# R para Ciência de Dados 2

## NSE


Agosto de 2021

# Motivação

Se o `{tidyverse}` é o conjunto mais incrível de pacotes, faz sentido querermos criar nossos pacotes e fazer nossas análises usando seus princípios. Funções cujo primeiro argumento é um *data frame* e operações "pipeáveis" são apenas o começo.

Uma das características mais marcantes do `{tidyverse}` é a possibilidade de trabalhar com colunas como se elas fossem objetos comuns, criados fora da função. Basta pensar na `mutate()`: como ela sabe que `mpg` é uma coluna da tabela e não um objeto externo com o mesmo nome?

Existe uma ferramenta especial (e exclusiva ao R!) que nos permite fazer esse tipo de mágica: a *non-standard evaluation* (NSE) ou, em português, a avaliação não-padrão. Usamos NSE desde a primeira vez que escrevemos o comando `library()`, mas o seu funcionamento ainda é um mistério para a maioria.

Para criar funções flexíveis e exutas como as do `{tidyverse}`, é necessário entender o básico da NSE e de como trabalhar com alguns operadores desse novo mundo.

# Introdução

*Non-standard evaluation* é uma propriedade do R que permite capturar o seu código sem avaliá-lo. Isso mesmo: o R deixa de ser uma caixa para a qual nós enviamos comandos a serem executados e passa a ter a capacidade de **interpretar e trabalhar** os próprios comandos. Genericamente isso se chama **metaprogramação**.

```
# O objeto 'dplyr' simplesmente não existe...
dplyr
```

```
#> Error in eval(expr, envir, enclos): object 'dplyr' not found
```

```
# Por que então esse comando funciona?
library(dplyr)
```

Na chamada `library()` acima, a palavra "dplyr" não se refere a nenhum objeto! O R consegue ler o código que nós escrevemos e agir em cima dessa informação. Apesar de ser comum para nós, outras linguagens são incapazes de fazer isso por causa da forma com que elas tratam seus argumentos.

O R tem o que chamamos de avaliação tardia (*delayed evaluation*), ou seja, uma expressão só é avaliada quando ela é necessária e não logo que ela é criada. No R, `print(1 + 2)` é diferente de `print(3)`, mas nas outras linguagens isso não é o caso!

# Tidy evaluation

A faceta da NSE que nos interessa no momento é a chamada *tidy evaluation*, ou **tidy eval**, a avaliação não-padrão utilizada pelas funções do `{tidyverse}` e outros pacotes feitos para trabalhar com o mesmo paradigma.

- O mundo sem tidy eval é extremamente verborrágico, pois a tabela precisa ser especificada toda vez que nos referirmos a uma coluna:

```
starwars[starwars$homeworld == "Naboo" & starwars$species == "Human", ,]
```

- Para não precisar do `$`, a nossa única saída é criar objetos com as colunas:

```
homeworld <- starwars$homeworld; species <- starwars$species
starwars[homeworld == "Naboo" & species == "Human", ,]
```

- O `{tidyverse}` permite criar um "mini ambiente" em que as colunas da tabela estão disponíveis como se fossem objetos declarados explicitamente:

```
filter(starwars, homeworld == "Naboo", species == "Human")
```

# Uma faca de dois gumes

- O problema de capturar o código sem avaliá-lo é que fica difícil avaliar algo antes que ele seja capturado.

```
# O código 'birth_year' é capturado
starwars %>% filter(is.na(birth_year)) %>% nrow()
```

```
#> [1] 44
```

```
# O objetivo é filtrar uma coluna especificada pelo usuário
filter_na <- function(df, col) {
  filter(df, is.na(col))
}

# A função captura o código 'col'
starwars %>% filter_na(col = birth_year) %>% nrow()
```

```
#> Error: Problem with `filter()` input `..1`.
#> ℹ Input `..1` is `is.na(col)`.
#> x object 'birth_year' not found
```

# Curly-curly

- O operador que vai resolver nosso problema é o `{{ }}` (lê-se *curly curly*), que permite interpolar o código, ou seja, avaliá-lo antes da captura.

```
# O objetivo é filtrar uma coluna especificada pelo usuário
filter_na <- function(df, col) {
  filter(df, is.na( {{col}} ))
}

# Agora a função captura o código 'birth_year'
starwars %>% filter_na(col = birth_year) %>% nrow()
```

```
#> [1] 44
```

- Essa sintaxe vem da interpolação de strings:

```
col <- "birth_year"
stringr::str_glue("Interpolando '{col}'!")
```

```
#> Interpolando 'birth_year'!
```

# Múltiplos argumentos

- Para passar múltiplos argumentos nem é necessário usar o curly-curly já que a reticência já possui as capacidades de tidy eval:

```
# O objetivo é permitir vários cálculos
summarise_by <- function(df, ..., by) {
  df %>%
    group_by( {{by}} ) %>%
    summarise(...)
}

starwars %>%
  summarise_by(
    media = mean(height, na.rm = TRUE),
    maximo = max(height, na.rm = TRUE),
    by = gender
  )
```

```
#> # A tibble: 3 × 3
#>   gender    media maximo
#>   <chr>     <dbl>  <int>
#> 1 feminine   165.    213
#> 2 masculine  177.    264
#> 3 <NA>       181.    183
```

Note como não houve necessidade de interpolar `media` e `maximo`: a `summarise()` não tentou criar uma coluna chamada `...`

# Strings

- E se quisermos passar strings para as funções do `{tidyverse}`? Se pedirmos o nome de uma coluna para um usuário, a resposta virá como string.

```
# O objetivo é dar um nome para a média
summarise_mean <- function(df, nome, col) {
  summarise(df, nome = mean(col, na.rm = TRUE))
}

# É criada uma coluna 'nome' sem valor ('col' não existe)
summarise_mean(starwars, "media", "height")
```

```
#> # A tibble: 1 × 1
#>    nome
#>   <dbl>
#> 1    NA
```

# Strings na esquerda

- Quando o "lado esquerdo" (**antes de um igual**) de uma expressão com tidy eval é uma string (ou se tornará uma quando avaliado), precisamos apenas usar o operador `:=` (lê-se *walrus*, "morsa"):

```
# O objetivo é dar um nome para a média
summarise_mean <- function(df, nome, col) {
  summarise(df, {{nome}} := mean(col, na.rm = TRUE))
}

# É criada uma coluna 'media' sem valor ('col' não existe)
summarise_mean(starwars, "media", "height")
```

```
#> # A tibble: 1 × 1
#>   media
#>   <dbl>
#> 1    NA
```

# Strings na direita

- Quando uma string (ou algo que se tornará uma quando avaliado) está no "lado direito" (**depois de um igual ou quando não há igual**) de uma expressão com tidy eval, precisamos apenas usar o pronome `.data`:

```
# O objetivo é dar um nome para a média
summarise_mean <- function(df, nome, col) {
  summarise(df, {{nome}} := mean(.data[[col]], na.rm = TRUE))
}

# É criada uma coluna 'media' com a média de 'height'
summarise_mean(starwars, "media", "height")
```

```
#> # A tibble: 1 × 1
#>   media
#>   <dbl>
#> 1  174.
```

- É como se estivéssemos chamando `df[["height"]]`, mas, como `df` não faria sentido dentro da expressão, usamos `.data`.

# Referências

- Para saber mais sobre NSE, tidy eval e metaprogramação, dê uma olhada nos materiais disponibilizados pelo time `{rlang}`:
  - rlang 0.4.0
  - Tidy eval now supports glue strings
  - Programming with dplyr
  - Metaprogramming
  - Tutorial: {rlang} para Filhotes