

# Parte 4

2025-10-26

## Introdução ao Tidyverse

### Lidando com dados

#### Manipulação e visualização de dados

- Manipular e visualizar dados (MVD) são atividades obrigatórias em qualquer atividade científica.
- 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, portável, etc.

#### Principal Referência - R for Data Science

Workflow do Tidyverse

### O framework tidyverse

#### O tidyverse

- Oferece uma reimplementação e extensão das funcionalidades do R para manipulação e visualização de dados.
- É uma coleção de 8 pacotes que operam em harmonia.
- Foram planejados e construídos para trabalhar em conjunto.
- Possuem gramática, organização, lógica e estruturas de dados mais claras.
- Maior facilidade de desenvolvimento de código e portabilidade.
- Outros pacotes acoplam muito bem com o {tidyverse}.
- Pacotes:
  - R4DS:.
  - Cookbook:.

```

library(tidyverse)

## -- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## vforcats   1.0.0     v stringr   1.5.1
## v ggplot2   4.0.0     v tibble    3.3.0
## v lubridate 1.9.4     v tidyrr    1.3.1
## v purrr    1.1.0
## -- Conflicts ----- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()   masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors

```

```
tidyverse_packages()
```

```

## [1] "broom"          "conflicted"      "cli"           "dbplyr"
## [5] "dplyr"          "dtplyr"         "forcats"       "ggplot2"
## [9] "googledrive"    "googlesheets4"  "haven"        "hms"
## [13] "httr"           "jsonlite"       "lubridate"     "magrittr"
## [17] "modelr"         "pillar"         "purrr"        "ragg"
## [21] "readr"          "readxl"         "reprex"        "rlang"
## [25] "rstudioapi"    "rvest"          "stringr"      "tibble"
## [29] "tidyrr"         "xml2"          "tidyverse"

```

## Os pacotes do {tidyverse}

### Princípios dos dados organizados

- Cada variável está em uma coluna.
- Cada observação está em uma linha.
- Cada tipo de unidade observacional está em uma célula.

### Tarefas comuns ao lidar com dados

- Importação.
- Arrumação.
- Manipulação.
- Combinação.
- Exportação.

## Importação de dados

### O pacote {readr}

- O processo de análise de dados começa com a importação dos dados para o ambiente de manipulação.
- Existem vários meios para armazenar dados.
  - Arquivos de texto pleno (tsv, txt, csv, etc).
  - Planilhas eletrônicas.
  - Bancos de dados relacionais.

- Etc.
- O **readr** tem recursos para importação de dados retangulares na forma de texto pleno.
- **Documentação:**
  - **readr.**
  - **data import.**
  - **cran-r.**

## Importando arquivos de texto pleno

```
library(readr)
url <- "http://leg.ufpr.br/~wagner/scientificR/anovareg.txt"
dados <- read_tsv(url, col_names = TRUE)
```

## Importando dados do tipo .txt.

```
## # A tibble: 6 x 4
##   cultivar dose bloco indice
##   <chr>     <dbl> <chr>  <dbl>
## 1 Ag-1002    0 I      46
## 2 Ag-1002    0 II     48
## 3 Ag-1002    0 III    44
## 4 Ag-1002    0 IV     46
## 5 Ag-1002    60 I     48
## 6 Ag-1002    60 II    47
```

```
library(readr)
url <- "http://leg.ufpr.br/~wagner/scientificR/reglinear.csv"
dados<-read_table(url,col_names=TRUE)
```

## Importando dados do tipo .csv.

```
##
## -- Column specification -----
## cols(
##   "y" = col_double(),
##   "x" = col_double()
## )
```

```
head(dados)

## # A tibble: 6 x 2
##      "y"  "x"
##   <dbl> <dbl>
## 1 207318.    55
## 2 250846.    69
## 3 165755.    46
## 4 219817.    61
## 5 268582.    73
## 6 229060.    63
```

```
library(readxl)
library(httr)
url <- "http://leg.ufpr.br/~wagner/scientificR/meus_dados.xlsx"
GET(url, write_disk(tf <- tempfile(fileext = ".xlsx")))# primeiro faz um download temporário
```

### Importando uma planilha eletrônica

```
## Response [http://leg.ufpr.br/~wagner/scientificR/meus_dados.xlsx]
##   Date: 2025-11-06 18:32
##   Status: 200
##   Content-Type: application/vnd.openxmlformats-officedocument.spreadsheetml.sheet
##   Size: 10.7 kB
## <ON DISK> C:\Users\anton\AppData\Local\Temp\RtmpAxC6IS\file772c704558b.xlsx

tb <- read_excel(tf, sheet = "mtcars")
head(tb[,1:4])
```

```
## # A tibble: 6 x 4
##       mpg     cyl   disp     hp
##   <dbl> <dbl> <dbl> <dbl>
## 1    21       6   160    110
## 2    21       6   160    110
## 3  22.8      4   108     93
## 4  21.4      6   258    110
## 5  18.7      8   360    175
## 6  18.1      6   225    105
```

### Conexão com bancos de dados relacionais

```
library(DBI)
library(RMySQL)
# Criando a conexão.
db <- dbConnect(
  RMySQL::MySQL(),
```

```

user = "rfamro", password = "",
port = 4497, dbname = "Rfam",
host = "mysql-rfam-public.ebi.ac.uk")

# Lista as tabelas do BD.
dbListTables(db)

```

Conectando e importando tabelas de banco srelacionais - MySQL.

```

## [1] "_annotated_file"           "_family_file"
## [3] "_genome_data"              "_lock"
## [5] "_overlap"                  "_overlap_membership"
## [7] "_post_process"             "alignment_and_tree"
## [9] "author"                     "clan"
## [11] "clan_database_link"        "clan_literature_reference"
## [13] "clan_membership"           "database_link"
## [15] "db_version"                 "dead_clan"
## [17] "dead_family"                "ensembl_names"
## [19] "family"                     "family_author"
## [21] "family_literature_reference" "family_long"
## [23] "family_ncbi"                "features"
## [25] "full_region"                "genome"
## [27] "genome_temp"                 "genseq"
## [29] "genseq_temp"                 "html_alignment"
## [31] "keywords"                   "literature_reference"
## [33] "matches_and_fasta"          "motif"
## [35] "motif_database_link"        "motif_family_stats"
## [37] "motif_file"                  "motif_literature"
## [39] "motif_matches"               "motif_old"
## [41] "motif_pdb"                   "motif_ss_image"
## [43] "pdb"                         "pdb_full_region"
## [45] "pdb_full_region_old"         "pdb_rfam_reg"
## [47] "pdb_sequence"                 "processed_data"
## [49] "pseudoknot"                  "refseq"
## [51] "refseq_full_region"          "rfamseq"
## [53] "rfamseq_temp"                 "rnacentral_matches"
## [55] "rscape_annotations"          "secondary_structure_image"
## [57] "seed_region"                  "sunburst"
## [59] "taxonomic_tree"                "taxonomy"
## [61] "taxonomy_websearch"          "version"
## [63] "wikitext"

# Listas as colunas em uma tabela.
dbListFields(db, "keywords")

```

```

## [1] "rfam_acc"      "rfam_id"       "description"   "rfam_general" "literature"
## [6] "wiki"           "pdb_mappings"  "clan_info"

```

```

# Importando a tabela.
tb <- RMySQL::dbFetch(
  RMySQL::dbSendQuery(
    db, "SELECT * FROM keywords"))
str(tb)

```

```

## 'data.frame':   500 obs. of  8 variables:
## $ rfam_acc    : chr  "RF00001" "RF00002" "RF00003" "RF00004" ...
## $ rfam_id     : chr  "5S rRNA"  "5 8S rRNA" "U1"      "U2"      ...
## $ description : chr  "5S ribosomal RNA" "5 8S ribosomal RNA" "U1 spliceosomal RNA" "U2 spliceosomal ...
## $ rfam_general: chr  "Gene rRNA Griffiths Jones SR Mifsud Gardner PP Szymanski et al 5S ribosomal d ...
## $ literature   : chr  "Szymanski Barciszewska MZ Erdmann VA Barciszewski 5S Ribosomal RNA Database Y ...
## $ wiki         : chr  "5S ribosomal RNA 5S ribosomal RNA Predicted secondary structure and sequence c ...
## $ pdb_mappings: chr  "1qvg 4u4q 4v9i 5fdv 6r6p 7p7t 8cku 8g38 8urh 4csu 4u27 4v5q 4w2i 5gak 5v7q 6s ...
## $ clan_info    : chr  "CL00113 5S rRNA 5S rRNA clan Includes 5S rRNA family with permuted secondary s ...

# Desconecta
dbDisconnect(db)

## Warning: Closing open result sets

## [1] TRUE

```

## Consumindo dados de APIs

- API (Application Programming Interface) é um conjunto de rotinas que permitem diferentes softwares interagirem.
- Pense como uma espécie de ponte ou garçom entre sistemas/softwares.
- Por meio de uma API você pode usar modelos pré-prontos de diversos fornecedores.
- Acessar dados de fornecedores especializados.
- Uma API é o que vai permitir criar novas soluções usando dados/modelos existentes.

### Como funciona?

- **Requisição:** Você envia um ‘pedido’ para a API.
- **Processamento:** O serviço (dados/modelos, etc) processa o pedido e gera uma resposta.
- **Resposta:** A API devolve a resposta para você.

Fluxo de chamada de API.

## A importância de APIs para Cientistas de Dados

- APIs são portas de entrada para dados e serviços.
- Permitem: automação, integração e acesso em tempo real a informações externas.
- Entender como acessar e construir APIs vai permitir ir além do uso de planilhas e banco de dados.

## Casos práticos em Ciência de Dados

- Coleta de dados em larga escala (ex: dados climáticos, financeiros, sociais).
- Integração com modelos de IA, como o ChatGPT ou modelos de predição customizados.
- Dashboards dinâmicos com dados atualizados via APIs REST.
- Automatização de processos de negócio: envio de relatórios, alertas, atualizações.

## Pacote httr

- httr é um pacote para facilitar o consumo de APIs.
- Ele abstrai os detalhes do protocolo HTTP, permitindo chamadas simples e legíveis.
- Ideal para requisições GET, POST, PUT, DELETE, com suporte a autenticação e manipulação de headers.
- Instalação direto do CRAN

```
## Instalação do pacote
install.packages("httr")
```

```
## Warning: o pacote 'httr' está em uso e não será instalado
```

## Principais funções

	Função	Descrição
GET()	Realiza uma requisição HTTP GET	
POST()	Envia dados para a API via HTTP POST	
PUT()	Atualiza recursos	
DELETE()	Remove recursos	
add_headers()	Adiciona cabeçalhos HTTP, como tokens	
content()	Extrai e interpreta o conteúdo da resposta	

## Procurar por documentação oficial

**Exemplo de requisição httr** Vamos acessar a API do github.

```
library(httr) ## Carregando o pacote
res <- GET("https://api.github.com/users/hadley") ## Requisição
user_info <- content(res, as = "parsed") ## Capturando o conteúdo
user_info$login ## Verificando a informação recebida

## [1] "hadley"
```

## Acessando APIs; Pacote httr

**Exemplos** Exemplo de requisição do tipo GET

```
GET("https://api.exemplo.com/dados")
```

É comum precisar mandarmos dados para a API. Exemplo de requisição do tipo POST

```
POST("https://api.exemplo.com/envio", body = list(nome = "Ana", idade = 30))
```

Para capturar o retorno da API usamos a função content()

```
res <- GET("https://api.exemplo.com/dados") status_code(res) dados <- content(res, as = "parsed")
```

Importante saber lidar com erros da API.

```
if (status_code(res) == 200) { content(res, as = "parsed") } else { message("Erro:", status_code(res)) }
```

## Dicas e boas práticas

- Sempre leia a documentação da API
- Use tryCatch() para capturar falhas inesperadas
- Armazene chaves em arquivos .env (nunca no código!)
- Script de exemplos práticos: acesso\_apis.R.

imagem sobre APIs

## Introdução ao dplyr

O dplyr é a gramática para manipulação de dados. - Apresenta um conjunto consistente de verbos para atuar sobre tabelas.

- Verbos principais: arrange(), select(), mutate(), slice, rename(), filter(), summarise(), etc.
- Sufixos: \_at() , \_if() , \_all() , etc.
- Agrupamento: group\_by() e ungroup() .
- Junções: inner\_join(), full\_join(), left\_join(), right\_join().
- Funções resumo: n(), n\_distinct(), first(), last(), nth(), etc.

## Referências úteis

Cheat sheet Documentação oficial: manual Capítulo do livro help do Cran

## Setup pipe

### Carregamento dos pacotes necessários

```
require(tidyverse)

dados <- readr::read_csv("C:/Users/anton/OneDrive/Área de Trabalho/CE302---2025.2/Parte 4/Mental Health.csv")

## Rows: 292364 Columns: 17
## -- Column specification -----
## Delimiter: ","
## chr (17): Timestamp, Gender, Country, Occupation, self-employed, family_hist...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.

## Para vermos os dados, podemos utilizar a função head()
head(dados, 2)
```

```

## # A tibble: 2 x 17
##   Timestamp    Gender Country Occupation self_employed family_history treatment
##   <chr>        <chr>  <chr>    <chr>        <chr>        <chr>
## 1 8/27/2014 11~ Female United~ Corporate <NA>          No         Yes
## 2 8/27/2014 11~ Female United~ Corporate <NA>          Yes        Yes
## # i 10 more variables: Days_Outdoors <chr>, Growing_Stress <chr>,
## #   Changes_Habits <chr>, Mental_Health_History <chr>, Mood_Swings <chr>,
## #   Coping_Struggles <chr>, Work_Interest <chr>, Social_Weakness <chr>,
## #   mental_health_interview <chr>, care_options <chr>

```

## Operador Pipe

- O operador `%>%` é chamado de *pipe* e é utilizado para encadear funções.
- Utilizado para facilitar a leitura e escrita de códigos.
  - $x \%>% f$  é equivalente à  $f(x)$
  - $x \%>% f(y)$  é equivalente à  $f(x, y)$
  - $x \%>% f \%>% g \%>% h$  é equivalente à  $h(g(f(x)))$
- O `%>%` significa que o elemento à esquerda será avaliado pela função à direita.
- Podemos também utilizar o `.` como espaço reservado para o elemento à esquerda, isto é:
  - $x \%>% f(y, .)$  é equivalente à  $f(y, x)$
  - $x \%>% f(., y)$  é equivalente à  $f(x, y)$
  - $x \%>% f(y, z = .)$  é equivalente à  $f(y, z = x)$  .

**Exemplo do uso do operador `%>%`** Suponha que queremos calcular o cosseno dos valores únicos de um vetor  $x$ , ordená-los em ordem decrescente.

- 1) Sem o uso do operador `%>%` : Sem identação

```
x <- c(-2:2)
x
```

```
## [1] -2 -1  0  1  2
```

```
# Opção 1 - Sem identação
sort(cos(unique(x)), decreasing = TRUE)
```

```
## [1] 1.0000000 0.5403023 0.5403023 -0.4161468 -0.4161468
```

- 2) Sem o uso do operador `%>%` : Com identação

```
# Opção 2 - Com identação
sort(
  cos(
    unique(
      x
    )
  ),
  decreasing = TRUE)
```

```
## [1] 1.0000000 0.5403023 0.5403023 -0.4161468 -0.4161468
```

3) Exemplo do uso do operador %>%

```
# Opção 3 - Utilizando o operador %>%
require(magrittr)
```

```
## Carregando pacotes exigidos: magrittr
```

```
##
```

```
## Anexando pacote: 'magrittr'
```

```
## O seguinte objeto é mascarado por 'package:purrr':
```

```
##
```

```
##     set_names
```

```
## O seguinte objeto é mascarado por 'package:tidy়':
```

```
##
```

```
##     extract
```

```
x %>%
  unique() %>%
  cos() %>%
  sort(decreasing = TRUE)
```

```
## [1] 1.0000000 0.5403023 0.5403023 -0.4161468 -0.4161468
```

### Pipe de Atribuição %<>%

- O operador %<>% é utilizado para atribuir o resultado de uma operação ao objeto original.
- Isto é, x %<>% f() é equivalente à x <- f(x) .

```
x <- 1:10
x
```

```
## [1] 1 2 3 4 5 6 7 8 9 10
```

```
x %<>% log()
x
```

```
## [1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379 1.7917595 1.9459101
## [8] 2.0794415 2.1972246 2.3025851
```

## Mutate

- Criando Variáveis com mutate()
- Para criar novas variáveis, podemos utilizar a função mutate() .
- Utilizando o banco de dados dados , vamos criar uma nova variável chamada mercosul que indica se o país é membro do Mercosul ou não.

```

## Criar Colunas com Mutate
dados <- dados %>%
  mutate(mercosul = ifelse(Country %in%
                           c("Argentina", "Brazil", "Paraguay", "Uruguay"),
                           "Mercosul", "Não Mercosul"))
glimpse(dados)

## Rows: 292,364
## Columns: 18
## $ Timestamp
## $ Gender
## $ Country
## $ Occupation
## $ self_employed
## $ family_history
## $ treatment
## $ Days_Indoors
## $ Growing_Stress
## $ Changes_Habits
## $ Mental_Health_History
## $ Mood_Swings
## $ Coping_Struggles
## $ Work_Interest
## $ Social_Weakness
## $ mental_health_interview
## $ care_options
## $ mercosul

```

<chr> "8/27/2014 11:29", "8/27/2014 11:31", "8/27/20~  
<chr> "Female", "Female", "Female", "Female", "Femal~  
<chr> "United States", "United States", "United Stat~  
<chr> "Corporate", "Corporate", "Corporate", "Corpor~  
<chr> NA, NA, NA, "No", "No", "No", "No", "No", "No"~  
<chr> "No", "Yes", "Yes", "Yes", "Yes", "No", "Yes", ~  
<chr> "Yes", "Yes", "Yes", "Yes", "Yes", "Yes", "Yes~  
<chr> "1-14 days", "1-14 days", "1-14 days", "1-14 d~  
<chr> "Yes", "Yes", "Yes", "Yes", "Yes", "Yes", "Yes~  
<chr> "No", "No", "No", "No", "No", "No", "No", "No"~  
<chr> "Yes", "Yes", "Yes", "Yes", "Yes", "Yes", "Yes~  
<chr> "Medium", "Medium", "Medium", "Medium", "Mediu~  
<chr> "No", "No", "No", "No", "No", "No", "No"~  
<chr> "No", "No", "No", "No", "No", "No", "No"~  
<chr> "Yes", "Yes", "Yes", "Yes", "Yes", "Yes", "Yes~  
<chr> "No", "No", "No", "Maybe", "No", "Maybe", "No"~  
<chr> "Not sure", "No", "Yes", "Yes", "Yes", "Not su~  
<chr> "Não Mercosul", "Não Mercosul", "Não Mercosul"~

## Selecionando e Removendo Variáveis

### Selecionando Variáveis com select()

- Para selecionar variáveis, podemos utilizar a função select() .
- Seleção por nomes: Vamos selecionar as variáveis Country, Timestamp, Days\_Indoors e mercosul.

```

## Selecionar colunas com select()
dados2 <- dados %>%
  select(Country, Timestamp, Days_Indoors, mercosul) # selecionando variáveis nominalmente
glimpse(dados2)

```

```

## Rows: 292,364
## Columns: 4
## $ Country      <chr> "United States", "United States", "United States", "Unite~  

## $ Timestamp    <chr> "8/27/2014 11:29", "8/27/2014 11:31", "8/27/2014 11:32", ~  

## $ Days_Indoors <chr> "1-14 days", "1-14 days", "1-14 days", "1-14 days", "1-14~  

## $ mercosul     <chr> "Não Mercosul", "Não Mercosul", "Não Mercosul", "Não Merc~
```

### Incrementando o select()

#### Seleção por índices

- Podemos utilizar o índice das variáveis para selecioná-las.

## Seleção Intervalar

- Podemos utilizar o operador: para selecionar um intervalo de variáveis.
  - Vamos selecionar as variáveis treatment até Changes\_Habits.

### Selecionando várias colunas por padrão

- Podemos utilizar a função starts\_with(), ends\_with(), contains() e matches() para selecionar variáveis que atendam a um padrão.
  - Vamos selecionar as variáveis que começam com a letra t.

- Vamos selecionar as variáveis que terminam com a letra s.

```
dados7 <- dados %>%
  select(ends_with("s"))
glimpse(dados7)
```

- Vamos selecionar as variáveis que contém as letra ing.

```
dados8 <- dados %>%
  select(contains("ing"))
glimpse(dados8)
```

```
## Rows: 292,364
## Columns: 3
## $ Growing_Stress    <chr> "Yes", "Yes", "Yes", "Yes", "Yes", "Yes", "Yes", "Yes~
## $ Mood_Swings        <chr> "Medium", "Medium", "Medium", "Medium", "Medium", "Me~
## $ Coping_Struggles  <chr> "No", "No", "No", "No", "No", "No", "No", "No", "No", ~
```

- Vamos selecionar as variáveis que contém a letra t ou T.

```
dados9 <- dados %>%
  select(matches("[tT]"))
glimpse(dados9)
```

## Removendo Variáveis com select()

- Para remover variáveis, podemos utilizar a função select() com o operador - .
  - Vamos remover as variáveis Country, Timestamp, Days Indoors e mercosul.

```
dados10 <- dados %>%
  select(-Country, -Timestamp, -Days_Indoors, -mercossul)
glimpse(dados10)
```

```

## Rows: 292,364
## Columns: 14
## $ Gender <chr> "Female", "Female", "Female", "Female", "Femal~
## $ Occupation <chr> "Corporate", "Corporate", "Corporate", "Corpor~
## $ self-employed <chr> NA, NA, NA, "No", "No", "No", "No", "No"~
## $ family_history <chr> "No", "Yes", "Yes", "Yes", "Yes", "No", "Yes", ~
## $ treatment <chr> "Yes", "Yes", "Yes", "Yes", "Yes", "Yes", "Yes~
## $ Growing_Stress <chr> "Yes", "Yes", "Yes", "Yes", "Yes", "Yes", "Yes~
## $ Changes_Habits <chr> "No", "No", "No", "No", "No", "No", "No"~
## $ Mental_Health_History <chr> "Yes", "Yes", "Yes", "Yes", "Yes", "Yes", "Yes~
## $ Mood_Swings <chr> "Medium", "Medium", "Medium", "Medium", "Mediu~
## $ Coping_Struggles <chr> "No", "No", "No", "No", "No", "No", "No"~
## $ Work_Interest <chr> "No", "No", "No", "No", "No", "No", "No"~
## $ Social_Weakness <chr> "Yes", "Yes", "Yes", "Yes", "Yes", "Yes", "Yes~
## $ mental_health_interview <chr> "No", "No", "No", "Maybe", "No", "Maybe", "No"~
## $ care_options <chr> "Not sure", "No", "Yes", "Yes", "Yes", "Not su~

```

```

dados11 <- dados %>%
  select(-c(Country, Timestamp, Days_Indoors, mercosul))
glimpse(dados11)

```

```

## Rows: 292,364
## Columns: 14
## $ Gender <chr> "Female", "Female", "Female", "Female", "Femal~
## $ Occupation <chr> "Corporate", "Corporate", "Corporate", "Corpor~
## $ self-employed <chr> NA, NA, NA, "No", "No", "No", "No", "No"~
## $ family_history <chr> "No", "Yes", "Yes", "Yes", "Yes", "No", "Yes", ~
## $ treatment <chr> "Yes", "Yes", "Yes", "Yes", "Yes", "Yes", "Yes~
## $ Growing_Stress <chr> "Yes", "Yes", "Yes", "Yes", "Yes", "Yes", "Yes~
## $ Changes_Habits <chr> "No", "No", "No", "No", "No", "No", "No"~
## $ Mental_Health_History <chr> "Yes", "Yes", "Yes", "Yes", "Yes", "Yes", "Yes~
## $ Mood_Swings <chr> "Medium", "Medium", "Medium", "Medium", "Mediu~
## $ Coping_Struggles <chr> "No", "No", "No", "No", "No", "No", "No"~
## $ Work_Interest <chr> "No", "No", "No", "No", "No", "No", "No"~
## $ Social_Weakness <chr> "Yes", "Yes", "Yes", "Yes", "Yes", "Yes", "Yes~
## $ mental_health_interview <chr> "No", "No", "No", "Maybe", "No", "Maybe", "No"~
## $ care_options <chr> "Not sure", "No", "Yes", "Yes", "Yes", "Not su~

```

## Seleção de variáveis por Tipos específicos de dados

- Podemos utilizar a função `select_if()` para selecionar variáveis que atendam a um critério específico.
- Vamos selecionar as variáveis que são do tipo character.

```

dados12 <- dados %>%
  select_if(is.character)
glimpse(dados12)

```

```

## Rows: 292,364
## Columns: 18
## $ Timestamp <chr> "8/27/2014 11:29", "8/27/2014 11:31", "8/27/20~
## $ Gender <chr> "Female", "Female", "Female", "Female", "Femal~
## $ Country <chr> "United States", "United States", "United Stat~

```

```

## $ Occupation <chr> "Corporate", "Corporate", "Corporate", "Corpor~  

## $ self_employed <chr> NA, NA, NA, "No", "No", "No", "No", "No"~  

## $ family_history <chr> "No", "Yes", "Yes", "Yes", "Yes", "No", "Yes",~  

## $ treatment <chr> "Yes", "Yes", "Yes", "Yes", "Yes", "Yes", "Yes",~  

## $ Days_Indoors <chr> "1-14 days", "1-14 days", "1-14 days", "1-14 d~  

## $ Growing_Stress <chr> "Yes", "Yes", "Yes", "Yes", "Yes", "Yes", "Yes~  

## $ Changes_Habits <chr> "No", "No", "No", "No", "No", "No", "No", "No"~  

## $ Mental_Health_History <chr> "Yes", "Yes", "Yes", "Yes", "Yes", "Yes", "Yes~  

## $ Mood_Swings <chr> "Medium", "Medium", "Medium", "Medium", "Mediu~  

## $ Coping_Struggles <chr> "No", "No", "No", "No", "No", "No", "No"~  

## $ Work_Interest <chr> "No", "No", "No", "No", "No", "No", "No", "No"~  

## $ Social_Weakness <chr> "Yes", "Yes", "Yes", "Yes", "Yes", "Yes", "Yes~  

## $ mental_health_interview <chr> "No", "No", "No", "Maybe", "No", "Maybe", "No"~  

## $ care_options <chr> "Not sure", "No", "Yes", "Yes", "Yes", "Not su~  

## $ mercosul <chr> "Não Mercosul", "Não Mercosul", "Não Mercosul"~

```

## Seleção por critérios

- Podemos definir externamente um critério para selecionar variáveis. Por exemplo, temos o nome de algumas variáveis que queremos selecionar.
- all\_of() é utilizado para selecionar variáveis que atendam a um critério externo.
- any\_of() é utilizado para selecionar variáveis que atendam a pelo menos um critério externo.

```

dados %>%
  select_if(is.logical) #nenhuma

## # A tibble: 292,364 x 0

dados %>%
  select_if(is.numeric) #nenhuma

## # A tibble: 292,364 x 0

variaveis <- c("Country", "Timestamp", "Days_Indoors", "mercosul")
dados13 <- dados %>%
  select(all_of(variaveis))
glimpse(dados13)

## Rows: 292,364
## Columns: 4
## $ Country      <chr> "United States", "United States", "United States", "Unite~  

## $ Timestamp    <chr> "8/27/2014 11:29", "8/27/2014 11:31", "8/27/2014 11:32", ~  

## $ Days_Indoors <chr> "1-14 days", "1-14 days", "1-14 days", "1-14 days", "1-14~  

## $ mercosul     <chr> "Não Mercosul", "Não Mercosul", "Não Mercosul", "Não Merc~

variaveis <- c("Country", "Timestamp", "Days_Indoors", "mercosul", "Loss_of_Smell")
dados14 <- dados %>%
  select(any_of(variaveis))
glimpse(dados14)

```

```

## Rows: 292,364
## Columns: 4
## $ Country      <chr> "United States", "United States", "United States", "Unite~
## $ Timestamp    <chr> "8/27/2014 11:29", "8/27/2014 11:31", "8/27/2014 11:32", ~
## $ Days_Indoors <chr> "1-14 days", "1-14 days", "1-14 days", "1-14 days", "1-14~
## $ mercosul     <chr> "Não Mercosul", "Não Mercosul", "Não Mercosul", "Não Merc~

```

## Filtrando observações

Para filtrar observações, podemos utilizar a função filter()

- ==: Igual a
- !=: Diferente de
- <: Menor que
- : Maior que
- <=: Menor ou igual a
- =: Maior ou igual a

Vamos filtrar as observações pertencentes ao mercosul.

Para aprendermos filtros, vamos usar o banco de dados *car\_crash*.

```

library(data.table)

## 
## Anexando pacote: 'data.table'

## Os seguintes objetos são mascarados por 'package:lubridate':
## 
##   hour, isoweek, mday, minute, month, quarter, second, wday, week,
##   yday, year

## Os seguintes objetos são mascarados por 'package:dplyr':
## 
##   between, first, last

## O seguinte objeto é mascarado por 'package:purrr':
## 
##   transpose

car_crash <- fread("C:/Users/anton/OneDrive/Área de Trabalho/CE302---2025.2/Parte 4/Brazil Total highway crashes.csv")

library(magrittr)
library(tidyverse)

glimpse(car_crash)

```

```

## Rows: 864,561
## Columns: 24

## $ data
## $ horario
## $ n_da_ocorrencia
## $ tipo_de_ocorrencia
## $ km
## $ trecho
## $ sentido
## $ lugar_acidente
## $ tipo_de_acidente
## $ automovel
## $ bicicleta
## $ caminhao
## $ moto
## $ onibus
## $ outros
## $ tracao_animal
## $ transporte_de_cargas_especiais
## $ trator_maquinas
## $ utilitarios
## $ ilesos
## $ levemente_feridos
## $ moderadamente_feridos
## $ gravemente_feridos
## $ mortos

<chr> "01/01/2010", "01/01/2010", "01/01/2010~
<chr> "04:21:00", "02:13:00", "03:35:00", "07~
<chr> "18", "20", "000024/2010", "000038/2010~
<chr> "sem vítima", "sem vítima", "sem vítima~
<chr> "167", "269,5", "77", "52", "33", "24", ~
<chr> "BR-393/RJ", "BR-116/PR", "BR-290/RS", ~
<chr> "Norte", "Sul", "Norte", "Norte", "Nort~
<chr> "Rodovia do Aço", "Autopista Regis Bitt~
<chr> "Derrapagem", "Colisão Traseira", "COLI~
<dbl> 1, 2, 2, 0, 0, 1, 1, 1, 2, 1, NA, 1, 1, ~
<dbl> NA, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
<dbl> NA, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
<dbl> NA, NA, 0, 1, 1, 0, NA, NA, NA, NA, 1, ~
<dbl> NA, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
<dbl> NA, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
<dbl> NA, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
<dbl> NA, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
<dbl> NA, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
<dbl> NA, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
<dbl> NA, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
<dbl> 1, 3, 2, 1, 1, 1, 3, 4, 4, 1, 0, 1, 1, ~
<dbl> 0, NA, 0, 0, 0, 0, NA, NA, 5, NA, 2, NA~
<dbl> 0, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
<dbl> 0, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
<dbl> 0, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~

```

## Filtro Simples

Vamos filtrar as observações cujo tipo de ocorrência é *sem vítima*

```
# Filtrando linhas com filter()
car_crash2 <- car_crash %>%
  filter(tipo_de_ocorrencia == "sem vítima")
glimpse(car_crash2)
```

```

## Rows: 411,519
## Columns: 24
## $ data
## $ horario
## $ n_da_ocorrencia
## $ tipo_de_ocorrencia
## $ km
## $ trecho
## $ sentido
## $ lugar_acidente
## $ tipo_de_acidente
## $ automovel
## $ bicicleta
## $ caminhao
## $ moto
## $ onibus
## $ outros

```

```

## $ tracao_animal <dbl> NA, NA, 0, 0, 0, 0, NA, NA, NA, NA, 0, ~
## $ transporte_de_cargas_especiais <dbl> NA, NA, 0, 0, 0, 0, NA, NA, NA, NA, ~
## $ trator_maquinas <dbl> NA, NA, 0, 0, 0, 0, NA, NA, NA, NA, 0, ~
## $ utilitarios <dbl> NA, NA, 0, 0, 0, 0, NA, NA, NA, NA, 0, ~
## $ ilesos <dbl> 1, 3, 2, 1, 1, 3, 4, 1, 1, 2, 2, ~
## $ levemente_feridos <dbl> 0, NA, 0, 0, 0, 0, NA, NA, NA, 0, N~
## $ moderadamente_feridos <dbl> 0, NA, 0, 0, 0, 0, NA, NA, NA, 0, N~
## $ gravemente_feridos <dbl> 0, NA, 0, 0, 0, 0, NA, NA, NA, 0, N~
## $ mortos <dbl> 0, NA, 0, 0, 0, 0, NA, NA, NA, 0, N~

```

## Filtros Combinados

- Podemos combinar filtros utilizando os operadores lógicos & (E) e | (OU).
- Vamos filtrar as observações cujo tipo de ocorrência é sem vítima envolvendo pelo menos 3 automóveis.

```

## Filtros com múltiplas condições
car_crash3 <- car_crash %>%
  filter(tipo_de_ocorrencia == "sem vítima" & automovel >= 3)
glimpse(car_crash3)

```

```

## Rows: 19,431
## Columns: 24
## $ data <chr> "01/01/2010", "01/01/2011", "01/01/2011~"
## $ horario <chr> "13:14:00", "23:21:00", "12:21:00", "13~"
## $ n_da_ocorrencia <chr> "150", "542", "212", "135", "309", "145~"
## $ tipo_de_ocorrencia <chr> "sem vítima", "sem vítima", "sem vítima~"
## $ km <chr> "560", "137,5", "68,8", "269", "193", "~"
## $ trecho <chr> "BR-116/PR", "BR-101/SC", "BR-116/SP", ~
## $ sentido <chr> "Sul", "Norte", "Pista Sul", "Norte", "~"
## $ lugar_acidente <chr> "Autopista Regis Bittencourt", "Autopis~"
## $ tipo_de_acidente <chr> "Colisão Traseira", "Colisão Traseira", ~
## $ automovel <dbl> 3, 3, 3, 4, 3, 3, 3, 3, 6, 3, 3, 3, ~
## $ bicicleta <dbl> NA, NA, 0, NA, NA, NA, NA, NA, NA, ~
## $ caminhao <dbl> NA, NA, 0, NA, NA, NA, NA, NA, NA, ~
## $ moto <dbl> NA, NA, 0, NA, NA, NA, NA, NA, NA, ~
## $ onibus <dbl> NA, NA, 0, NA, NA, NA, NA, NA, NA, ~
## $ outros <dbl> NA, NA, 0, NA, NA, NA, NA, NA, NA, ~
## $ tracao_animal <dbl> NA, NA, 0, NA, NA, NA, NA, NA, NA, ~
## $ transporte_de_cargas_especiais <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, ~
## $ trator_maquinas <dbl> NA, NA, 0, NA, NA, NA, NA, NA, NA, ~
## $ utilitarios <dbl> NA, NA, 0, NA, NA, NA, NA, NA, NA, ~
## $ ilesos <dbl> 14, 3, 11, 7, 3, 3, 3, 8, 6, 3, 3, 3~
## $ levemente_feridos <dbl> NA, NA, 0, NA, NA, NA, NA, NA, NA, ~
## $ moderadamente_feridos <dbl> NA, NA, 0, NA, NA, NA, NA, NA, NA, ~
## $ gravemente_feridos <dbl> NA, NA, 0, NA, NA, NA, NA, NA, NA, ~
## $ mortos <dbl> NA, NA, 0, NA, NA, NA, NA, NA, NA, ~

```

## Filtrando valores em um intervalo

- Podemos filtrar **valores** em um intervalo numérico utilizando a função *between*.
- Vamos filtrar as observações cujo número de automóveis envolvidos está entre 3 e 5

```
car_crash4 <- car_crash %>%
  filter(between(automovel, 3, 5))
```

- Podemos filtrar strings utilizando o operator `%in%`.
- Vamos filtrar as observações cujo tipo de ocorrência é *sem vítima* ou com *vítima*.

```
tipos <- c("sem vítima", "com vítima")
tipos
```

```
## [1] "sem vítima" "com vítima"
```

```
car_crash5 <- car_crash %>%
  filter(tipo_de_ocorrencia %in% tipos)
car_crash5
```

```
##           data horario n_da_ocorrencia tipo_de_ocorrencia      km      trecho
##           <char>    <char>          <char>          <char> <char>    <char>
## 1: 01/01/2010 04:21:00                      18    sem vítima   167 BR-393/RJ
## 2: 01/01/2010 02:13:00                      20    sem vítima  269,5 BR-116/PR
## 3: 01/01/2010 03:35:00 000024/2010    sem vítima   77 BR-290/RS
## 4: 01/01/2010 07:31:00 000038/2010    sem vítima   52 BR-116/RS
## 5: 01/01/2010 04:57:00 000027/2010    sem vítima   33 BR-290/RS
## ---
## 608681: 31/12/2019 18:00:00                  239    sem vítima   56,3 BR-381/SP
## 608682: 31/12/2019 19:50:00                  265  com vítima  917,07 BR-381/MG
## 608683: 31/12/2019 21:11:00                  273  com vítima  477,2 BR-381/MG
## 608684: 31/12/2019 21:59:00                  283    sem vítima  923 BR-381/MG
## 608685: 31/12/2019 12:56:26                  150  com vítima   98 BR-290/RS
##           sentido          lugar_acidente tipo_de_acidente automovel
##           <char>          <char>          <char>        <num>
## 1:    Norte          Rodovia do Aço     Derrapagem       1
## 2:     Sul Autopista Regis Bittencourt Colisão Traseira     2
## 3:    Norte          Concepa COLISÃO LATERAL     2
## 4:    Norte          Concepa QUEDA DE MOTO      0
## 5:    Norte          Concepa QUEDA DE MOTO      0
## ---
## 608681:     Sul      Autopista Fernão Dias Engavetamento       1
## 608682:     Sul      Autopista Fernão Dias Colisão Frontal     1
## 608683:    Norte      Autopista Fernão Dias      Tombamento    NA
## 608684:    Norte      Autopista Fernão Dias Colisão Lateral     1
## 608685:    Oeste          Via Sul Queda de moto      0
##           bicicleta caminhao  moto onibus outros tracao_animal
##           <num>    <num> <num> <num> <num>          <num>
## 1:      NA      NA    NA    NA    NA          NA
## 2:      NA      NA    NA    NA    NA          NA
## 3:      0       0    0     0     0          0
## 4:      0       0    1     0     0          0
## 5:      0       0    1     0     0          0
## ---
## 608681:      NA      NA    NA    NA     1          NA
## 608682:      NA      NA    NA    NA     NA          NA
## 608683:      NA      NA     1    NA    NA          NA
```

```

## 608684:      NA      NA      1      NA      NA      NA
## 608685:      0       0       1       0       0       0
##          transporte_de_cargas_especiais trator_maquinas utilitarios ilesos
##                               <num>           <num>           <num>   <num>
##     1:             NA             NA             NA       1
##     2:             NA             NA             NA       3
##     3:             0              0              0       2
##     4:             0              0              0       1
##     5:             0              0              0       1
##    ---
## 608681:             NA             NA             1       3
## 608682:             NA             NA             1       0
## 608683:             NA             NA             NA       0
## 608684:             NA             NA             NA       2
## 608685:             0              0              0       0
##          levemente_feridos moderadamente_feridos gravemente_feridos mortos
##                               <num>           <num>           <num>   <num>
##     1:             0              0              0       0
##     2:             NA             NA             NA       NA
##     3:             0              0              0       0
##     4:             0              0              0       0
##     5:             0              0              0       0
##    ---
## 608681:             NA             NA             NA       NA
## 608682:             1              1             NA       NA
## 608683:             1              NA             NA       NA
## 608684:             NA             NA             NA       NA
## 608685:             1              0              0       0

```

- Para filtrarmos o contrário, podemos utilizar o operador `!`, ou definirmos um operador `not in` como `%ni% <- Negate(%in%).`

```

car_crash6 <- car_crash %>%
  filter(!tipo_de_ocorrencia %in% c("sem vítima", "com vítima"))
glimpse(car_crash6)

```

```

## Rows: 255,876
## Columns: 24
## $ data
## $ horario
## $ n_da_ocorrencia
## $ tipo_de_ocorrencia
## $ km
## $ trecho
## $ sentido
## $ lugar_acidente
## $ tipo_de_acidente
## $ automovel
## $ bicicleta
## $ caminhao
## $ moto
## $ onibus
## $ outros
<chr> "01/01/2014", "01/01/2014", "01/01/2015~"
<chr> "18:31:00", "11:28:00", "19:10:00", "12~"
<chr> "182", "97", "215", "153", "10", "288",~
<chr> "Acidente com vítima", "Acidente com ví~"
<chr> "354,2", "308", "77", "220,5", "266", "~"
<chr> "BR-101/ES", "BR-101/ES", "BR-060/GO", ~
<chr> "Norte", "Sul", "Sul", "Sul", "Norte", ~
<chr> "EC0101", "EC0101", "Concebra", "EC0101~"
<chr> "Queda de Moto", "Engavetamento", "Capo~"
<dbl> 0, 3, 1, 1, 0, 1, 1, 1, 1, 1, 1, ~
<dbl> 0, 0, NA, 0, 0, 0, 0, 0, 1, 0, 0, 0, ~
<dbl> 0, 0, NA, 0, 1, 0, 0, 0, 0, 0, 0, 0, ~
<dbl> 1, 0, NA, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, ~
<dbl> 0, 0, NA, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
<dbl> 0, 0, NA, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~

```

```

## $ tracao_animal <dbl> NA, NA, NA, NA, 0, NA, 0, NA, 0, NA~  

## $ transporte_de_cargas_especiais <dbl> NA, NA, NA, NA, 0, NA, 0, NA~  

## $ trator_maquinas <dbl> NA, NA, NA, NA, 0, NA, 0, NA, 0, NA~  

## $ utilitarios <dbl> NA, NA, NA, NA, 0, NA, 0, NA, 0, NA~  

## $ ilesos <dbl> 0, 0, 0, 1, 1, 0, 0, 2, 0, 1, 3, 5, 0, ~  

## $ levemente_feridos <dbl> 1, 1, NA, 2, 0, 0, 0, 0, 1, 0, 0, 0, 0, ~  

## $ moderadamente_feridos <dbl> 0, 0, 3, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, ~  

## $ gravemente_feridos <dbl> 0, 0, NA, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, ~  

## $ mortos <dbl> 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~

## Usando o operador %ni%
`%ni%` <- Negate(`%in%`)
car_crash7 <- car_crash %>%
  filter(tipo_de_ocorrencia %ni% c("sem vítima", "com vítima"))
glimpse(car_crash7)

## Rows: 255,876
## Columns: 24
## $ data <chr> "01/01/2014", "01/01/2014", "01/01/2015~  

## $ horario <chr> "18:31:00", "11:28:00", "19:10:00", "12~  

## $ n_da_ocorrencia <chr> "182", "97", "215", "153", "10", "288",~  

## $ tipo_de_ocorrencia <chr> "Acidente com vítima", "Acidente com ví~  

## $ km <chr> "354,2", "308", "77", "220,5", "266", "~  

## $ trecho <chr> "BR-101/ES", "BR-101/ES", "BR-060/GO", ~  

## $ sentido <chr> "Norte", "Sul", "Sul", "Sul", "Norte", ~  

## $ lugar_acidente <chr> "ECO101", "ECO101", "Concebra", "ECO101~  

## $ tipo_de_acidente <chr> "Queda de Moto", "Engavetamento", "Capo~  

## $ automovel <dbl> 0, 3, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, ~  

## $ bicicleta <dbl> 0, 0, NA, 0, 0, 0, 0, 0, 1, 0, 0, 0, ~  

## $ caminhao <dbl> 0, 0, NA, 0, 1, 0, 0, 0, 0, 0, 0, 0, ~  

## $ moto <dbl> 1, 0, NA, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, ~  

## $ onibus <dbl> 0, 0, NA, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~  

## $ outros <dbl> 0, 0, NA, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~  

## $ tracao_animal <dbl> NA, NA, NA, NA, 0, NA, 0, NA, 0, NA~  

## $ transporte_de_cargas_especiais <dbl> NA, NA, NA, NA, 0, NA, 0, NA~  

## $ trator_maquinas <dbl> NA, NA, NA, NA, 0, NA, 0, NA, 0, NA~  

## $ utilitarios <dbl> NA, NA, NA, NA, 0, NA, 0, NA, 0, NA~  

## $ ilesos <dbl> 0, 0, 0, 1, 1, 0, 0, 2, 0, 1, 3, 5, 0, ~  

## $ levemente_feridos <dbl> 1, 1, NA, 2, 0, 0, 0, 0, 1, 0, 0, 0, 0, ~  

## $ moderadamente_feridos <dbl> 0, 0, 3, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, ~  

## $ gravemente_feridos <dbl> 0, 0, NA, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, ~  

## $ mortos <dbl> 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~

```

## Buscando padrões com filter()

- Podemos buscar padrões em strings utilizando o operador `%like%`.
- Vamos filtrar as observações cujo tipo de ocorrência contém a palavra *vítima*.

```

## Operador like %like%
car_crash8 <- car_crash %>%
  filter(tipo_de_ocorrencia %like% "vítima")

glimpse(car_crash8)

```

```

## Rows: 728,448
## Columns: 24

## $ data
## $ horario
## $ n_da_ocorrencia
## $ tipo_de_ocorrencia
## $ km
## $ trecho
## $ sentido
## $ lugar_acidente
## $ tipo_de_acidente
## $ automovel
## $ bicicleta
## $ caminhao
## $ moto
## $ onibus
## $ outros
## $ tracao_animal
## $ transporte_de_cargas_especiais
## $ trator_maquinas
## $ utilitarios
## $ ilesos
## $ levemente_feridos
## $ moderadamente_feridos
## $ gravemente_feridos
## $ mortos

<chr> "01/01/2010", "01/01/2010", "01/01/2010~
<chr> "04:21:00", "02:13:00", "03:35:00", "07~
<chr> "18", "20", "000024/2010", "000038/2010~
<chr> "sem vítima", "sem vítima", "sem vítima~
<chr> "167", "269,5", "77", "52", "33", "24", ~
<chr> "BR-393/RJ", "BR-116/PR", "BR-290/RS", ~
<chr> "Norte", "Sul", "Norte", "Norte", "Nort~
<chr> "Rodovia do Aço", "Autopista Regis Bitt~
<chr> "Derrapagem", "Colisão Traseira", "COLI~
<dbl> 1, 2, 2, 0, 0, 1, 1, 1, 2, 1, NA, 1, 1, ~
<dbl> NA, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
<dbl> NA, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
<dbl> NA, NA, 0, 1, 1, 0, NA, NA, NA, NA, 1, ~
<dbl> NA, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
<dbl> NA, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
<dbl> NA, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
<dbl> NA, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
<dbl> NA, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
<dbl> NA, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
<dbl> NA, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
<dbl> 1, 3, 2, 1, 1, 1, 3, 4, 4, 1, 0, 1, 1, ~
<dbl> 0, NA, 0, 0, 0, 0, NA, NA, 5, NA, 2, NA~
<dbl> 0, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
<dbl> 0, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
<dbl> 0, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~

```

- Podemos filtrar por textos específicos, por exemplo, *ilesa* ou *fatal*. Para isso utilizamos o *grep*.

```
car_crash9 = car_crash %>%
  filter(grepl("ilesa|fatal", tipo_de_ocorrencia))

glimpse(car_crash9)
```

```

## Rows: 2,005
## Columns: 24

## $ data
## $ horario
## $ n_da_ocorrencia
## $ tipo_de_ocorrencia
## $ km
## $ trecho
## $ sentido
## $ lugar_acidente
## $ tipo_de_acidente
## $ automovel
## $ bicicleta
## $ caminhao
## $ moto
## $ onibus
## $ outros
## $ tracao_animal
## $ transporte_de_cargas_especiais <chr> "01/01/2021", "01/01/2021", "01/01/2023~
<chr> "04:46:54", "14:00:09", "12:34:00", "16~
<chr> "23", "83", "58", "90", "95", "121", "1~
<chr> "ac03 - Acidente com vítima lesa", "ac~
<chr> "163,2", "37,8", "17,781", "42", "109", ~
<chr> "BR-050/GO", "BR-050/MG", "BR-116/RJ", ~
<chr> "Norte", "Sul", "Sul", "Sul", "Norte", ~
<chr> "EC0050", "EC0050", "Ecoriominas", "Eco~
<chr> "Atropelamento de Animal", "Choque - De~
<dbl> 1, 1, NA, 2, 1, NA, NA, 1, 1, NA, 1, 1, ~
<dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, ~
<dbl> NA, NA, 1, NA, NA, NA, NA, NA, NA, 1, N~
<dbl> NA, NA, NA, NA, NA, 1, NA, NA, NA, NA, ~
<dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, ~
<dbl> NA, NA, NA, NA, 1, NA, NA, NA, NA, NA, ~
<dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, ~
<dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, ~

```

```

## $ trator_maquinas <dbl> NA, ~
## $ utilitarios <dbl> NA, NA, NA, NA, NA, 1, NA, NA, NA, ~
## $ ilesos <dbl> 1, 2, 1, 4, 2, 2, 1, 1, 1, 1, 7, 5, ~
## $ levemente_feridos <dbl> NA, ~
## $ moderadamente_feridos <dbl> NA, ~
## $ gravemente_feridos <dbl> NA, ~
## $ mortos <dbl> NA, ~

```

## Ordenando e Fatiando Observações

### Ordenando Observações com arrange()

- Para ordenar observações, podemos utilizar a função *arrange()*.
- Vamos ordenar as observações do banco de dados *car\_crash* pela variável *automovel* em ordem decrescente.

```

## Ordenando linhas com arrange()
car_crash10 = car_crash %>%
  arrange(desc(automovel))
glimpse(car_crash10)

```

```

## Rows: 864,561
## Columns: 24
## $ data <chr> "02/03/2016", "27/11/2017", "19/08/2020~"
## $ horario <chr> "20:58:00", "05:37:00", "16:01:00", "11~"
## $ n_da_ocorrencia <chr> "851", "79", "222", "280", "154", "125"~
## $ tipo_de_ocorrencia <chr> "sem vítima", "sem vítima", "Acidente s~"
## $ km <chr> "210,5", "54,385", "537,9", "660,195", ~
## $ trecho <chr> "BR-116/SP", "BR-381/SP", "BR-040/MG", ~
## $ sentido <chr> "Pista Norte", "Norte", "Norte", "Sul", ~
## $ lugar_acidente <chr> "Novadutra", "Autopista Fernão Dias", "~"
## $ tipo_de_acidente <chr> "Outros", "Engavetamento", "Engavetamen~"
## $ automovel <dbl> 15, 15, 14, 13, 13, 12, 12, 12, 12, ~
## $ bicicleta <dbl> NA, NA, 0, NA, 0, 0, NA, 0, NA, 0, N~
## $ caminhao <dbl> NA, 5, 0, 2, 3, 0, NA, 1, 2, 0, 3, 2, 2~
## $ moto <dbl> NA, NA, 3, NA, 0, 0, NA, 0, NA, 0, 0, 1~
## $ onibus <dbl> NA, NA, 0, NA, 0, 0, NA, 0, NA, 0, 0, N~
## $ outros <dbl> NA, NA, 3, 1, 2, 0, NA, 0, NA, 1, 0, NA~
## $ tracao_animal <dbl> NA, NA, 0, NA, 0, 0, NA, 0, NA, 0, 0, N~
## $ transporte_de_cargas_especiais <dbl> NA, NA, 0, NA, 0, NA, NA, NA, NA, 0~
## $ trator_maquinas <dbl> NA, NA, 0, NA, 0, 0, NA, 0, NA, 0, 0, N~
## $ utilitarios <dbl> NA, NA, 0, NA, 0, 0, NA, 0, 1, 0, 0, NA~
## $ ilesos <dbl> 22, 23, 0, 15, 11, 16, 12, 10, 29, 16, ~
## $ levemente_feridos <dbl> NA, NA, 0, 4, 6, 2, NA, 3, 4, 0, 1, 1, ~
## $ moderadamente_feridos <dbl> NA, NA, 0, 2, 3, 0, NA, 0, 1, 0, 3, NA, ~
## $ gravemente_feridos <dbl> NA, NA, 0, NA, 0, 0, NA, 0, 1, 0, 0, NA~
## $ mortos <dbl> NA, NA, 0, 0, NA, 0, 0, NA, 0, 1, 0, 1, N~

```

- Podemos ordenar por mais de uma variável. Vamos ordenar as observações do banco de dados *car\_crash* pela variável *automovel* em ordem decrescente e pelo número de *mortos* em ordem crescente.

```

car_crash11 = car_crash %>%
  arrange(desc(automovel), mortos) %>%
  select(automovel, mortos) %>%
  na.exclude()
head(car_crash11)

```

```

##      automovel mortos
##            <num>  <num>
## 1:        14    0
## 2:        13    0
## 3:        13    0
## 4:        12    0
## 5:        12    0
## 6:        12    1

```

### Fatiando Linhas com slice()

- Para fatiar linhas, podemos utilizar a função slice() .

```

car_crash_slice1 = car_crash %>%
  select(1:5) %>%
  slice(3:5)
car_crash_slice1

```

```

##      data horario n_da_ocorrencia tipo_de_ocorrencia      km
##            <char>   <char>           <char>                  <char> <char>
## 1: 01/01/2010 03:35:00     000024/2010      sem vítima     77
## 2: 01/01/2010 07:31:00     000038/2010      sem vítima     52
## 3: 01/01/2010 04:57:00     000027/2010      sem vítima     33

```

```

car_crash_slice2 = car_crash %>%
  select(1:5) %>%
  slice_head(n = 3)
car_crash_slice2

```

```

##      data horario n_da_ocorrencia tipo_de_ocorrencia      km
##            <char>   <char>           <char>                  <char> <char>
## 1: 01/01/2010 04:21:00          18      sem vítima     167
## 2: 01/01/2010 02:13:00          20      sem vítima   269,5
## 3: 01/01/2010 03:35:00     000024/2010      sem vítima     77

```

```

car_crash_slice3 = car_crash %>%
  select(1:5) %>%
  slice_tail(n = 3)
car_crash_slice3

```

```

##      data horario n_da_ocorrencia tipo_de_ocorrencia      km
##            <char>   <char>           <char>                  <char> <char>
## 1: 31/12/2022 05:05:55          14      Sem vítima 115,100
## 2: 31/12/2022 13:49:33         339  Acidente com Danos Materiais 379,000
## 3: 31/12/2022 12:12:09         188      Com vítima 223,520

```

## Exercícios

1 - Utilizando o banco de dados *storms* , faça o que se pede:

```
View(storms)
```

- Filtre as observações cujo tipo de evento é *Tropical Depression*. Quantas observações existem?

```
storms %>%
  filter(status == "tropical depression") %>%
  nrow()
```

```
## [1] 3569
```

- Filtre as observações cujo tipo de evento é Tropical Depression e a velocidade do vento é maior ou igual a 40. Quantas observações existem?

```
help("storms")
```

```
## inicializando servidor httpd de ajuda ... concluído
```

```
storms %>%
  filter(status == "tropical depression" & wind >= 40) %>%
  nrow()
```

```
## [1] 0
```

- Selecione as variáveis numéricas e ordene as observações pela variável pressure em ordem crescente.

```
storms %>%
  select_if(is.numeric) %>%
  arrange(pressure)
```

```
## # A tibble: 19,537 x 11
##   year month day hour lat long category wind pressure
##   <dbl> <dbl> <int> <dbl> <dbl> <dbl> <dbl> <int> <int>
## 1 2005     10    19    12  17.3 -82.8      5   160     882
## 2 1988      9    14     0  19.7 -83.8      5   160     888
## 3 1988      9    14     6  19.9 -85.3      5   155     889
## 4 1988      9    14    12  20.4 -86.5      5   145     892
## 5 2005     10    19     6   17  -82.2      5   150     892
## 6 2005     10    19    18  17.4 -83.4      5   140     892
## 7 2005     10    20     0  17.9  -84       4   135     892
## 8 2005      9    22     3  24.7 -87.3      5   155     895
## 9 2005      9    22     0  24.5 -86.9      5   150     897
## 10 2005     9    22     6  24.8 -87.6      5   155     897
## # i 19,527 more rows
## # i 2 more variables: tropicalstorm_force_diameter <int>,
## #   hurricane_force_diameter <int>
```

## Renomeando, Realocando e Transmutando Colunas

### Renomeando Colunas com `rename()`

- Para renomear variáveis, podemos utilizar a função `rename()`.
- Vamos renomear a variável automovel para numero\_automoveis no banco de dados `car_crash`.

```
# Rename
car_carsh12 = car_crash %>%
  rename(numero_automoveis = automovel)
glimpse(car_carsh12)

## Rows: 864,561
## Columns: 24
## $ data
## $ horario
## $ n_da_ocorrencia
## $ tipo_de_ocorrencia
## $ km
## $ trecho
## $ sentido
## $ lugar_acidente
## $ tipo_de_acidente
## $ numero_automoveis
## $ bicicleta
## $ caminhao
## $ moto
## $ onibus
## $ outros
## $ tracao_animal
## $ transporte_de_cargas_especiais
## $ trator_maquinas
## $ utilitarios
## $ ilesos
## $ levemente_feridos
## $ moderadamente_feridos
## $ gravemente_feridos
## $ mortos
<chr> "01/01/2010", "01/01/2010", "01/01/2010~
<chr> "04:21:00", "02:13:00", "03:35:00", "07~
<chr> "18", "20", "000024/2010", "000038/2010~
<chr> "sem vítima", "sem vítima", "sem vítima~
<chr> "167", "269,5", "77", "52", "33", "24", ~
<chr> "BR-393/RJ", "BR-116/PR", "BR-290/RS", ~
<chr> "Norte", "Sul", "Norte", "Norte", "Nort~
<chr> "Rodovia do Aço", "Autopista Regis Bitt~
<chr> "Derrapagem", "Colisão Traseira", "COLI~
<dbl> 1, 2, 2, 0, 0, 1, 1, 1, 2, 1, NA, 1, 1, ~
<dbl> NA, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
<dbl> NA, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
<dbl> NA, NA, 0, 1, 1, 0, NA, NA, NA, NA, 1, ~
<dbl> NA, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
<dbl> NA, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
<dbl> NA, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
<dbl> NA, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
<dbl> NA, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
<dbl> 1, 3, 2, 1, 1, 3, 4, 4, 1, 0, 1, 1, ~
<dbl> 0, NA, 0, 0, 0, 0, NA, NA, 5, NA, 2, NA~
<dbl> 0, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
<dbl> 0, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
<dbl> 0, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
```

### Realocando Colunas com `relocate()`

- Para realocar variáveis, podemos utilizar a função `relocate()`.
- Vamos realocar a variável automovel para a primeira posição no banco de dados `car_crash`.

```
# Relocate
car_crash_relocate = car_crash %>%
  relocate(automovel, .before = 1)
glimpse(car_crash_relocate)

## Rows: 864,561
## Columns: 24
## $ automovel
<dbl> 1, 2, 2, 0, 0, 1, 1, 1, 2, 1, NA, 1, 1, ~
```

```

## $ data
## $ horario
## $ n_da_ocorrencia
## $ tipo_de_ocorrencia
## $ km
## $ trecho
## $ sentido
## $ lugar_acidente
## $ tipo_de_acidente
## $ bicicleta
## $ caminhao
## $ moto
## $ onibus
## $ outros
## $ tracao_animal
## $ transporte_de_cargas_especiais
## $ trator_maquinas
## $ utilitarios
## $ ilesos
## $ levemente_feridos
## $ moderadamente_feridos
## $ gravemente_feridos
## $ mortos

<chr> "01/01/2010", "01/01/2010", "01/01/2010"
<chr> "04:21:00", "02:13:00", "03:35:00", "07~"
<chr> "18", "20", "000024/2010", "000038/2010~"
<chr> "sem vítima", "sem vítima", "sem vítima~"
<chr> "167", "269,5", "77", "52", "33", "24", ~
<chr> "BR-393/RJ", "BR-116/PR", "BR-290/RS", ~
<chr> "Norte", "Sul", "Norte", "Norte", "Nort~
<chr> "Rodovia do Aço", "Autopista Regis Bitt~
<chr> "Derrapagem", "Colisão Traseira", "COLI~
<dbl> NA, NA, 0, 0, 0, NA, NA, NA, NA, NA, ~
<dbl> NA, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
<dbl> NA, NA, 0, 1, 1, 0, NA, NA, NA, NA, 1, ~
<dbl> NA, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
<dbl> NA, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
<dbl> NA, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
<dbl> NA, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
<dbl> NA, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
<dbl> NA, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
<dbl> 1, 3, 2, 1, 1, 1, 3, 4, 4, 1, 0, 1, 1, ~
<dbl> 0, NA, 0, 0, 0, 0, NA, NA, 5, NA, 2, NA~
<dbl> 0, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
<dbl> 0, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
<dbl> 0, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~

```

- Vamos realocar a variável mortos para a última posição no banco de dados car\_crash .

```
car_crash_relocate2 = car_crash %>%
  relocate(mortos, .after = last_col())
glimpse(car_crash_relocate2)
```

```

## Rows: 864,561
## Columns: 24

## $ data
## $ horario
## $ n_da_ocorrencia
## $ tipo_de_ocorrencia
## $ km
## $ trecho
## $ sentido
## $ lugar_acidente
## $ tipo_de_acidente
## $ automovel
## $ bicicleta
## $ caminhao
## $ moto
## $ onibus
## $ outros
## $ tracao_animal
## $ transporte_de_cargas_especiais
## $ trator_maquinas
## $ utilitarios
## $ ilesos
## $ levemente_feridos

```

```

## $ moderadamente_feridos      <dbl> 0, NA, 0, 0, 0, 0, NA, NA, NA, NA, ~
## $ gravemente_feridos        <dbl> 0, NA, 0, 0, 0, 0, NA, NA, NA, NA, ~
## $ mortos                      <dbl> 0, NA, 0, 0, 0, 0, NA, NA, NA, NA, ~

```

### Transformando Dados com transmute()

- Para transformar dados, podemos utilizar a função *transmute()*.
- Vamos criar uma nova variável chamada *automovel\_10* que é o número de automóveis envolvidos em acidentes dividido por 10 no banco de dados *car\_crash*.
- Quando utilizamos o *transmute()*, apenas as variáveis criadas são mantidas no novo data frame.

```

# Transmute
car_crash_transmute <- car_crash %>%
  transmute(automovel_10 = automovel / 10)
glimpse(car_crash_transmute)

```

```

## Rows: 864,561
## Columns: 1
## $ automovel_10 <dbl> 0.1, 0.2, 0.2, 0.0, 0.0, 0.1, 0.1, 0.1, 0.2, 0.1, NA, 0.1~

```

### Alterando NA s com replace\_na()

- Para alterar valores *NA*, podemos utilizar a função *replace\_na()*.
- Vamos substituir os valores *NA* da variável *mortos* por 0 no banco de dados *car\_crash*.

```

# NA Replace
car_crash_replace_na <- car_crash %>%
  mutate(mortos = replace_na(mortos, 0))
glimpse(car_crash_replace_na)

```

```

## Rows: 864,561
## Columns: 24
## $ data                  <chr> "01/01/2010", "01/01/2010", "01/01/2010~"
## $ horario                <chr> "04:21:00", "02:13:00", "03:35:00", "07~"
## $ n_da_ocorrencia       <chr> "18", "20", "000024/2010", "000038/2010~"
## $ tipo_de_ocorrencia    <chr> "sem vítima", "sem vítima", "sem vítima~"
## $ km                     <chr> "167", "269,5", "77", "52", "33", "24", ~
## $ trecho                  <chr> "BR-393/RJ", "BR-116/PR", "BR-290/RS", ~
## $ sentido                 <chr> "Norte", "Sul", "Norte", "Norte", "Nort~"
## $ lugar_acidente         <chr> "Rodovia do Aço", "Autopista Regis Bitt~
## $ tipo_de_acidente       <chr> "Derrapagem", "Colisão Traseira", "COLI~"
## $ automovel               <dbl> 1, 2, 2, 0, 0, 1, 1, 1, 2, 1, NA, 1, 1, ~
## $ bicicleta                <dbl> NA, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
## $ caminhao                 <dbl> NA, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
## $ moto                     <dbl> NA, NA, 0, 1, 1, 0, NA, NA, NA, NA, 1, ~
## $ onibus                   <dbl> NA, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
## $ outros                    <dbl> NA, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
## $ tracao_animal            <dbl> NA, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
## $ transporte_de_cargas_especiais <dbl> NA, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
## $ trator_maquinas          <dbl> NA, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
## $ utilitarios              <dbl> NA, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~

```

```

## $ilesos <dbl> 1, 3, 2, 1, 1, 1, 3, 4, 4, 1, 0, 1, 1, ~
## $levemente_feridos <dbl> 0, NA, 0, 0, 0, NA, NA, 5, NA, 2, NA, ~
## $moderadamente_feridos <dbl> 0, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
## $gravemente_feridos <dbl> 0, NA, 0, 0, 0, 0, NA, NA, NA, NA, NA, ~
## $mortos <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~

```

## Classificando Dados com `cut()`

- Para classificar dados, podemos utilizar a função `cut()`.
  - Vamos classificar a variável `automovel` em 3 categorias: *sem automóveis, entre 1 e 3 automóveis, mais do que três* no banco de dados `car_crash`.

```
# Cut
car_crash_cut <- car_crash %>%
  mutate(automovel = replace_na(automovel, 0)) %>%
  mutate(automovel_cat = cut(automovel,
                             breaks = c(-Inf, 0, 3, Inf),
                             labels = c("sem automóveis",
                                       "entre 1 e 3 automóveis",
                                       "mais do que três")))
glimpse(car_crash_cut)
```

```

## Rows: 864,561
## Columns: 25
## $ data
## $ horario
## $ n_da_ocorrencia
## $ tipo_de_ocorrencia
## $ km
## $ trecho
## $ sentido
## $ lugar_acidente
## $ tipo_de_acidente
## $ automovel
## $ bicicleta
## $ caminhao
## $ moto
## $ onibus
## $ outros
## $ tracao_animal
## $ transporte_de_cargas_especiais
## $ trator_maquinas
## $ utilitarios
## $ ilesos
## $ levemente_feridos
## $ moderadamente_feridos
## $ gravemente_feridos
## $ mortos
## $ automovel_cat

table( car_crash_cut$automovel,
       car_crash_cut$automovel_cat)

```

```

## sem automóveis entre 1 e 3 automóveis mais do que três
## 0          269093           0           0
## 1          0             449366           0
## 2          0             111714           0
## 3          0             24817            0
## 4          0               0           6751
## 5          0               0           1850
## 6          0               0           613
## 7          0               0           207
## 8          0               0            77
## 9          0               0            41
## 10         0               0            14
## 11         0               0             7
## 12         0               0             5
## 13         0               0             3
## 14         0               0             1
## 15         0               0             2

```

## Sumarizando e Agrupando Dados

### Sumarizando Dados com summarise()

- Para sumarizar dados, podemos utilizar a função *summarise()*.
- Vamos sumarizar o número total de automóveis envolvidos em acidentes no banco de dados *car\_crash*.

```

## Summarise
car_crash13 = car_crash %>%
  summarise(total_automoveis = sum(automovel, na.rm = TRUE))
car_crash13

##   total_automoveis
## 1           789971

sum(car_crash$automovel, na.rm = T)

## [1] 789971

```

### Sumarizando Múltiplas Variáveis com summarise()

- Podemos sumarizar mais de uma variável. Vamos sumarizar o número total de automóveis envolvidos em acidentes e o número total de mortos.

```

car_crash14 = car_crash %>%
  summarise(total_automoveis = sum(automovel, na.rm = TRUE),
            total_mortos = sum(mortos, na.rm = TRUE),
            n = n(),
            media_mortos = mean(mortos, na.rm = TRUE))
car_crash14

```

```
##    total_automoveis total_mortos      n media_mortos
## 1           789971       19430 864561   0.06615864

sum(car_crash$mortos, na.rm = T)

## [1] 19430

nrow(car_crash)

## [1] 864561

mean(car_crash$mortos, na.rm = T)

## [1] 0.06615864
```

## Agrupando Dados com group\_by()

- Para agrupar dados, podemos utilizar a função `group_by()`.
  - Vamos agrupar o banco de dados `car_crash` pela variável `ano`.
  - Primeiro, vamos criar a variável `ano` a partir da variável `data`.
  - Para trabalharmos com datas, precisamos utilizar o pacote **lubridate**.
  - A função `dmy()` é utilizada para transformar strings no formato dia-mês-ano em objetos do tipo `date`.
  - A função `year()` é utilizada para extrair o ano de um objeto do tipo `date`.

```

require(lubridate)
## Agrupamento com group_by()
car_crash15 = car_crash %>%
  mutate(ano = year(dmy(data))) %>%
  group_by(ano)
glimpse(car_crash15)

## Rows: 864,561
## Columns: 25
## Groups: ano [14]
## $ data
## $ horario
## $ n_da_ocorrencia
## $ tipo_de_ocorrencia
## $ km
## $ trecho
## $ sentido
## $ lugar_acidente
## $ tipo_de_acidente
## $ automovel
## $ bicicleta
## $ caminhao
## $ moto
## $ onibus
## $ outros
## $ tracao_animal

```

```

## $ transporte_de_cargas_especiais <dbl> NA, NA, 0, 0, 0, NA, NA, NA, NA, NA, ~
## $ trator_maquinas <dbl> NA, NA, 0, 0, 0, NA, NA, NA, NA, NA, ~
## $ utilitarios <dbl> NA, NA, 0, 0, 0, NA, NA, NA, NA, NA, ~
## $ ilesos <dbl> 1, 3, 2, 1, 1, 3, 4, 4, 1, 0, 1, 1, ~
## $ levemente_feridos <dbl> 0, NA, 0, 0, 0, NA, NA, 5, NA, 2, NA~
## $ moderadamente_feridos <dbl> 0, NA, 0, 0, 0, NA, NA, NA, NA, ~
## $ gravemente_feridos <dbl> 0, NA, 0, 0, 0, NA, NA, NA, NA, ~
## $ mortos <dbl> 0, NA, 0, 0, 0, NA, NA, NA, NA, ~
## $ ano <int> 2010, 2010, 2010, 2010, 2010, 2010, 201~
```

### Sumarizando Dados com summarise()

- Agora vamos sumarizar o número total de automóveis envolvidos em acidentes e o número total de mortos por ano.

```

car_crash16 = car_crash %>%
  mutate(ano = year(dmy(data))) %>%
  group_by(ano) %>%
  summarise(total_automoveis = sum(automovel, na.rm = TRUE),
            total_mortos = sum(mortos, na.rm = TRUE))
head(car_crash16)

## # A tibble: 6 x 3
##       ano total_automoveis total_mortos
##   <int>          <dbl>        <dbl>
## 1  2010           51223       1472
## 2  2011           57531       1514
## 3  2012           59020       1468
## 4  2013           59855       1502
## 5  2014           67626       1684
## 6  2015           72166       1786
```

### Encadeando Funções

- Podemos encadear funções utilizando o operador %>%.
- Vamos filtrar as observações cujo tipo de ocorrência é com vítima e sumarizar o número total de automóveis envolvidos em acidentes e o número total de mortos.

```

car_crash17 = car_crash %>%
  filter(tipo_de_ocorrencia == "com vítima") %>%
  summarise(total_automoveis = sum(automovel, na.rm = TRUE),
            total_mortos = sum(mortos, na.rm = TRUE))
car_crash17

##   total_automoveis total_mortos
## 1           152409      13356
```

### Exercícios

Utilizando o banco de dados starwars faça o que se pede:

```
View(starwars)
```

- Qual é o número total de espécies únicas presentes? Qual a frequência de indivíduos por espécie?

```
starwars %>%
  summarise(n_especies = n_distinct(species))
```

```
## # A tibble: 1 x 1
##   n_especies
##       <int>
## 1          38
```

```
starwars %>%
  group_by(species) %>%
  summarise(freq_especies = n()) %>%
  arrange(desc(freq_especies))
```

```
## # A tibble: 38 x 2
##   species freq_especies
##   <chr>      <int>
## 1 Human        35
## 2 Droid         6
## 3 <NA>         4
## 4 Gungan        3
## 5 Kaminoan     2
## 6 Mirialan     2
## 7 Twi'lek       2
## 8 Wookiee      2
## 9 Zabrak        2
## 10 Aleena       1
## # i 28 more rows
```

- Calcule a altura média de personagens masculinos e femininos.

```
starwars %>%
  filter(sex %in% c("female", "male")) %>%
  group_by(sex) %>%
  summarise(media_altura = mean(height, na.rm = TRUE))
```

```
## # A tibble: 2 x 2
##   sex    media_altura
##   <chr>      <dbl>
## 1 female     172.
## 2 male       179.
```

- Qual é o peso médio dos personagens de cada espécie para personagens masculinos?

```
starwars %>%
  filter(sex == "male") %>%
  group_by(species) %>%
  summarise(media_peso = mean(mass, na.rm = TRUE))
```

```

## # A tibble: 31 x 2
##   species    media_peso
##   <chr>        <dbl>
## 1 Aleena       15
## 2 Besalisk     102
## 3 Cerean       82
## 4 Chagrian     NaN
## 5 Dug          40
## 6 Ewok          20
## 7 Geonosian    80
## 8 Gungan        74
## 9 Human         85.7
## 10 Iktotchi    NaN
## # i 21 more rows

```

- Para cada espécie presente na base de dados, identifique o personagem mais pesado e seu peso correspondente.

```

starwars %>%
  group_by(species) %>%
  filter(mass == max(mass, na.rm = TRUE)) %>%
  select(species, name, mass)

```

```

## Warning: There were 6 warnings in 'filter()'.
## The first warning was:
## i In argument: 'mass == max(mass, na.rm = TRUE)'.
## i In group 4: 'species = "Chagrian"'.
## Caused by warning in 'max()':
## ! nenhum argumento não faltante para max; retornando -Inf
## i Run 'dplyr::last_dplyr_warnings()' to see the 5 remaining warnings.

```

```

## # A tibble: 32 x 3
## # Groups:   species [32]
##   species      name      mass
##   <chr>        <chr>     <dbl>
## 1 Human        Darth Vader 136
## 2 Rodian       Greedo      74
## 3 Hutt         Jabba Desilijic Tiure 1358
## 4 <NA>         Jek Tono Porkins 110
## 5 Yoda's species Yoda      17
## 6 Droid        IG-88      140
## 7 Trandoshan   Bossk      113
## 8 Mon Calamari Ackbar     83
## 9 Ewok         Wicket Systri Warrick 20
## 10 Sullustan   Nien Nunb   68
## # i 22 more rows

```

## Trabalhando com Datas - lubridate

### Manipulação de datas

- Quando importamos datas em R (dentro de um data frame), elas são importadas como strings.

- Precisamos, portanto, transformar essas strings em objetos do tipo date para podermos manipulá-las, como por exemplo, extrair o ano, o mês, o dia, etc.
- Podemos extrair o ano, o mês e o dia de uma data utilizando as funções year() , month() e day() .

```
## Trabalhando com datas - lubridate
car_crash %>%
  mutate(data = dmy(data)) %>%
  select(data) %>%
  glimpse()

## Rows: 864,561
## Columns: 1
## $ data <date> 2010-01-01, 2010-01-01, 2010-01-01, 2010-01-01, 2010-01-01, 2010~

car_crash %>%
  mutate(data = dmy(data)) %>%
  mutate(ano = year(data),
        mes = month(data),
        dia = day(data)) %>%
  select(data, ano, mes, dia) %>%
  glimpse()

## Rows: 864,561
## Columns: 4
## $ data <date> 2010-01-01, 2010-01-01, 2010-01-01, 2010-01-01, 2010-01-01, 2010~
## $ ano  <int> 2010, 2010, 2010, 2010, 2010, 2010, 2010, 2010, 2010, 2010, ~
## $ mes   <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
## $ dia   <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
```

- Podemos calcular a diferença entre duas datas utilizando a função difftime() .

```
car_crash %>%
  mutate(data = dmy(data)) %>%
  mutate(dias_desde_acidente = difftime(Sys.Date(), data, units = "days")) %>%
  select(data, dias_desde_acidente) %>%
  head()

##           data dias_desde_acidente
##           <Date>             <difftime>
## 1: 2010-01-01      5788 days
## 2: 2010-01-01      5788 days
## 3: 2010-01-01      5788 days
## 4: 2010-01-01      5788 days
## 5: 2010-01-01      5788 days
## 6: 2010-01-01      5788 days
```

- Podemos somar ou subtrair dias de uma data utilizando a função lubridate::days() .

```
car_crash %>%
  mutate(data = dmy(data)) %>%
  mutate(data_mais_10_dias = data + lubridate::days(10)) %>%
  select(data, data_mais_10_dias) %>%
  head()
```

```

##           data data_mais_10_dias
##      <Date>          <Date>
## 1: 2010-01-01      2010-01-11
## 2: 2010-01-01      2010-01-11
## 3: 2010-01-01      2010-01-11
## 4: 2010-01-01      2010-01-11
## 5: 2010-01-01      2010-01-11
## 6: 2010-01-01      2010-01-11

```

### Manipulação de datas - Hora, minutos e segundos

- Podemos extrair a hora, os minutos e os segundos de uma data utilizando as funções hour() , minute() e second() .

```

## Extraendo componentes de data e hora
data <- ymd_hms("2023-08-21 15:30:45")
ano <- year(data)
mes <- month(data)
dia <- day(data)
hora <- hour(data)
minuto <- minute(data)
segundo <- second(data)

print(ano)

```

```
## [1] 2023
```

```
print(mes)
```

```
## [1] 8
```

```
print(dia)
```

```
## [1] 21
```

```
print(hora)
```

```
## [1] 15
```

```
print(minuto)
```

```
## [1] 30
```

```
print(segundo)
```

```
## [1] 45
```

### Conversão de fuso horário

- Podemos converter o fuso horário de uma data utilizando a função with\_tz() .

```

# Data original no fuso horário de Nova Iorque
data_ny <- ymd_hms("2025-10-21 12:00:00", tz = "America/New_York")

# Converter para o fuso horário de Londres
data_london <- with_tz(data_ny, tz = "Europe/London")

print(data_ny)

## [1] "2025-10-21 12:00:00 EDT"

print(data_london)

```

```
## [1] "2025-10-21 17:00:00 BST"
```

## Exercícios - Datas

Utilizando o banco de dados car\_crash faça o que se pede:

- Quais os meses do ano com maior número de acidentes fatais?

```

## Exercícios com datas
car_crash %>%
  mutate(data = dmy(data)) %>%
  mutate(ano = year(data),
        mes = month(data)) %>%
  select(data, ano, mes, mortos) %>%
  filter(mortos > 0) %>%
  group_by(mes) %>%
  summarise(total_mortos = sum(mortos)) %>%
  arrange(desc(total_mortos))

## # A tibble: 12 x 2
##       mes total_mortos
##   <int>      <dbl>
## 1     12      1901
## 2     7       1789
## 3     5       1660
## 4    10      1643
## 5     8       1640
## 6     9       1607
## 7     1       1574
## 8     3       1572
## 9     6       1555
## 10    11      1555
## 11    2       1482
## 12    4       1452

```

- Quais os dias da semana com maior número de acidentes fatais? Dica: Busque por uma função que retorne o dia da semana a partir de uma data.

```

car_crash %>%
  mutate(data = dmy(data)) %>%
  mutate(dia_semana = lubridate::wday(data, label = T, abbr = F)) %>%
  select(dia_semana, mortos) %>%
  filter(mortos > 0) %>%
  group_by(dia_semana) %>%
  summarise(total_mortos_dia = sum(mortos)) %>%
  arrange(desc(total_mortos_dia))

## # A tibble: 7 x 2
##   dia_semana   total_mortos_dia
##   <ord>           <dbl>
## 1 domingo        3739
## 2 sábado         3691
## 3 sexta-feira    2862
## 4 quinta-feira   2386
## 5 segunda-feira  2383
## 6 terça-feira    2253
## 7 quarta-feira   2116

```

## Pivotagem de Dados

### O que é tidy data?

Hadley Wickham, em seu artigo “Tidy Data” (2014), define que um conjunto de dados é tidy se:

1. Cada variável forma uma coluna.
2. Cada observação forma uma linha.
3. Cada tipo de unidade observacional forma uma tabela.

Existem dois formatos de dados que podem ser tidy:

- Dados em formato wide.
- Dados em formato long.

### Dados no formato wide

- Cada variável é representada por uma coluna separada e cada observação (ou instância) ocupa uma única linha.
- Adequado para conjuntos de dados com poucas variáveis, onde as informações são bem condensadas.

### Dados no formato long

Exemplos

**Considere o seguinte conjunto de dados em formato wide:**

	id	nome
idade	25	
sexo	F	
altura	1.65	
	2	João
	30	
	M	
	1.80	
Ana		
Maria		
Pedro		
22		
4		
1.70		
28		
1.75		

**Para passar para o formato long, basta empilhar as variáveis:**

```
|-----| | id | variável | valor
| :— | :— | :— | 1 | nome | Ana | 1 | idade | 25 | 1 | sexo | F | 1 | altura | 1.65 | 2 | nome | João | 2
| idade | 30 | 2 | sexo | M | 2 | altura | 1.80 | -----|
```

**Considere o seguinte conjunto de dados com informações sobre tratamento de indivíduos com pedra nos rins:**

Tamanho da pedra	Tratamento A (Recuperados)	Tratamento A (Falhas)	Tratamento B (Recuperados)	Tratamento B (Falhas)
Pequena	10	5	15	3
Média	5	3	10	2
Grande	2	1	5	1

**Para passar para o formato long, basta empilhar as variáveis:**

Tamanho da pedra	Tratamento	Recuperados	Falhas
Pequena	A	10	5
Pequena	B	15	3
Média	A	5	3
Média	B	10	2
Grande	A	2	1
Grande	B	5	1

## Pivotando dados em R

- Vamos utilizar o banco de dados table1 .
- Dados de casos reportados de Tuberculose e o tamanho da população em dois anos para três países.
- Esses dados são provenientes dos dados WHO.

table1

```
## # A tibble: 6 x 4
##   country     year  cases population
##   <chr>      <dbl> <dbl>      <dbl>
## 1 Afghanistan 1999    745 19987071
## 2 Afghanistan 2000   2666 20595360
## 3 Brazil       1999  37737 172006362
## 4 Brazil       2000  80488 174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

- A função pivot\_wider() é utilizada para transformar dados de formato long para wide.
- A função pivot\_wider() requer os seguintes argumentos:
  - names\_from : coluna que contém os nomes das variáveis que serão transformadas em colunas.
  - values\_from : coluna que contém os valores das variáveis que serão transformadas em colunas.
- Vamos transformar os dados de table1 para o formato wide. Suponha que queremos que os dados sejam organizados por país e ano, e as observações sejam os casos de tuberculose.

```
table1 %>%
  select(-population) %>%
  pivot_wider(names_from = year,
              values_from = cases)
```

```
## # A tibble: 3 x 3
##   country      '1999' '2000'
##   <chr>        <dbl>   <dbl>
## 1 Afghanistan    745    2666
## 2 Brazil         37737   80488
## 3 China          212258  213766
```

### Pivotando com mais de uma variável

- Suponha que queremos que os dados sejam organizados por país, e as observações sejam os casos de tuberculose, separados por ano e tamanho da população.

```
table1 %>%
  pivot_wider(names_from = year,
              values_from = c(cases, population))
```

```
## # A tibble: 3 x 5
##   country      cases_1999 cases_2000 population_1999 population_2000
##   <chr>        <dbl>     <dbl>       <dbl>           <dbl>
## 1 Afghanistan    745      2666      19987071      20595360
## 2 Brazil         37737    80488     172006362     174504898
## 3 China          212258   213766    1272915272    1280428583
```

- A função pivot\_longer() é utilizada para transformar dados de formato wide para long.
- A função pivot\_longer() requer os seguintes argumentos:
  - cols : colunas que serão empilhadas.
  - names\_to : coluna que conterá os nomes das variáveis empilhadas.
  - values\_to : coluna que conterá os valores das variáveis empilhadas.
  - values\_fill : valor que preencherá as células vazias.
  - values\_fn : função que será aplicada aos valores empilhados.
- Vamos transformar os dados de table1 para o formato long. Suponha que queremos que os dados sejam organizados por país, e as observações sejam os casos de tuberculose e a população.

```
table1 %>%
  pivot_longer(cols = c(cases, population),
               names_to = "variable",
               values_to = "total")
```

```
## # A tibble: 12 x 4
##   country      year variable      total
##   <chr>        <dbl> <chr>        <dbl>
## 1 Afghanistan  1999 cases          745
## 2 Afghanistan  1999 population  19987071
## 3 Afghanistan  2000 cases          2666
## 4 Afghanistan  2000 population  20595360
```

```

## 5 Brazil      1999 cases        37737
## 6 Brazil      1999 population 172006362
## 7 Brazil      2000 cases        80488
## 8 Brazil      2000 population 174504898
## 9 China       1999 cases        212258
## 10 China      1999 population 1272915272
## 11 China      2000 cases        213766
## 12 China      2000 population 1280428583

```

## Separando observações

- Algumas vezes, as observações estão agrupadas em uma única coluna e precisamos separar elas.
- A função separate() é utilizada para separar observações em diferentes colunas.
- Observe os dados em table3.

```
table3
```

```

## # A tibble: 6 x 3
##   country     year rate
##   <chr>       <dbl> <chr>
## 1 Afghanistan 1999 745/19987071
## 2 Afghanistan 2000 2666/20595360
## 3 Brazil      1999 37737/172006362
## 4 Brazil      2000 80488/174504898
## 5 China       1999 212258/1272915272
## 6 China       2000 213766/1280428583

```

- Suponha que queremos separar a coluna rate em duas colunas: cases e population .

```
table3 %>%
  separate(rate, into = c("cases",
                         "population"))
```

```

## # A tibble: 6 x 4
##   country     year cases population
##   <chr>       <dbl> <chr>    <chr>
## 1 Afghanistan 1999 745     19987071
## 2 Afghanistan 2000 2666    20595360
## 3 Brazil      1999 37737   172006362
## 4 Brazil      2000 80488   174504898
## 5 China       1999 212258  1272915272
## 6 China       2000 213766  1280428583

```

## Juntando observações

- função unite() é utilizada para juntar observações de diferentes colunas em uma única coluna.
- Suponha que queremos juntar as colunas cases e population em uma única coluna chamada rate .

```
table1 %>%
  unite(rate, cases, population, sep = "/")
```

```

## # A tibble: 6 x 3
##   country     year rate
##   <chr>      <dbl> <chr>
## 1 Afghanistan 1999 745/19987071
## 2 Afghanistan 2000 2666/20595360
## 3 Brazil      1999 37737/172006362
## 4 Brazil      2000 80488/174504898
## 5 China       1999 212258/1272915272
## 6 China       2000 213766/1280428583

```

## Exercícios

Utilizando os dados de flights , do pacote *nycflights13* , crie uma matriz que mostra o número de voos entre cada par de aeroportos.

```

# Utilizando os dados do pacote nycflights13
require(nycflights13)

## Carregando pacotes exigidos: nycflights13

## Warning: pacote 'nycflights13' foi compilado no R versão 4.5.2

library(nycflights13)
flights %>%
  count(origin, dest) %>%
  pivot_wider(names_from = origin,
              values_from = n,
              values_fill = 0)

## # A tibble: 105 x 4
##   dest    EWR    JFK    LGA
##   <chr> <int> <int> <int>
## 1 ALB      439     0     0
## 2 ANC       8     0     0
## 3 ATL     5022   1930  10263
## 4 AUS      968   1471     0
## 5 AVL      265     0    10
## 6 BDL      443     0     0
## 7 BNA     2336    730   3267
## 8 BOS     5327   5898   4283
## 9 BQN      297    599     0
## 10 BTV     931   1364   294
## # i 95 more rows

```

## Strings

### Manipulando Strings em R

- Strings são sequências de caracteres que representam texto.
- Em R, strings são representadas por aspas simples ( ' ) ou aspas duplas ( " ).
- O pacote stringr , parte do tidyverse, oferece diversas funções para manipulação de strings.

## Funções Básicas do stringr

- str\_length(): Retorna o comprimento de uma string.
- str\_to\_lower(): Converte uma string para minúsculas.
- str\_to\_upper(): Converte uma string para maiúsculas.
- str\_sub(): Extrai uma substring de uma string.
- str\_replace(): Substitui uma parte de uma string por outra.
- str\_detect(): Verifica se uma string contém um padrão específico.

## Exemplos de Manipulação de Strings

```
library(stringr)
texto <- "Olá, Mundo!"

# Comprimento da string
str_length(texto)

## [1] 11

# Converter para minúsculas
str_to_lower(texto)

## [1] "olá, mundo!"

# Converter para maiúsculas
str_to_upper(texto)

## [1] "OLÁ, MUNDO!"

str_to_title(texto)

## [1] "Olá, Mundo!"

# Converter para sentença
str_to_sentence(texto)

## [1] "Olá, mundo!"

# Extrair substring
str_sub(texto, 1, 3)

## [1] "Olá"

# Substituir parte da string
str_replace(texto, "Mundo", "R")

## [1] "Olá, R!"
```

```
# Verificar se a string contém um padrão  
str_detect(texto, "Mundo")
```

```
## [1] TRUE
```

```
str_detect(texto, "R!")
```

```
## [1] FALSE
```

## Regex Básico

- Regex (expressões regulares) são padrões utilizados para buscar e manipular strings.
- Alguns metacaracteres comuns:
  - . : Corresponde a qualquer caractere.
  - ^ : Início da string.
  - \$ : Fim da string.
  - - : Zero ou mais ocorrências do caractere anterior.
  - - : Uma ou mais ocorrências do caractere anterior.
  - ? : Zero ou uma ocorrência do caractere anterior.
  - [] : Conjunto de caracteres.
  - | : Operador “ou”.

```
# Expressões Regulares (Regex)
```

```
# Correspondem qualquer caractere  
str_detect("abc", "a.c") # TRUE
```

## Exemplos de Regex

```
## [1] TRUE
```

```
# Início da string  
str_detect("abc", "^a") # TRUE
```

```
## [1] TRUE
```

```
# Fim da string  
str_detect("abc", "c$") # TRUE
```

```
## [1] TRUE
```

```

# Zero ou mais ocorrências
str_detect("aaab", "a*b") # TRUE

## [1] TRUE

# Uma ou mais ocorrências
str_detect("aaab", "a+b") # TRUE

## [1] TRUE

# Conjunto de caracteres: corresponde a 'a', 'b' ou 'c'
str_detect("abc", "[abc]") # TRUE

## [1] TRUE

```

## Para saber mais

Cheatsheet de expressões regulares Cheatsheet do pacote stringr Testar o Regex online Palavra cruzada

## Combinação de dados

### Concatenação

- A concatenação permite adicionar novas observações a uma tabela ou novas variáveis.
- Seja por linha ou colunas, entradas com NA são criadas para os índices que não foram especificados.

### Criação de um tibble

```

library(tidyverse)
# Tabela com alunos do curso de
# Matemática e de Estatística.
df1 <- tibble(
  mat = c(256, 487, 965,
  125, 458, 874, 963),
  nome = c("João", "Vanessa", "Tiago",
  "Luana", "Gisele", "Pedro",
  "André"),
  curso = c("Mat", "Mat", "Est", "Est",
  "Est", "Mat", "Est"),
  prova1 = c(80, 75, 95, 70, 45, 55, 30),
  prova2 = c(90, 75, 80, 85, 50, 75, NA),

```

```
prova3 = c(80, 75, 75, 50, NA, 90, 30),  
faltas = c(4, 4, 0, 8, 16, 0, 20))
```

```
df1
```

### Criação por colunas

```
## # A tibble: 7 x 7  
##   mat nome    curso prova1 prova2 prova3 faltas  
##   <dbl> <chr>   <chr>  <dbl>  <dbl>  <dbl>  <dbl>  
## 1   256 João     Mat      80      90      80      4  
## 2   487 Vanessa  Mat      75      75      75      4  
## 3   965 Tiago    Est      95      80      75      0  
## 4   125 Luana    Est      70      85      50      8  
## 5   458 Gisele   Est      45      50      NA     16  
## 6   874 Pedro    Mat      55      75      90      0  
## 7   963 André    Est      30      NA     30     20
```

```
# Informações de cadastro dos alunos  
# em outra base de dados.  
df_extra <- tribble(  
~mat, ~nome, ~idade, ~bolsista,  
256, 'João' , 18, "S",  
965, 'Tiago' , 18, "N",  
285, 'Tiago' , 22, "N",  
125, 'Luana' , 21, "S",  
874, 'Pedro' , 19, "N",  
321, 'Mia'   , 18, "N",  
669, 'Luana' , 19, "S",  
967, 'André' , 20, "N",  
)  
  
df_extra
```

### Criação por linhas

```
## # A tibble: 8 x 4  
##   mat nome  idade bolsista  
##   <dbl> <chr> <dbl> <chr>  
## 1   256 João    18 S  
## 2   965 Tiago   18 N  
## 3   285 Tiago   22 N  
## 4   125 Luana   21 S  
## 5   874 Pedro   19 N  
## 6   321 Mia     18 N  
## 7   669 Luana   19 S  
## 8   967 André   20 N
```

### Concatenação

```
# Concatenação na vertical (pilha).
bind_rows(df1[1:3, c(1, 3, 5)],
df1[5:7, c(1, 3, 5, 4)],
df1[4, c(1, 5, 4)])
```

### De linhas (vertical)

```
## # A tibble: 7 x 4
##       mat curso prova2 prova1
##   <dbl> <chr>  <dbl>   <dbl>
## 1    256 Mat      90     NA
## 2    487 Mat      75     NA
## 3    965 Est      80     NA
## 4    458 Est      50     45
## 5    874 Mat      75     55
## 6    963 Est      NA     30
## 7    125 <NA>     85     70
```

```
# Concatenação na horizontal (fila).
bind_cols(df1[, c(1:3)],
df1[, c(6:7)])
```

### De colunas (horizontal)

```
## # A tibble: 7 x 5
##       mat nome  curso prova3 faltas
##   <dbl> <chr>  <chr>  <dbl>   <dbl>
## 1    256 João   Mat      80     4
## 2    487 Vanessa Mat      75     4
## 3    965 Tiago  Est      75     0
## 4    125 Luana  Est      50     8
## 5    458 Gisele Est      NA    16
## 6    874 Pedro  Mat      90     0
## 7    963 André  Est      30    20
```

## Junções

- Junções permitem parear dados de tabelas separadas quando elas possuem uma chave (ou chave primária).
- As operações de junção podem ser inicialmente de 4 tipos:
  - Junção por interseção (inner join).
  - Junção por união (full join).
  - Junção à esquerda (left join).
  - Junção à direita (right join).
  - Existe também os exclusive joins.

```

# Full join = união.
full_join(df1, df_extra,
by = c("mat" = "mat", "nome"))

## # A tibble: 11 x 9
##   mat nome  curso prova1 prova2 prova3 faltas idade bolsista
##   <dbl> <chr> <chr>  <dbl>  <dbl>  <dbl>  <dbl> <dbl> <chr>
## 1 256 João  Mat     80     90     80      4    18 S
## 2 487 Vanessa Mat    75     75     75      4    NA <NA>
## 3 965 Tiago  Est    95     80     75      0    18 N
## 4 125 Luana  Est    70     85     50      8    21 S
## 5 458 Gisele  Est    45     50     NA     16    NA <NA>
## 6 874 Pedro  Mat    55     75     90      0    19 N
## 7 963 André  Est    30     NA     30     20    NA <NA>
## 8 285 Tiago <NA>    NA     NA     NA     NA    22 N
## 9 321 Mia   <NA>    NA     NA     NA     NA    18 N
## 10 669 Luana <NA>   NA     NA     NA     NA    19 S
## 11 967 André <NA>   NA     NA     NA     NA    20 N

```

```

# Inner join = intersecção.
inner_join(df1,
df_extra,
by = c("mat" = "mat",
"nome"))

```

```

## # A tibble: 4 x 9
##   mat nome  curso prova1 prova2 prova3 faltas idade bolsista
##   <dbl> <chr> <chr>  <dbl>  <dbl>  <dbl>  <dbl> <dbl> <chr>
## 1 256 João  Mat     80     90     80      4    18 S
## 2 965 Tiago Est    95     80     75      0    18 N
## 3 125 Luana Est    70     85     50      8    21 S
## 4 874 Pedro Mat    55     75     90      0    19 N

```

```

# Todos os que estão na 1º tabela
left_join(df1, df_extra,
by = c("mat" = "mat",
"nome"))

```

```

## # A tibble: 7 x 9
##   mat nome  curso prova1 prova2 prova3 faltas idade bolsista
##   <dbl> <chr> <chr>  <dbl>  <dbl>  <dbl>  <dbl> <dbl> <chr>
## 1 256 João  Mat     80     90     80      4    18 S
## 2 487 Vanessa Mat    75     75     75      4    NA <NA>
## 3 965 Tiago  Est    95     80     75      0    18 N
## 4 125 Luana  Est    70     85     50      8    21 S
## 5 458 Gisele  Est    45     50     NA     16    NA <NA>
## 6 874 Pedro  Mat    55     75     90      0    19 N
## 7 963 André  Est    30     NA     30     20    NA <NA>

```

```

# Todos os que estão na 2º tabela
right_join(df1, df_extra,
by = c("mat" = "mat",
"nome"))

```

```

## # A tibble: 8 x 9
##   mat nome  curso prova1 prova2 prova3 faltas idade bolsista
##   <dbl> <chr> <chr>  <dbl>  <dbl>  <dbl> <dbl> <dbl> <chr>
## 1   256 João   Mat      80     90     80     4    18 S
## 2   965 Tiago  Est      95     80     75     0    18 N
## 3   125 Luana  Est      70     85     50     8    21 S
## 4   874 Pedro  Mat      55     75     90     0    19 N
## 5   285 Tiago <NA>     NA     NA     NA     NA    22 N
## 6   321 Mia   <NA>     NA     NA     NA     NA    18 N
## 7   669 Luana <NA>     NA     NA     NA     NA    19 S
## 8   967 André  <NA>     NA     NA     NA     NA    20 N

# Os da 2º que não aparecem na 1º.
anti_join(df1, df_extra,
by = c("mat" = "mat",
"nome"))

```

```

## # A tibble: 3 x 7
##   mat nome  curso prova1 prova2 prova3 faltas
##   <dbl> <chr> <chr>  <dbl>  <dbl>  <dbl> 
## 1   487 Vanessa Mat      75     75     75     4
## 2   458 Gisele  Est      45     50     NA     16
## 3   963 André   Est      30     NA     30     20

```

## Exportação de Dados

```
#write_csv(df1,
          #file = "Nome_do_arquivo.csv")
```

## Exportando arquivos em texto pleno

```
#save(df1,
      #file = "Nome_do_arquivo.RData")
## Carregando arquivo .RData
#load("Nome_do_arquivo.RData")
```

## Arquivo binário do R

```
#library(writexl)
#write_xlsx(df1, "Nome_do_arquivo.xlsx")
```

## Criando planilha eletronica