que todas as pessoas legais usam
- um formato binário, autocontido que é absurdamente rápido e extremamente leve
- Desenvolvido pela Apache para big data, mas funciona incrível para armazenar qualquer tipo de dado tabular, e é compatível com virutalmente todas as linguagens de programação
- Para utilizar precisamos da biblioteca arrow:

. . .

```
install.packages("arrow")
library(arrow)
```

```
write_parquet(students, "students.parquet")
read_parquet("students.parquet")
#> # A tibble: 6 × 5
#>   student_id full_name        favourite_food     meal_plan             age
#>        <dbl> <chr>            <chr>              <fct>               <dbl>
#> 1          1 Sunil Huffmann   Strawberry yoghurt Lunch only              4
#> 2          2 Barclay Lynn     French fries       Lunch only              5
#> 3          3 Jayendra Lyne    NA                 Breakfast and lunch     7
#> 4          4 Leon Rossini     Anchovies          Lunch only             NA
#> 5          5 Chidiegwu Dunkel Pizza              Breakfast and lunch     5
#> 6          6 Güvenç Attila    Ice cream          Lunch only              6
```

```r
library(nycflights13)
library(tidyverse)
```

```
-- Attaching packages --------------------------------------- tidyverse 1.3.2 --
v ggplot2 3.4.0      v purrr   1.0.1
v tibble  3.1.8      v dplyr   1.0.10
v tidyr   1.3.0      v stringr 1.5.0
v readr   2.1.3      v forcats 0.5.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
```

```r
library(arrow)
```

```
Some features are not enabled in this build of Arrow. Run `arrow_info()` for more information

Attaching package: 'arrow'

The following object is masked from 'package:utils':

    timestamp
```

```r
#| echo: true
system.time(replicate(100, write_csv(flights,"flights.csv")))
```

```r
#| echo: true
system.time(replicate(100, write_rds(flights,"flights.rds")))
```

```r
#| echo: true
system.time(replicate(100, write_parquet(flights,"flights.parquet")))
```

8.6 Entrada de dados

- As vezes você precisa montar uma tabela "manualmente" no seu script
- Para isso temos duas funções que te permitem entrar dados por **linhas** ou por **colunas**

`tibble()` para escrever as colunas

- Nossa querida tabela neozelandesa, montada por **colunas**

. . .

```
tibble(
  x = c(1, 2, 5),
  y = c("h", "m", "g"),
  z = c(0.08, 0.83, 0.60)
)
#> # A tibble: 3 × 3
#>       x y         z
#>   <dbl> <chr> <dbl>
#> 1     1 h      0.08
#> 2     2 m      0.83
#> 3     5 g      0.6
```

- Sim, João, cada coluna precisa ter o mesmo número de linhas
- Nesse formato é mais difícil enxergar a relação entre linhas

---

`tribble()` para escrever as linhas

- **tr**ansposed **tibble**, estruturada por **linhas**
- Cada nome de coluna é precedido por um ~
- todas as entradas são separadas por vírgulas
- Permite uma visualização mais fácil de tabelas com poucos dados

. . .

```
tribble(
  ~x, ~y, ~z,
  "h", 1, 0.08,
  "m", 2, 0.83,
  "g", 5, 0.60,
)
#> # A tibble: 3 × 3
#>   x         y     z
#>   <chr> <dbl> <dbl>
#> 1 h         1  0.08
```

```
#> 2 m         2  0.83
#> 3 g         5  0.6
```

8.7 Sumário

- Neste capítulo você viu
  - como carregar arquivos CSV com `read_csv()`
  - como escrever seus dados com `tibble()` e `tribble()`
  - como arquivos CSV funcionam
  - alguns problemas comuns e como superá-los

- Agora você já sabe um tanto de R
  - Pelo menos, tem uma ideia que vai te permitir encontrar ajuda quando precisar
  - Esse é o mais importante

Perguntas?

Nos vemos no projeto!