ANÁLISE DOS DADOS COM R

DIY Funções

James R. Hunter, PhD

Retrovirologia, EPM, UNIFESP

FUNÇÕES - ALÉM DO QUE R PODE FAZER

FUNÇÕES EM R

- Já conhecemos funções que R fornece
- Funções simples matemáticas
  - sqrt(), max(), mean()
- Funções que ajuda manipular dados e conjuntos de dados
  - Manipulação
    - arrange(), select(), left_join()
  - Conjuntos de dados
    - read_excel(), saveRDS(), read_csv()

COMO FUNCIONA AS FUNÇÕES INTERNAS DE R

- São scripts de R que executam os comandos necessárias para o resultado
- Funções podem chamar outras funções escritas em outras linguagens (e.g., C, FORTRAN)
- Exemplo: `chisq_test()` – executa o teste estatístico χ^2 em 2 vetores
- Definição

```
chisq_test(x, y = NULL, correct = TRUE, p = rep(1/length(x), length(x)),
  rescale.p = FALSE, simulate.p.value = FALSE, B = 2000)
```

```
set.seed(42)
x <- sample(1:500, 200, replace = TRUE)
y <- sample(500:1000, 200, replace = TRUE)

chisq.test(x, y)
```

```
    Pearson's Chi-squared test

data:  x and y
X-squared = 26533, df = 26400, p-value = 0.2802
```

ABAIXO DO PANO

```
> chisq_test
function (x, y = NULL, correct = TRUE, p = rep(1/length(x), length(x)),
    rescale.p = FALSE, simulate.p.value = FALSE, B = 2000)
{
    args ← as.list(environment()) %>% add_item(method = "chisq_test")
    if (is.data.frame(x))
        x ← as.matrix(x)
    if (inherits(x, c("matrix", "table")))
        n ← sum(x)
    else n ← length(x)
    res.chisq ← stats::chisq.test(x, y, correct = correct, p = p,
        rescale.p = rescale.p, simulate.p.value = simulate.p.value,
        B = B)
    as_tidy_stat(res.chisq, stat.method = "Chi-square test") %>%
        add_significance("p") %>% add_columns(n = n, .before = 1) %>%
        set_attrs(args = args, test = res.chisq) %>% add_class(c("rstatix_test",
        "chisq_test"))
}
<bytecode: 0x11cdc1588>
<environment: namespace:rstatix>
```

QUANDO DEVE ESCREVER SUA PRÓPRIA FUNÇÃO

- Temos tendência de copiar/colar blocos de código que queremos utilizar
  - Trocando um elemento, nome de variável, paramêtro, etc.
- Esta prática induz erros
  - Se for na programação (R levanta um erro), precisa fazer debugging
  - Pode ser no resultado – você consegue perceber que o resultado do cálculo é errado
    - Perda de controle

Important

Se vai cópiar um bloco de código mais de **duas vezes**, trocar ele para uma função

UM EXEMPLO[1]

- Quero que as variáveis de um tibble ficam no intervalo entre 0 e 1
  - Mas recebo os valores numa outra escala
- A mesma coisa que a função `scales::rescale()` faz

```
set.seed(42)
data <- tibble(
  a = rnorm(5),
  b = rnorm(5),
  c = rnorm(5),
  d = rnorm(5),
)
data
```

```
# A tibble: 5 × 4
       a       b      c      d
   <dbl>   <dbl>  <dbl>  <dbl>
1  1.37  -0.106   1.30   0.636
2 -0.565  1.51    2.29  -0.284
3  0.363 -0.0947 -1.39  -2.66
4  0.633  2.02   -0.279 -2.44
5  0.404 -0.0627 -0.133  1.32
```

PARA CALCULAR A NOVA ESCALA PARA VARIÁVEL a

- Para caso i de variável a

$$\frac{a_i - \min(a)}{\max(a) - \min(a)}$$

```
1  nova_a <- data |>
2    mutate(a = (a - min(a, na.rm = TRUE)) /
3      (max(a, na.rm = TRUE) - min(a, na.rm = TRUE))) |>
4    pull(a)
5  nova_a
```

```
[1] 1.0000000 0.0000000 0.4793343 0.6186845 0.5005880
```

CÓDIGO - UM POUCO MAIS EFICIENTE

```r
range_a <- range(data$a, na.rm = TRUE)
range_a
```

```
[1] -0.5646982  1.3709584
```

```r
nova_a <- data |>
  mutate(a = (a - range_a[1]) /
    (range_a[2] - range_a[1])) |>
  pull(a)
nova_a
```

```
[1] 1.0000000 0.0000000 0.4793343 0.6186845 0.5005880
```

UMA SOLUÇÃO

- Obter com copiar-colar

```r
nova_data <- data |>
  mutate(
  a = (a - min(a, na.rm = TRUE)) /
    (max(a, na.rm = TRUE) - min(a, na.rm = TRUE)),
  b = (b - min(b, na.rm = TRUE)) /
    (max(b, na.rm = TRUE) - min(a, na.rm = TRUE)),
  c = (c - min(c, na.rm = TRUE)) /
    (max(c, na.rm = TRUE) - min(c, na.rm = TRUE)),
  d = (d - min(d, na.rm = TRUE)) /
    (max(d, na.rm = TRUE) - min(d, na.rm = TRUE)),
)
nova_data
```

```
# A tibble: 5 × 4
      a       b     c      d
  <dbl>   <dbl> <dbl>  <dbl>
1 1     0       0.733 0.828
2 0     0.801   1     0.597
3 0.479 0.00568 0     0
4 0.619 1.05    0.302 0.0543
5 0.501 0.0215  0.342 1
```

> **Warning**
>
> Todos os valores entre 0 e 1? cada coluna tem um 0 e um 1?

LIVE CODING

Vamos ao Posit Cloud

ESTUDAR A COMPUTAÇÃO

- *O que quer fazer?*
  - Criar uma nova escala para uma variável
- *O que é o cálculo que quer empenha?*
  - `(a - min(a, na.rm = TRUE)) / (max(a, na.rm = TRUE) - min(a, na.rm = TRUE))`
  - O que varia nesta expressão?
  - Essa vai ser o argumento que nós vamos dar para função
  - Pode dar para ele um nome abstrato, vamos dizer `v` (variável)
  - `(v - min(v, na.rm = TRUE)) / (max(v, na.rm = TRUE) - min(v, na.rm = TRUE))`

4 ELEMENTOS DE UMA FUNÇÃO

- **Nome**: como vamos chamar esta função (`rescale1`)
- **Keyword**: `function()`
- **Argumento(s)**: valores que variam quando a função é chamada (`x`)
- **Corpo**: o código que executa o cálculo ## O Que Estamos Tentando Fazer com a Função
- O cálculo que colocamos dentro da função

```
1  (a - min(a, na.rm = TRUE)) / (max(a, na.rm = TRUE) - min(a, na.rm = TRUE))
```

- Cada vez que tem um a, sabemos que esse vai ser aplicado a todas as variáveis a - d
- Cada vez que chamamos a função, vai ter outra variável no lugar de a

TEMPLATE PARA UMA FUNÇÃO

```
nome <- function(argumentos){
  corpo
  }
```

FUNÇÃO NOSSA

- Chamar o **argumento** v (para variável)

```
rescale1 <- function(v) {
  (v - min(v, na.rm = TRUE)) / (max(v, na.rm = TRUE) - min(v, na.rm = TRUE)
}
```

ANTES DE APLICAR - TESTAR

- Sempre queremos testar nossas funções para ver que estão produzindo resultados corretos
- Vamos dar para funções vários vetores para ver se faz certo

```
1 rescale1(c(-10, 0 , 10))
```

```
[1] 0.0 0.5 1.0
```

```
1 rescale1(c(1, 2, 3, 4, 5))
```

```
[1] 0.00 0.25 0.50 0.75 1.00
```

```
1 rescale1(c(1, 2, 3, 4, 101))
```

```
[1] 0.00 0.01 0.02 0.03 1.00
```

SUBSTITUIR A FUNÇÃO NO CÓDIGO ORIGINAL - VELHO

```r
nova_data <- data |>
  mutate(
  a = (a - min(a, na.rm = TRUE)) /
    (max(a, na.rm = TRUE) - min(a, na.rm = TRUE)),
  b = (b - min(b, na.rm = TRUE)) /
    (max(b, na.rm = TRUE) - min(b, na.rm = TRUE)),
  c = (c - min(c, na.rm = TRUE)) /
    (max(c, na.rm = TRUE) - min(c, na.rm = TRUE)),
  d = (d - min(d, na.rm = TRUE)) /
    (max(d, na.rm = TRUE) - min(d, na.rm = TRUE)),
)
nova_data
```

```
# A tibble: 5 × 4
      a       b     c      d
  <dbl>   <dbl> <dbl>  <dbl>
1 1     0       0.733 0.828
2 0     0.761   1     0.597
3 0.479 0.00540 0     0
4 0.619 1       0.302 0.0543
5 0.501 0.0204  0.342 1
```

COM FUNÇÃO

```
nova_data2 <- data |>
  mutate(
  a = rescale1(a),
  b = rescale1(b),
  c = rescale1(c),
  d = rescale1(d),
  )
nova_data2
```

```
# A tibble: 5 × 4
      a       b     c      d
  <dbl>   <dbl> <dbl>  <dbl>
1 1     0       0.733 0.828
2 0     0.761   1     0.597
3 0.479 0.00540 0     0
4 0.619 1       0.302 0.0543
5 0.501 0.0204  0.342 1
```

CRIAR UMA FUNÇÃO QUE FALTA EM R

- e.g., Coeficiente de Variação - mede a variabilidade de uma distribuição em relação à média

$$CV = \frac{\sigma}{\mu}$$

- A relação de desvio padrão à média – maior a relação, maior a variabilidade
- CV maior que 1 normalmente indica um problema com a alta variabilidade da distribuição
- Existe em vários pacotes, mas não em Base R
- Se queremos usar ele dentro de `dplyr::summarize()`, precisamos criar uma função

CRIAR A FUNÇÃO `cv`

O que a função vai fazer?

- Recebe um vetor dos dados (argumento)
  - Pode ser no formato de uma variável de um data frame
- Calcular a média (`mean()`) e desvio padrão (`sd()`) do vetor
- Retornar o valor da divisão do desvio padrão pela média

PROGRAMAÇÃO - "PSEUDO-CODE"

```r
set.seed(42)
v <- sample(1:100, 10, replace = TRUE) # criar um vetor

s <-  sd(v, na.rm = TRUE)
print(paste0("s = ", round(s, 3)))
```

```
[1] "s = 25.883"
```

```r
m <-  mean(v, na.rm = TRUE)
print(paste0("m = ", round(m, 3)))
```

```
[1] "m = 52.2"
```

```r
cv <- s/m
print(paste0("cv = ", round(cv, 4)))
```

```
[1] "cv = 0.4959"
```

PROGRAMAÇÃO - A FUNÇÃO

- Nome: `cv` (qualquer coisa que queremos)
- Argumento: um vetor
  - Para testes, vamos continuar com o vetor `v` que já definimos

```
cv <- function(v){
  ...
}
```

```
cv <- function(v){
  m <- mean(v, na.rm = TRUE)
  s <- sd(v, na.rm = TRUE)
  cv <- s/m
  return(cv)
}
```

Testar a função com nosso vetor `v`, que vamos renomear para `vec`

```
vec <- v

cv(vec)
```

```
[1] 0.4958525
```

PODEMOS FAZER A FUNÇÃO MAIS EFICIENTE

- Pode combinar os cálculos em 1 linha
- R automaticamente retorna ao ambiente superior o resultado da última linha - não precisa uma linha de `return()`

```
cv_old <- function(v){
  m <- mean(v, na.rm = TRUE)
  s <- sd(v, na.rm = TRUE)
  cv <- s/m
  return(cv)
}

cv <- function(v){
  sd(v, na.rm = TRUE) / mean(v, na.rm = TRUE)
}

(cv(vec))
```

TESTAR A FUNÇÃO

```
cv(c(-100, 0, 200))
```

```
[1] 4.582576
```

```
cv(runif(rnorm(100, mean = 10, sd = 3.14)))
```

```
[1] 0.6728386
```

```
cv(c(-100, 0, 200, NA))
```

```
[1] 4.582576
```

PODEMOS USAR DENTRO DE

dplyr::summarize()

```r
data |>
  summarise(cv_a = cv(a),
            cv_b = cv(b),
            cv_c = cv(c),
            cv_d = cv(d))
```

```
# A tibble: 1 × 4
   cv_a  cv_b  cv_c  cv_d
  <dbl> <dbl> <dbl> <dbl>
1  1.57  1.58  4.03 -2.62
```

> ⚠ **Warning**
>
> Pode cv ser negativo como cv_d? Explique porque

FUNÇÕES QUE USAM VERBOS DE *TIDYVERSE*

- Vamos tentar usar verbos de *tidyverse* dentro de uma função
- Dados vêm de um estudo de 2022 de Prof. Reinaldo Salamão sobre casos clínicos de COVID-19
- Só alguns dos dados

```r
covid <- readRDS(here("mad_covid_data.rds"))
head(covid, 10)
```

```
# A tibble: 10 × 7
   id    idoso     death    gender   age lymphocytes   crp
   <fct> <chr>     <chr>    <fct>  <dbl>       <dbl> <dbl>
 1 3     nao_idoso survival m       60           480  NA
 2 4     idoso     survival m       62.7          NA  NA
 3 5     nao_idoso death    m       59.8         527  NA
 4 6     nao_idoso survival f       44          1675  20.8
 5 7     nao_idoso survival m       21.6        1465  NA
 6 8     idoso     survival m       65.8        1391  66.6
 7 9     idoso     survival m       64.5          NA  NA
 8 10    nao_idoso survival f       54.7        1212  81.8
 9 11    nao_idoso survival f       58.7          NA  NA
10 12    idoso     death    m       63.4         373  87.4
```

SE SÓ VAMOS FAZER ÚNICO CÁLCULO COM `covid`

```
covid |>
    group_by(death) |>
    summarise(mean(crp, na.rm = TRUE))
```

```
# A tibble: 2 × 2
  death    `mean(crp, na.rm = TRUE)`
  <chr>                        <dbl>
1 death                        162.
2 survival                      72.5
```

FUNÇÃO

- Desenvolver função que mede a média de qualquer uma das variáveis quantitativas agrupada por uma das variáveis categóricas
- Agrupamento: verbo `group_by()` : `group_by(death)`
- Determinar a média: mean() - lembre a usar o argumento `na.rm=TRUE` porque existem, sim

```
gr_media <- function(grp_var, calc_var){
  covid |>
    group_by(grp_var) |>
    summarise(mean(calc_var, na.rm = TRUE))
}
```

TESTAR A FUNÇÃO

```
1 gr_media(death, crp)
```

```
1  {r sars_3}
2
3  gr_media(death, crp)
```

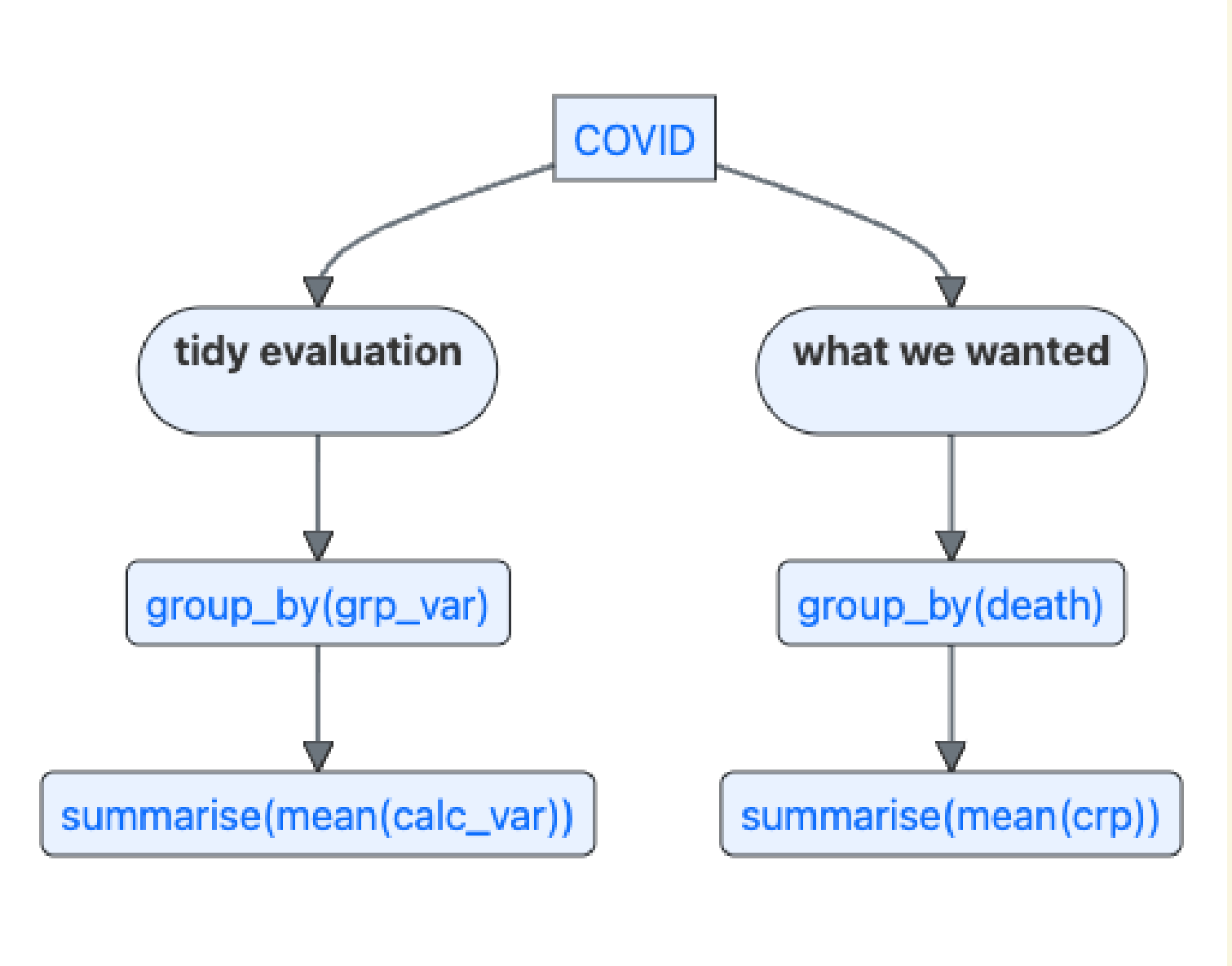
```
Error in `group_by()`:
! Must group by variables found in `.data`.
✖ Column `grp_var` is not found.
Backtrace:
 1. global gr_media(death, crp)
 4. dplyr:::group_by.data.frame(covid, grp_var)
```

```
Error in group_by(covid, grp_var) :
  ✖ Column `grp_var` is not found.
```

???? - TIDY EVALUATION

- O que acontece quando uma variável é o argumento de uma função que fica **dentro** de uma outra função
- Problema vem porque os comandos de *tidyverse* sempre usam "*tidy evaluation*"
- *tidy evaluation* permite referência aos nomes das varíaveis de um tibble sem tratamento special
- Funções que passam nomes de colunas (variáveis) de tibbles para um verbo de `dplyr` usam *tidy evaluation*
- `cv` está passando `death` e `crp` para verbos de `dplyr`
- `cv` vai tentar explicitamente avaliar `grp_var` como se fosse uma variável de um tibble
  - Mas, é um placeholder que seria substituído por `death` na função
  - Semelhente com `calc_var` e `crp`

DIAGRAMA FACILITA ENTENDIMENTO



ABRAÇANDO A VARIÁVEL

- Deve ter maneira para avisar `group_by()` e `summarise()` para não tratar `group_var` e `mean_var` como os nomes das variáveis
  - Invés disso, R deve olhar dentro destas variáveis para achar a variável que queremos usar
- Fazemos isso com **o abraço**, abraçando a variável com dupla chaves – {{ var }}
  - `{ grp_var }`
- Assim, R olhe dentro da `grp_var` para achar a variável que realmente está em jogo.
  - Diz: "Oi, `grp_var` é um placeholder. Olhe nos argumentos da chamada para a variável verdadeiro"
    - Que é `death`

CV COM ABRAÇO

```r
gr_media <- function(grp_var, calc_var){
  covid |>
    group_by({{ grp_var }}) |>
    summarise(mean_by_group = mean({{ calc_var }}, na.rm = TRUE))
}

gr_media(death, crp)
```

```
# A tibble: 2 × 2
  death    mean_by_group
  <chr>            <dbl>
1 death             162.
2 survival           72.5
```

FUNÇÕES ANÔNIMAS

- Voltamos a 1º conjunto hoje: `data` com 4 variáveis `a` - `d`
- Podemos aplicar a função `cv` para todas as variáveis em 1 linha com `across()`
- Se o que queremos é uma pequena função de poucas palavras,
  - Podemos usar uma função anônima - uma função sem nome

```
data |>
  mutate(across(a:d, function(x) cv(x)))
```

```
# A tibble: 5 × 4
      a     b     c     d
  <dbl> <dbl> <dbl> <dbl>
1  1.57  1.58  4.03 -2.62
2  1.57  1.58  4.03 -2.62
3  1.57  1.58  4.03 -2.62
4  1.57  1.58  4.03 -2.62
5  1.57  1.58  4.03 -2.62
```

```
data |>
  mutate(across(a:d, \(x) cv(x)))
```

```
# A tibble: 5 × 4
      a     b     c     d
  <dbl> <dbl> <dbl> <dbl>
1  1.57  1.58  4.03 -2.62
2  1.57  1.58  4.03 -2.62
3  1.57  1.58  4.03 -2.62
4  1.57  1.58  4.03 -2.62
5  1.57  1.58  4.03 -2.62
```

QUANDO DEVEMOS USAR FUNÇÕES

APLICAR FUNÇÃO ÀS VÁRIAS COLUNAS

- Introdução a programação funcional

3 FUNÇÕES E FAMÍLIAS DAS FUNÇÕES

- `apply` - família
- `across()` - `dplyr`
- `purrr::map_x()` - família

FAMÍLIA `apply`

- `apply()`, `lapply()`, `sapply()`, e `tapply()`
- Base R
- Aprender `apply()`
- Dados: `pacdemo`

```r
pacdemo |>
  slice(1:4) |>
  gt()
```

sexo	idade	cv	cd4	cd8
Masculino	60	5200	898	1311
Masculino	73	1947	958	817
Feminino	51	480000	958	817
Masculino	50	257313	142	1009

MÉDIA DAS VARIÁVEIS NÚMERICAS

- Pode fazer uma lista dos cálculos

```
(med_idade <- mean(pacdemo$idade))
```

```
[1] 51.2
```

```
(med_cv <- mean(pacdemo$cv))
```

```
[1] 90692.82
```

```
(med_cd4 <- mean(pacdemo$cd4))
```

```
[1] 513.9
```

```
(med_cd8 <- mean(pacdemo$cd8))
```

```
[1] 994.3
```

- Lembre a regra de não copiar/colar mais de 2 vezes
- Pode fazer um função que vai de coluna em coluna

`apply()`

- Template:

```
apply(X, MARGIN, FUN, ...)
```

- `X`: o objeto que estamos avaliando (`pacdemo`)
- `MARGIN`: colunas (2) ou fileiras (1)
- `FUN`: função a ser aplicada (sem parênteses)

SOMA DOS VALORES NÚMERICOS

```r
apply(pacdemo[,2:5], 2, sum)
```

```
idade      cv     cd4     cd8
 2560 4534641   25695   49715
```

MÉDIA DOS VALORES NÚMERICOS

- Levando em conta que pode ter um valor NA
- `mean(x, na.rm = TRUE)`
- Os `...` no template

```
1 apply(pacdemo[,2:5], 2, mean, na.rm = TRUE)
```

```
   idade       cv      cd4      cd8
   51.20 90692.82   513.90   994.30
```

TIDYVERSE - `dplyr::across()`

- Permite que você fazer um cálculo ou operação em várias colunas
  - Como `apply`, mas mais flexível
  - Facilita trabalho com grupos e com colunas na mesma operação
  - Como apply, expressa a função dentro da função `across()`
  - Pode ser aplicada a uma variedade das colunas de um conjunto grande
  - Funções anônimas
    - Equivalente em R para funções lambda (λ) em Python e outras idiomas

CASO MAIS SIMPLES

- Mesma operação que com `apply()`
- Template:

```
across(.cols, .fns, ...)
```

```
1 test <- pacdemo |>
2   summarise(sum = across(idade:cd8, sum),
3             mean = across(idade:cd8, mean))
4 test
```

```
# A tibble: 1 × 2
  sum$idade     $cv  $cd4  $cd8 mean$idade    $cv  $cd4  $cd8
      <dbl>   <dbl> <dbl> <dbl>       <dbl>  <dbl> <dbl> <dbl>
1      2560 4534641 25695 49715        51.2 90693.  514.  994.
```

ESTRUTURA DE `test`

```
1  str(test)
```

```
tibble [1 × 2] (S3: tbl_df/tbl/data.frame)
 $ sum : tibble [1 × 4] (S3: tbl_df/tbl/data.frame)
  ..$ idade: num 2560
  ..$ cv   : num 4534641
  ..$ cd4  : num 25695
  ..$ cd8  : num 49715
 $ mean: tibble [1 × 4] (S3: tbl_df/tbl/data.frame)
  ..$ idade: num 51.2
  ..$ cv   : num 90693
  ..$ cd4  : num 514
  ..$ cd8  : num 994
```

- *Tibbles*, *tibbles* em todos os lugares!
- `across()` retorna um *tibble* com una coluna por
  - Cada coluna em `.cols`
  - Cada função em `.funs`

OUTRA MANEIRA DE MOSTRA AS DUAS FUNÇÕES

```
pac_summary <- pacdemo |>
 summarise(res = across(idade:cd8, c(sum, mean)))
pac_summary$res
```

```
# A tibble: 1 × 8
  idade_1 idade_2    cv_1   cv_2 cd4_1 cd4_2 cd8_1 cd8_2
    <dbl>   <dbl>   <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl>
1    2560    51.2 4534641 90693. 25695  514. 49715  994.
```

FUNÇÃO ANÔNIMA

- Anônima porque não tem nome
- Função que pode ser jogado no lixo depois de uso
- `\(x)` seguido pelo código da função
  - `\(x) x + 1` - aumentar o valor de `x` por 1
  - `\(x) median(x, na.rm = TRUE)`
- Frequentemente usada com `across()`

APLICAR PARA `pacdemo summarise`

- Calcular média das colunas númericas
  - Cuidando de chance de ter NA

```
pac_summary <- pacdemo |>
 summarise(res = across(idade:cd8, \(x) mean(x, na.rm = TRUE)))
pac_summary$res
```

```
# A tibble: 1 × 4
  idade     cv   cd4   cd8
  <dbl>  <dbl> <dbl> <dbl>
1  51.2 90693.  514.  994.
```

FAMÍLIA `purrr::map_()`

- Pacote `purrr` desenhado para fornecer funções para programação funcional
- purrr::map() transforma cada elemento de uma lista ou vetor por aplicacão de uma função
  - Retorna uma lista
- `map_lgl()`, `map_int()`, `map_dbl()`, e `map_chr()` retornam um vetor do tipo indicado

```r
# work with penguins

n_unique <- function(x) length(unique(x))

penguin |>
  select(species, island, sex, ) |>
  map(n_unique)
```

```
$species
[1] 3

$island
[1] 3

$sex
[1] 3
```

```r
penguin |>
  select(species, island, sex, ) |>
  map_int(n_unique)
```

```
species  island     sex
      3       3       3
```

```r
penguin |>
  select(flipper_length_mm, body_mass_g, bill_length_mm ) |>
  map_dbl(\(x) mean(x, na.rm = TRUE))
```

```
flipper_length_mm       body_mass_g    bill_length_mm
        200.91520         4201.75439           43.92193
```

SÓ O INÍCIO COM

purrr

MUITO MAIS SOBRE FUNÇÕES

- Só o início sobre funções
- Precisa estudar a documentação, esp. R4DS