Dia 2

Lógica, Funções e Introdução ao Tidyverse

Vinícius Silva Junqueira

2025-11-03

Sumário

1 Lógica, Funções e Introdução ao Tidyverse

Objetivos do dia

- Dominar operadores lógicos/relacionais e condicionais (`if`, `ifelse`, `case_when`).
- Entender loops vs. vetorização e criar **funções próprias**.
- Aplicar um **pipeline básico** com `dplyr` e introduzir **datas** com `lubridate`.
- Registrar o aprendizado com um commit no seu fork no GitHub.

Tempo previsto 19h00–22h00 (intervalo 20h30–20h50)

---

2 Operadores e Condicionais (30 min)

2.1 O que são operadores?

Operadores são símbolos especiais que realizam operações entre valores. Eles são fundamentais para tomar decisões no código e controlar o fluxo de execução.

2.2 Operadores lógicos e relacionais

Operadores relacionais comparam dois valores e retornam `TRUE` ou `FALSE`:

- `==` : igual a
- `!=` : diferente de
- `>` : maior que
- `<` : menor que
- `>=` : maior ou igual
- `<=` : menor ou igual

Operadores lógicos combinam condições:

- `&` : E (AND) - ambas condições devem ser verdadeiras
- `|` : OU (OR) - pelo menos uma condição deve ser verdadeira
- `!` : NÃO (NOT) - inverte o valor lógico
- `xor()` : OU EXCLUSIVO - apenas uma condição pode ser verdadeira

```
# ==========================================
# OPERADORES RELACIONAIS EM MELHORAMENTO
# ==========================================

# Avaliação de um reprodutor
dep_leite <- 850  # DEP (Diferença Esperada na Progênie) para leite em kg
acuracia <- 0.85  # Acurácia da avaliação genética
idade_meses <- 36
```

```r
# Verificações básicas
dep_leite > 500        # TRUE - DEP positiva e alta?
```

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

```r
acuracia >= 0.70       # TRUE - avaliação confiável?
```

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

```r
idade_meses <= 48      # TRUE - ainda jovem?
```

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

```r
dep_leite == 850       # TRUE - exatamente 850 kg?
```

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

```r
acuracia != 1.0        # TRUE - não tem acurácia perfeita
```

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

```r
# Avaliando múltiplos animais
deps_peso <- c(15, 22, 8, 30, 18)  # DEPs para peso ao desmame (kg)
deps_peso >= 20        # FALSE FALSE FALSE TRUE FALSE
```

```
#> [1] FALSE  TRUE FALSE  TRUE FALSE
```

```r
sum(deps_peso >= 20)   # 2 animais atingem o critério
```

```
#> [1] 2
```

```r
# ==========================================
# OPERADORES LÓGICOS - SELEÇÃO DE ANIMAIS
# ==========================================

# Critérios para touro elite
dep_leite <- 950
dep_gordura <- 35
dep_proteina <- 28
confiabilidade <- 0.88
consanguinidade <- 0.03

# Touro elite: DEP leite > 800 E gordura > 30 E proteína > 25
touro_elite <- dep_leite > 800 & dep_gordura > 30 & dep_proteina > 25
print(touro_elite)  # TRUE
```

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

```r
# Touro elite CONFIÁVEL: critérios acima E confiabilidade alta E baixa consanguinidade
touro_elite_confiavel <- touro_elite & confiabilidade >= 0.80 & consanguinidade < 0.05
print(touro_elite_confiavel)  # TRUE
```

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

```r
# Critérios para descarte de fêmea
idade_anos <- 8
producao_ultima_lactacao <- 4500  # kg de leite
intervalo_partos <- 450  # dias
problemas_reprodutivos <- TRUE

# Descartar se: idade > 7 OU produção < 5000 OU problemas reprodutivos
descartar <- idade_anos > 7 | producao_ultima_lactacao < 5000 | problemas_reprodutivos
print(descartar)  # TRUE (atende múltiplos critérios)
```

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

```r
# Seleção para inseminação artificial
fertilidade_campo <- 0.65
libido <- "alta"
saude_andrológica <- TRUE
dep_habilidade_materna <- 5

# Apto para IA se: fertilidade > 0.60 E libido adequada E saúde OK
apto_ia <- fertilidade_campo > 0.60 &
           (libido == "alta" | libido == "média") &
           saude_andrológica
print(apto_ia)  # TRUE
```

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

```r
# XOR - Apenas uma característica excepcional
dep_crescimento_excepcional <- TRUE  # DEP > 30 kg
dep_qualidade_carcaca_excepcional <- FALSE  # DEP carne < 2

# Animal especializado (apenas uma característica excepcional)
especializado <- xor(dep_crescimento_excepcional, dep_qualidade_carcaca_excepcional)
print(especializado)  # TRUE - é especializado em crescimento
```

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

```r
# Negação (NOT)
tem_defeito_genetico <- FALSE
!tem_defeito_genetico  # TRUE - não tem defeito, pode reproduzir
```

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

```r
# ==========================================
# EXEMPLO PRÁTICO: DATABASE DE REBANHO
# ==========================================

# Plantel de vacas leiteiras
rebanho <- data.frame(
  animal_id = c("V001", "V002", "V003", "V004", "V005"),
  dep_leite = c(850, 650, 920, 550, 780),
```

```
  dep_gordura = c(32, 28, 38, 25, 30),
  acuracia = c(0.85, 0.75, 0.90, 0.60, 0.80),
  idade_anos = c(4, 6, 3, 8, 5),
  consanguinidade = c(0.02, 0.06, 0.01, 0.08, 0.03),
  n_partos = c(2, 4, 1, 6, 3)
)

print(rebanho)
```

```
#>   animal_id dep_leite dep_gordura acuracia idade_anos consanguinidade n_partos
#> 1      V001       850          32     0.85          4             0.02        2
#> 2      V002       650          28     0.75          6             0.06        4
#> 3      V003       920          38     0.90          3             0.01        1
#> 4      V004       550          25     0.60          8             0.08        6
#> 5      V005       780          30     0.80          5             0.03        3
```

```
# 1. VACAS SUPERIORES para mães de touros
# Critérios: DEP leite > 800, DEP gordura > 30, acurácia > 0.80
maes_touros <- rebanho$dep_leite > 800 &
               rebanho$dep_gordura > 30 &
               rebanho$acuracia > 0.80

print("Candidatas a mães de touros:")
```

```
#> [1] "Candidatas a mães de touros:"
```

```
print(rebanho[maes_touros, ])
```

```
#>   animal_id dep_leite dep_gordura acuracia idade_anos consanguinidade n_partos
#> 1      V001       850          32     0.85          4             0.02        2
#> 3      V003       920          38     0.90          3             0.01        1
```

```
# 2. VACAS EM RISCO de descarte
# Critérios: idade > 7 OU consanguinidade > 0.07 OU DEP leite < 600
risco_descarte <- rebanho$idade_anos > 7 |
                  rebanho$consanguinidade > 0.07 |
                  rebanho$dep_leite < 600

print("\nAnimais em risco de descarte:")
```

```
#> [1] "\nAnimais em risco de descarte:"
```

```
print(rebanho[risco_descarte, ])
```

```
#>   animal_id dep_leite dep_gordura acuracia idade_anos consanguinidade n_partos
#> 4      V004       550          25      0.6          8             0.08        6
```

```
# 3. VACAS JOVENS COM ALTO POTENCIAL
# Critérios: idade <= 4 E DEP leite > 750 E acurácia > 0.75
jovens_potencial <- rebanho$idade_anos <= 4 &
                    rebanho$dep_leite > 750 &
```

```
                    rebanho$acuracia > 0.75

print("\nVacas jovens de alto potencial:")
```

```
#> [1] "\nVacas jovens de alto potencial:"
```

```
print(rebanho[jovens_potencial, ])
```

```
#>   animal_id dep_leite dep_gordura acuracia idade_anos consanguinidade n_partos
#> 1      V001       850          32     0.85          4            0.02        2
#> 3      V003       920          38     0.90          3            0.01        1
```

```
# 4. CONTROLE DE CONSANGUINIDADE
# Animais NÃO consanguíneos (para acasalamentos)
nao_consanguineos <- rebanho$consanguinidade < 0.05
print("\nAnimais não consanguíneos:")
```

```
#> [1] "\nAnimais não consanguíneos:"
```

```
print(rebanho[nao_consanguineos, c("animal_id", "consanguinidade")])
```

```
#>   animal_id consanguinidade
#> 1      V001            0.02
#> 3      V003            0.01
#> 5      V005            0.03
```

```
# ============================================
# EXEMPLO: DECISÕES DE ACASALAMENTO
# ============================================

# Características do touro
touro_dep_leite <- 900
touro_dep_gordura <- 35
touro_consanguinidade <- 0.02

# Características da vaca
vaca_dep_leite <- 700
vaca_dep_gordura <- 28
vaca_consanguinidade <- 0.03

# Compatibilidade genética
# Ambos devem ter DEPs positivas E consanguinidade combinada < 0.06
deps_complementares <- touro_dep_leite > 600 & vaca_dep_leite > 600 &
                       touro_dep_gordura > 25 & vaca_dep_gordura > 25

consanguinidade_aceitavel <- (touro_consanguinidade + vaca_consanguinidade) < 0.06

acasalamento_recomendado <- deps_complementares & consanguinidade_aceitavel
print(paste("Acasalamento recomendado?", acasalamento_recomendado))  # TRUE
```

```
#> [1] "Acasalamento recomendado? TRUE"
```

```r
# ==========================================
# ÍNDICE DE SELEÇÃO COMPOSTO
# ==========================================

# Cálculo de índice para gado de corte
dep_peso_desmame <- 25  # kg
dep_ganho_pos_desmame <- 18  # kg
dep_aol <- 2.5  # área de olho de lombo (cm²)
dep_eg <- -1.2  # espessura de gordura (mm) - negativo é bom!

# Pesos econômicos (exemplo)
peso_pd <- 2.0
peso_gpd <- 1.5
peso_aol <- 3.0
peso_eg <- -1.0

# Cálculo do índice
indice <- (dep_peso_desmame * peso_pd) +
          (dep_ganho_pos_desmame * peso_gpd) +
          (dep_aol * peso_aol) +
          (dep_eg * peso_eg)

print(paste("Índice de seleção:", round(indice, 2)))
```

```
#> [1] "Índice de seleção: 85.7"
```

```r
# Animal é TOP 10% se índice > 80
top_10_pct <- indice > 80
print(paste("Animal top 10%?", top_10_pct))
```

```
#> [1] "Animal top 10%? TRUE"
```

```r
# ==========================================
# EXERCÍCIO PRÁTICO
# ==========================================

# Você tem 5 touros para classificar:
touros <- data.frame(
  id = c("T001", "T002", "T003", "T004", "T005"),
  dep_leite = c(920, 780, 650, 850, 1050),
  dep_proteina = c(30, 25, 22, 28, 35),
  acuracia = c(0.88, 0.72, 0.65, 0.85, 0.92),
  filhas_avaliadas = c(85, 45, 30, 70, 120),
  consanguinidade = c(0.02, 0.05, 0.08, 0.03, 0.01)
)

# DESAFIO 1: Selecione touros ELITE
# Critérios: DEP leite > 900, proteína > 28, acurácia > 0.85, consanguinidade < 0.04
touros_elite <- touros$dep_leite > 900 &
```

```
                touros$dep_proteina > 28 &
                touros$acuracia > 0.85 &
                touros$consanguinidade < 0.04

print("TOUROS ELITE:")
```

```
#> [1] "TOUROS ELITE:"
```

```
print(touros[touros_elite, ])
```

```
#>     id dep_leite dep_proteina acuracia filhas_avaliadas consanguinidade
#> 1 T001       920           30     0.88               85            0.02
#> 5 T005      1050           35     0.92              120            0.01
```

```
# DESAFIO 2: Touros com avaliação CONFIÁVEL
# Critérios: acurácia > 0.80 OU filhas avaliadas >= 80
touros_confiaveis <- touros$acuracia > 0.80 | touros$filhas_avaliadas >= 80

print("\nTOUROS COM AVALIAÇÃO CONFIÁVEL:")
```

```
#> [1] "\nTOUROS COM AVALIAÇÃO CONFIÁVEL:"
```

```
print(touros[touros_confiaveis, ])
```

```
#>     id dep_leite dep_proteina acuracia filhas_avaliadas consanguinidade
#> 1 T001       920           30     0.88               85            0.02
#> 4 T004       850           28     0.85               70            0.03
#> 5 T005      1050           35     0.92              120            0.01
```

```
# DESAFIO 3: Touros para DESCARTE
# Critérios: consanguinidade > 0.06 OU (DEP leite < 700 E acurácia < 0.70)
touros_descarte <- touros$consanguinidade > 0.06 |
                   (touros$dep_leite < 700 & touros$acuracia < 0.70)

print("\nTOUROS PARA DESCARTE:")
```

```
#> [1] "\nTOUROS PARA DESCARTE:"
```

```
print(touros[touros_descarte, ])
```

```
#>     id dep_leite dep_proteina acuracia filhas_avaliadas consanguinidade
#> 3 T003       650           22     0.65               30            0.08
```

Operador especial: - %in% : verifica se um valor está presente em um vetor

```
# Lógicos: & | ! xor()
TRUE & FALSE    # FALSE (ambos precisam ser TRUE)
```

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

```
TRUE | FALSE    # TRUE (pelo menos um é TRUE)
```

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

```r
!TRUE          # FALSE (inverte)
```

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

```r
xor(TRUE, FALSE) # TRUE (apenas um é TRUE)
```

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

```r
# Relacionais: == != > < >= <=
3 == 3         # TRUE (igual)
```

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

```r
5 != 2         # TRUE (diferente)
```

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

```r
5 > 2; 1 < 0   # TRUE; FALSE
```

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

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

```r
2 >= 2; 3 <= 10 # TRUE; TRUE
```

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

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

```r
# %in% (avalia se está contido)
2 %in% c(1, 2, 3)                          # TRUE
```

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

```r
"Adelie" %in% c("Chinstrap", "Gentoo")   # FALSE
```

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

2.3 Condicionais: tomando decisões no código

Condicionais permitem que seu código tome decisões baseadas em condições. São como perguntas "se... então... senão...".

Três formas principais:

1. `if/else` - Estrutura clássica para **um único valor**
   - Avalia uma condição e executa diferentes blocos de código
   - Útil para controle de fluxo em funções
2. `ifelse()` - Versão **vetorizada** para múltiplos valores
   - Aplica a condição a cada elemento de um vetor
   - Retorna um vetor de resultados
   - Ideal para criar novas colunas em data.frames
3. `case_when()` - Para **múltiplas condições** complexas
   - Função do pacote `dplyr` (requer library(dplyr))
   - Avalia várias regras em sequência
   - Para na primeira regra verdadeira

- Mais legível que `ifelse()` aninhados
- Integra-se perfeitamente com o pipe %>% e mutate()

```
# ==========================================
# 1. IF/ELSE - DECISÕES ÚNICAS
# ==========================================

# Exemplo 1: Classificar um reprodutor
dep_leite <- 920
acuracia <- 0.88

if (dep_leite > 800 & acuracia >= 0.80) {
  categoria <- "Elite"
  print(paste("Touro classificado como:", categoria))
  print("Recomendado para IA em todo o rebanho")
} else {
  categoria <- "Padrão"
  print(paste("Touro classificado como:", categoria))
  print("Uso restrito no rebanho")
}
```

```
#> [1] "Touro classificado como: Elite"
#> [1] "Recomendado para IA em todo o rebanho"
```

```
# Exemplo 2: Decisão de acasalamento com múltiplas opções
consanguinidade_combinada <- 0.07

if (consanguinidade_combinada < 0.05) {
  print("  Acasalamento RECOMENDADO - baixo risco")
} else if (consanguinidade_combinada >= 0.05 & consanguinidade_combinada < 0.10) {
  print("  Acasalamento com CAUTELA - risco moderado")
} else {
  print("  Acasalamento NÃO RECOMENDADO - alto risco")
}
```

```
#> [1] "  Acasalamento com CAUTELA - risco moderado"
```

```
# Exemplo 3: Função para avaliar touros
avaliar_touro <- function(dep_leite, dep_gordura, acuracia) {

  if (acuracia < 0.60) {
    return("Avaliação não confiável - aguardar mais filhas")
  }

  if (dep_leite > 900 & dep_gordura > 35 & acuracia >= 0.85) {
    return("ELITE - Top 1%")
  } else if (dep_leite > 700 & dep_gordura > 28 & acuracia >= 0.75) {
    return("SUPERIOR - Top 10%")
  } else if (dep_leite > 500 & dep_gordura > 22) {
    return("BOM - Acima da média")
```

```
  } else {
    return("REGULAR - Abaixo da média")
  }
}

# Testando a função
print(avaliar_touro(920, 38, 0.88))  # ELITE
```

```
#> [1] "ELITE - Top 1%"
```

```
print(avaliar_touro(750, 30, 0.80))  # SUPERIOR
```

```
#> [1] "SUPERIOR - Top 10%"
```

```
print(avaliar_touro(450, 20, 0.55))  # Avaliação não confiável
```

```
#> [1] "Avaliação não confiável - aguardar mais filhas"
```

```
# ============================================
# 2. IFELSE() - VETORIZADO PARA MÚLTIPLOS VALORES
# ============================================

# Exemplo 1: Classificar múltiplas DEPs
deps_peso <- c(25, 18, 32, 15, 28, 22, 8, 30)

# Classificar cada DEP
classificacao <- ifelse(deps_peso >= 25, "Superior", "Padrão")
print(classificacao)
```

```
#> [1] "Superior" "Padrão"   "Superior" "Padrão"   "Superior" "Padrão"   "Padrão"
#> [8] "Superior"
```

```
# "Superior" "Padrão" "Superior" "Padrão" "Superior" "Padrão" "Padrão" "Superior"


# Exemplo 2: Criar coluna de classificação em dataframe
rebanho <- data.frame(
  animal_id = c("V001", "V002", "V003", "V004", "V005"),
  dep_leite = c(850, 650, 920, 550, 780),
  acuracia = c(0.85, 0.75, 0.90, 0.60, 0.80)
)

# Adicionar classificação genética
rebanho$categoria_genetica <- ifelse(
  rebanho$dep_leite > 800,
  "Elite",
  "Padrão"
)

print(rebanho)
```

```
#>   animal_id dep_leite acuracia categoria_genetica
#> 1      V001       850     0.85              Elite
#> 2      V002       650     0.75             Padrão
#> 3      V003       920     0.90              Elite
#> 4      V004       550     0.60             Padrão
#> 5      V005       780     0.80             Padrão
```

```r
# Exemplo 3: ifelse aninhado (cuidado - pode ficar confuso!)
rebanho$status <- ifelse(
  rebanho$dep_leite > 850,
  "Excelente",
  ifelse(
    rebanho$dep_leite > 700,
    "Bom",
    "Regular"
  )
)

print(rebanho[, c("animal_id", "dep_leite", "status")])
```

```
#>   animal_id dep_leite    status
#> 1      V001       850       Bom
#> 2      V002       650   Regular
#> 3      V003       920 Excelente
#> 4      V004       550   Regular
#> 5      V005       780       Bom
```

```r
# Exemplo 4: Decisão de seleção
rebanho$selecionar <- ifelse(
  rebanho$dep_leite > 750 & rebanho$acuracia >= 0.75,
  "SIM",
  "NÃO"
)

print(rebanho[, c("animal_id", "dep_leite", "acuracia", "selecionar")])
```

```
#>   animal_id dep_leite acuracia selecionar
#> 1      V001       850     0.85        SIM
#> 2      V002       650     0.75        NÃO
#> 3      V003       920     0.90        SIM
#> 4      V004       550     0.60        NÃO
#> 5      V005       780     0.80        SIM
```

```r
# ============================================
# 3. CASE_WHEN() - MÚLTIPLAS CONDIÇÕES COMPLEXAS
# ============================================

library(dplyr)

# Exemplo 1: Sistema de classificação completo
```

```r
touros <- data.frame(
  id = c("T001", "T002", "T003", "T004", "T005", "T006"),
  dep_leite = c(950, 780, 650, 850, 450, 1100),
  dep_gordura = c(38, 30, 25, 32, 20, 42),
  dep_proteina = c(32, 28, 22, 30, 18, 38),
  acuracia = c(0.90, 0.78, 0.65, 0.85, 0.55, 0.92),
  consanguinidade = c(0.02, 0.05, 0.08, 0.03, 0.10, 0.01)
)

# Classificação com case_when (MUITO MAIS LEGÍVEL!)
touros <- touros %>%
  mutate(
    classificacao = case_when(
      # Primeiro verifica problemas
      acuracia < 0.60 ~ "Não confiável - aguardar",
      consanguinidade > 0.08 ~ "Descarte - alta consanguinidade",

      # Depois classifica por mérito genético
      dep_leite > 1000 & dep_gordura > 40 & dep_proteina > 35 ~ "Elite AAA",
      dep_leite > 900 & dep_gordura > 35 & dep_proteina > 30 ~ "Elite AA",
      dep_leite > 800 & dep_gordura > 30 & dep_proteina > 28 ~ "Elite A",
      dep_leite > 700 & dep_gordura > 25 ~ "Superior",
      dep_leite > 600 ~ "Bom",

      # Caso contrário
      TRUE ~ "Regular"
    )
  )

print(touros[, c("id", "dep_leite", "classificacao")])
```

```
#>     id dep_leite            classificacao
#> 1 T001       950                Elite AA
#> 2 T002       780                Superior
#> 3 T003       650                     Bom
#> 4 T004       850                 Elite A
#> 5 T005       450 Não confiável - aguardar
#> 6 T006      1100               Elite AAA
```

```r
# Exemplo 2: Recomendação de uso do touro
touros <- touros %>%
  mutate(
    recomendacao = case_when(
      classificacao %in% c("Elite AAA", "Elite AA") ~ "IA comercial + FIV",
      classificacao == "Elite A" ~ "IA comercial",
      classificacao == "Superior" ~ "IA no próprio rebanho",
      classificacao == "Bom" ~ "Monta natural",
      classificacao == "Regular" ~ "Não usar",
```

```
      TRUE ~ "Avaliar novamente"
    )
  )

print(touros[, c("id", "classificacao", "recomendacao")])
```

```
#>     id            classificacao          recomendacao
#> 1 T001                 Elite AA    IA comercial + FIV
#> 2 T002                Superior IA no próprio rebanho
#> 3 T003                      Bom         Monta natural
#> 4 T004                  Elite A          IA comercial
#> 5 T005 Não confiável - aguardar     Avaliar novamente
#> 6 T006                Elite AAA    IA comercial + FIV
```

```
# Exemplo 3: Índice de seleção com múltiplos critérios
vacas <- data.frame(
  id = paste0("V", sprintf("%03d", 1:10)),
  dep_leite = c(850, 750, 920, 650, 780, 500, 880, 720, 950, 600),
  dep_gordura = c(32, 28, 38, 24, 30, 20, 34, 26, 40, 22),
  dep_proteina = c(28, 25, 32, 20, 27, 18, 30, 24, 35, 21),
  fertilidade = c(0.65, 0.58, 0.70, 0.45, 0.62, 0.40, 0.68, 0.55, 0.72, 0.48),
  idade = c(4, 6, 3, 8, 5, 9, 4, 7, 3, 8)
)

# Decisão complexa de seleção
vacas <- vacas %>%
  mutate(
    decisao = case_when(
      # Descarte por idade ou fertilidade
      idade > 8 ~ "Descarte - idade avançada",
      fertilidade < 0.50 ~ "Descarte - baixa fertilidade",

      # Mães de touros (melhor genética)
      dep_leite > 900 & dep_gordura > 35 & dep_proteina > 30 &
        fertilidade > 0.65 ~ "Mãe de touro - FIV",

      # Doadoras de embrião
      dep_leite > 800 & dep_gordura > 30 & dep_proteina > 27 &
        fertilidade > 0.60 & idade <= 6 ~ "Doadora - TE",

      # Matrizes superiores
      dep_leite > 750 & dep_gordura > 28 & fertilidade > 0.58 ~ "Matriz elite",

      # Matrizes padrão
      dep_leite > 600 & dep_gordura > 22 ~ "Matriz padrão",

      # Outros casos
      TRUE ~ "Avaliar individualmente"
```

```
    )
  )

print(vacas[, c("id", "dep_leite", "fertilidade", "idade", "decisao")])
```

```
#>      id dep_leite fertilidade idade                     decisao
#> 1  V001       850        0.65     4                 Doadora - TE
#> 2  V002       750        0.58     6                Matriz padrão
#> 3  V003       920        0.70     3           Mãe de touro - FIV
#> 4  V004       650        0.45     8 Descarte - baixa fertilidade
#> 5  V005       780        0.62     5                  Matriz elite
#> 6  V006       500        0.40     9    Descarte - idade avançada
#> 7  V007       880        0.68     4                 Doadora - TE
#> 8  V008       720        0.55     7                Matriz padrão
#> 9  V009       950        0.72     3           Mãe de touro - FIV
#> 10 V010       600        0.48     8 Descarte - baixa fertilidade
```

```
# ==========================================
# EXEMPLO 4: CÁLCULO DE ÍNDICE ECONÔMICO
# ==========================================

# Gado de corte - múltiplas características
bovinos_corte <- data.frame(
  id = paste0("B", 1:8),
  dep_peso_desmame = c(25, 18, 32, 15, 28, 22, 8, 30),
  dep_ganho_pos = c(20, 15, 25, 12, 22, 18, 10, 28),
  dep_aol = c(3.2, 2.1, 3.8, 1.5, 3.0, 2.5, 1.2, 4.0),  # área olho lombo
  dep_eg = c(-1.5, -0.8, -2.0, 0.5, -1.2, -0.5, 1.0, -2.5),  # esp. gordura
  temperamento = c("Ótimo", "Bom", "Ótimo", "Regular", "Bom", "Regular", "Ruim", "Ótimo")
)

# Calcular índice e classificar
bovinos_corte <- bovinos_corte %>%
  mutate(
    # Índice composto (pesos econômicos simplificados)
    indice = (dep_peso_desmame * 2) +
             (dep_ganho_pos * 1.5) +
             (dep_aol * 5) -
             (dep_eg * 2),  # gordura negativa é boa

    # Classificação final
    categoria = case_when(
      temperamento == "Ruim" ~ "Descarte - temperamento",
      indice > 100 ~ "Elite - Top 1%",
      indice > 80 ~ "Superior - Top 10%",
      indice > 60 ~ "Bom - Top 25%",
      indice > 40 ~ "Médio",
      TRUE ~ "Abaixo da média"
```

```
    )
  )

print(bovinos_corte[, c("id", "indice", "temperamento", "categoria")])
```

```
#>   id indice temperamento               categoria
#> 1 B1   99.0        Ótimo      Superior - Top 10%
#> 2 B2   70.6          Bom           Bom - Top 25%
#> 3 B3  124.5        Ótimo          Elite - Top 1%
#> 4 B4   54.5      Regular                   Médio
#> 5 B5  106.4          Bom          Elite - Top 1%
#> 6 B6   84.5      Regular      Superior - Top 10%
#> 7 B7   35.0         Ruim Descarte - temperamento
#> 8 B8  127.0        Ótimo          Elite - Top 1%
```

```
# ==========================================
# COMPARAÇÃO: ifelse() vs case_when()
# ==========================================

# DIFÍCIL DE LER com ifelse aninhado:
vacas$classe_ruim <- ifelse(
  vacas$dep_leite > 900,
  "Elite",
  ifelse(
    vacas$dep_leite > 800,
    "Superior",
    ifelse(
      vacas$dep_leite > 700,
      "Boa",
      ifelse(
        vacas$dep_leite > 600,
        "Regular",
        "Ruim"
      )
    )
  )
)

# FÁCIL DE LER com case_when:
vacas <- vacas %>%
  mutate(
    classe_boa = case_when(
      dep_leite > 900 ~ "Elite",
      dep_leite > 800 ~ "Superior",
      dep_leite > 700 ~ "Boa",
      dep_leite > 600 ~ "Regular",
      TRUE ~ "Ruim"
    )
```

```r
  )


# ==========================================
# EXERCÍCIO PRÁTICO
# ==========================================

# Base de dados completa
plantel <- data.frame(
  animal = paste0("A", sprintf("%03d", 1:15)),
  dep_leite = c(950, 780, 650, 850, 450, 920, 720, 550, 880, 600, 1050, 700, 820, 580, 960),
  dep_gordura = c(38, 30, 25, 32, 20, 36, 28, 22, 34, 24, 42, 26, 33, 21, 39),
  acuracia = c(0.90, 0.78, 0.65, 0.85, 0.55, 0.88, 0.72, 0.60, 0.82, 0.68, 0.92, 0.70, 0.80, 0.
  idade = c(4, 6, 8, 5, 9, 3, 7, 8, 4, 9, 3, 6, 5, 10, 4),
  n_partos = c(2, 4, 6, 3, 7, 1, 5, 6, 2, 7, 1, 4, 3, 8, 2),
  consanguinidade = c(0.02, 0.05, 0.08, 0.03, 0.10, 0.01, 0.06, 0.09, 0.03, 0.12, 0.01, 0.05, 0
)

# Criar sistema completo de classificação e decisão
plantel <- plantel %>%
  mutate(
    # 1. Confiabilidade da avaliação
    confiabilidade = case_when(
      acuracia >= 0.85 ~ "Alta",
      acuracia >= 0.70 ~ "Média",
      TRUE ~ "Baixa"
    ),

    # 2. Mérito genético
    merito = case_when(
      dep_leite > 900 & dep_gordura > 35 ~ "Excepcional",
      dep_leite > 800 & dep_gordura > 30 ~ "Superior",
      dep_leite > 700 & dep_gordura > 25 ~ "Bom",
      dep_leite > 600 ~ "Regular",
      TRUE ~ "Inferior"
    ),

    # 3. Status reprodutivo
    status_reprodutivo = case_when(
      idade > 9 | n_partos > 7 ~ "Final de vida produtiva",
      idade >= 7 | n_partos >= 5 ~ "Madura",
      idade <= 4 & n_partos <= 2 ~ "Jovem",
      TRUE ~ "Adulta"
    ),

    # 4. Decisão final de manejo
    decisao_final = case_when(
```

```
      # Problemas graves
      consanguinidade > 0.10 ~ "DESCARTE - alta consanguinidade",
      idade > 9 & dep_leite < 700 ~ "DESCARTE - idade e baixa produção",
      confiabilidade == "Baixa" & merito %in% c("Regular", "Inferior") ~
        "AVALIAR NOVAMENTE - dados insuficientes",

      # Animais elite
      merito == "Excepcional" & confiabilidade == "Alta" & idade <= 6 ~
        "MÃE DE TOURO - programa FIV",
      merito %in% c("Excepcional", "Superior") & confiabilidade %in% c("Alta", "Média") ~
        "DOADORA - programa TE",

      # Animais bons
      merito %in% c("Superior", "Bom") & consanguinidade < 0.05 ~
        "MATRIZ ELITE - manter no rebanho",
      merito == "Bom" ~ "MATRIZ PADRÃO - manter",

      # Outros
      merito == "Regular" & idade < 6 ~ "MONITORAR - dar mais oportunidade",
      TRUE ~ "AVALIAR CASO A CASO"
    )
  )

# Visualizar resultado
print(plantel[, c("animal", "merito", "confiabilidade", "decisao_final")])
```

```
#>    animal      merito confiabilidade                          decisao_final
#> 1    A001 Excepcional           Alta              MÃE DE TOURO - programa FIV
#> 2    A002         Bom          Média                   MATRIZ PADRÃO - manter
#> 3    A003     Regular          Baixa AVALIAR NOVAMENTE - dados insuficientes
#> 4    A004    Superior           Alta                    DOADORA - programa TE
#> 5    A005    Inferior          Baixa AVALIAR NOVAMENTE - dados insuficientes
#> 6    A006 Excepcional           Alta              MÃE DE TOURO - programa FIV
#> 7    A007         Bom          Média                   MATRIZ PADRÃO - manter
#> 8    A008    Inferior          Baixa AVALIAR NOVAMENTE - dados insuficientes
#> 9    A009    Superior          Média                    DOADORA - programa TE
#> 10   A010    Inferior          Baixa         DESCARTE - alta consanguinidade
#> 11   A011 Excepcional           Alta              MÃE DE TOURO - programa FIV
#> 12   A012     Regular          Média                      AVALIAR CASO A CASO
#> 13   A013    Superior          Média                    DOADORA - programa TE
#> 14   A014    Inferior          Baixa         DESCARTE - alta consanguinidade
#> 15   A015 Excepcional           Alta              MÃE DE TOURO - programa FIV
```

```
# Resumo por decisão
table(plantel$decisao_final)
```

```
#>
#>                     AVALIAR CASO A CASO AVALIAR NOVAMENTE - dados insuficientes
```

```
#>                                    1                                    3
#>         DESCARTE - alta consanguinidade                  DOADORA - programa TE
#>                                    2                                    3
#>             MÃE DE TOURO - programa FIV                MATRIZ PADRÃO - manter
#>                                    4                                    2
```

Dica didática: use `ifelse()` quando quiser **vetorizar**; `case_when()` quando houver **várias regras**.

---

3 Loops, Vetorização e Funções (25 min)

3.1 O que são loops?

Um **loop** (laço) é uma estrutura que repete um bloco de código várias vezes. São fundamentais para automatizar tarefas repetitivas em programas de melhoramento genético.

3.2 Tipos de loops no R

3.2.1 Opções Nativas (R Base)

3.2.1.1 1. Loop `for` - Número Definido de Iterações

O loop `for` é o mais comum e executa código uma vez para cada elemento de uma sequência.

Sintaxe: `for (variável in sequência) { código }`

```
# ============================================
# LOOP FOR - EXEMPLOS BÁSICOS
# ============================================

# Exemplo 1: Calcular DEPs padronizadas
deps_peso <- c(25, 18, 32, 15, 28, 22, 8, 30)
animal_ids <- paste0("A", sprintf("%03d", 1:length(deps_peso)))

media <- mean(deps_peso)
dp <- sd(deps_peso)

cat("DEPs PADRONIZADAS\n")
```

```
#> DEPs PADRONIZADAS
```

```
cat("=================\n\n")
```

```
#> =================
```

```
for (i in 1:length(deps_peso)) {
  dep_padronizada <- (deps_peso[i] - media) / dp
  cat("Animal", animal_ids[i], "- DEP:", deps_peso[i],
      "| Padronizada:", round(dep_padronizada, 2), "\n")
}
```

```
#> Animal A001 - DEP: 25 | Padronizada: 0.34
```

```
#> Animal A002 - DEP: 18 | Padronizada: -0.52
#> Animal A003 - DEP: 32 | Padronizada: 1.19
#> Animal A004 - DEP: 15 | Padronizada: -0.89
#> Animal A005 - DEP: 28 | Padronizada: 0.7
#> Animal A006 - DEP: 22 | Padronizada: -0.03
#> Animal A007 - DEP: 8 | Padronizada: -1.74
#> Animal A008 - DEP: 30 | Padronizada: 0.95
```

```r
# Exemplo 2: Simular ganho genético por geração
n_geracoes <- 10
dep_inicial <- 500
intensidade_selecao <- 1.5  # i
herdabilidade <- 0.30       # h²
desvio_padrao <- 100        #

# Resposta à seleção: R = i × h² ×
resposta <- intensidade_selecao * herdabilidade * desvio_padrao

deps_geracao <- numeric(n_geracoes)
deps_geracao[1] <- dep_inicial

cat("\n\nSIMULAÇÃO DE GANHO GENÉTICO\n")
```

```
#>
#>
#> SIMULAÇÃO DE GANHO GENÉTICO
```

```r
cat("============================\n\n")
```

```
#> ============================
```

```r
for (geracao in 2:n_geracoes) {
  deps_geracao[geracao] <- deps_geracao[geracao - 1] + resposta
  cat("Geração", geracao, "- DEP média:",
      round(deps_geracao[geracao], 1), "kg\n")
}
```

```
#> Geração 2 - DEP média: 545 kg
#> Geração 3 - DEP média: 590 kg
#> Geração 4 - DEP média: 635 kg
#> Geração 5 - DEP média: 680 kg
#> Geração 6 - DEP média: 725 kg
#> Geração 7 - DEP média: 770 kg
#> Geração 8 - DEP média: 815 kg
#> Geração 9 - DEP média: 860 kg
#> Geração 10 - DEP média: 905 kg
```

```r
ganho_total <- deps_geracao[n_geracoes] - deps_geracao[1]
cat("\nGanho total:", round(ganho_total, 1), "kg")
```

```
#>
```

```
#> Ganho total: 405 kg
```

3.2.1.2 2. Loop `while` - Baseado em Condição

O `while` repete **enquanto** uma condição for verdadeira. Usado quando você não sabe quantas iterações serão necessárias.

Sintaxe: `while (condição) { código }`

Cuidado: pode criar loops infinitos se a condição nunca ficar FALSE!

```r
# ==========================================
# LOOP WHILE - CONVERGÊNCIA
# ==========================================

# Algoritmo de convergência (exemplo simplificado de EM)
variancia_genetica <- 100
variancia_residual <- 200
criterio_convergencia <- 0.01
max_iteracoes <- 100

iteracao <- 0
convergiu <- FALSE

cat("ALGORITMO DE CONVERGÊNCIA\n")
```

```
#> ALGORITMO DE CONVERGÊNCIA
```

```r
cat("=========================\n\n")
```

```
#> =========================
```

```r
while (!convergiu & iteracao < max_iteracoes) {
  iteracao <- iteracao + 1

  vg_anterior <- variancia_genetica
  vr_anterior <- variancia_residual

  # Atualização simulada
  variancia_genetica <- vg_anterior * 1.05
  variancia_residual <- vr_anterior * 0.98

  # Verificar convergência
  mudanca_vg <- abs(variancia_genetica - vg_anterior)
  mudanca_vr <- abs(variancia_residual - vr_anterior)

  if (mudanca_vg < criterio_convergencia & mudanca_vr < criterio_convergencia) {
    convergiu <- TRUE
  }

  if (iteracao %% 10 == 0) {
    cat("Iteração", iteracao, "- VG:", round(variancia_genetica, 2),
        "VR:", round(variancia_residual, 2), "\n")
```

```
  }
}
```

```
#> Iteração 10 - VG: 162.89 VR: 163.41
#> Iteração 20 - VG: 265.33 VR: 133.52
#> Iteração 30 - VG: 432.19 VR: 109.1
#> Iteração 40 - VG: 704 VR: 89.14
#> Iteração 50 - VG: 1146.74 VR: 72.83
#> Iteração 60 - VG: 1867.92 VR: 59.51
#> Iteração 70 - VG: 3042.64 VR: 48.62
#> Iteração 80 - VG: 4956.14 VR: 39.73
#> Iteração 90 - VG: 8073.04 VR: 32.46
#> Iteração 100 - VG: 13150.13 VR: 26.52
```

```
cat("\nConvergiu em", iteracao, "iterações")
```

```
#>
#> Convergiu em 100 iterações
```

```
cat("\nHerdabilidade final:",
    round(variancia_genetica / (variancia_genetica + variancia_residual), 3))
```

```
#>
#> Herdabilidade final: 0.998
```

3.2.1.3 3. Loop `repeat` - Loop Infinito com Controle Manual

O `repeat` repete indefinidamente até encontrar um `break`. **Sempre** precisa de condição de parada.

Sintaxe: `repeat { código; if (condição) break }`

```
# ============================================
# LOOP REPEAT - MÚLTIPLOS CRITÉRIOS DE PARADA
# ============================================

set.seed(123)
populacao_size <- 100
dep_media <- 500
geracao <- 0
max_geracoes <- 50
meta_dep <- 700
sem_melhoria <- 0
max_sem_melhoria <- 10

cat("ALGORITMO GENÉTICO\n")
```

```
#> ALGORITMO GENÉTICO
```

```
cat("==================\n\n")
```

```
#> ==================
```

```r
repeat {
  geracao <- geracao + 1

  deps_populacao <- rnorm(populacao_size, mean = dep_media, sd = 50)
  deps_selecionados <- sort(deps_populacao, decreasing = TRUE)[1:20]
  nova_media <- mean(deps_selecionados)

  if (abs(nova_media - dep_media) < 1) {
    sem_melhoria <- sem_melhoria + 1
  } else {
    sem_melhoria <- 0
  }

  dep_media <- nova_media

  if (geracao %% 5 == 0) {
    cat("Geração", geracao, "- DEP:", round(dep_media, 1), "kg\n")
  }

  # Múltiplos critérios de parada
  if (dep_media >= meta_dep) {
    cat("\n META ATINGIDA!\n")
    break
  }
  if (geracao >= max_geracoes) {
    cat("\n Máximo de gerações\n")
    break
  }
  if (sem_melhoria >= max_sem_melhoria) {
    cat("\n Sem melhoria há", max_sem_melhoria, "gerações\n")
    break
  }
}
```

```
#>
#>   META ATINGIDA!
```

3.2.1.4 4. Família `apply()` - Alternativa Funcional As funções da família `apply` são alternativas mais elegantes e geralmente mais rápidas que loops tradicionais.

```r
# ============================================
# FAMÍLIA APPLY - R BASE
# ============================================

# apply() - Para matrizes/arrays
deps_matriz <- matrix(
  c(25, 32, 1.8,
    18, 28, 1.5,
```

```
    32, 38, 2.2,
    15, 22, 1.2,
    28, 35, 2.0),
  nrow = 5, byrow = TRUE,
  dimnames = list(
    paste0("Animal_", 1:5),
    c("DEP_Peso", "DEP_Ganho", "DEP_AOL")
  )
)

cat("Matriz de DEPs:\n")
```

```
#> Matriz de DEPs:
```

```
print(deps_matriz)
```

```
#>          DEP_Peso DEP_Ganho DEP_AOL
#> Animal_1       25        32     1.8
#> Animal_2       18        28     1.5
#> Animal_3       32        38     2.2
#> Animal_4       15        22     1.2
#> Animal_5       28        35     2.0
```

```
# Média por animal (linha) - MARGIN = 1
cat("\nMédia por animal:\n")
```

```
#>
#> Média por animal:
```

```
medias_animais <- apply(deps_matriz, 1, mean)
print(round(medias_animais, 2))
```

```
#> Animal_1 Animal_2 Animal_3 Animal_4 Animal_5
#>    19.60    15.83    24.07    12.73    21.67
```

```
# Média por característica (coluna) - MARGIN = 2
cat("\nMédia por característica:\n")
```

```
#>
#> Média por característica:
```

```
medias_caracteristicas <- apply(deps_matriz, 2, mean)
print(round(medias_caracteristicas, 2))
```

```
#>  DEP_Peso DEP_Ganho   DEP_AOL
#>     23.60     31.00      1.74
```

```
# lapply() - Retorna lista
rebanhos <- list(
  Rebanho_A = c(850, 780, 920, 750, 880),
  Rebanho_B = c(720, 650, 800, 680, 760),
  Rebanho_C = c(950, 880, 1020, 870, 930)
```

```
)

cat("\n\nEstatísticas por rebanho (lapply):\n")
```

```
#>
#>
#> Estatísticas por rebanho (lapply):
```

```
estatisticas <- lapply(rebanhos, function(deps) {
  list(
    media = mean(deps),
    dp = sd(deps),
    n = length(deps),
    cv = sd(deps)/mean(deps)*100
  )
})
print(estatisticas)
```

```
#> $Rebanho_A
#> $Rebanho_A$media
#> [1] 836
#>
#> $Rebanho_A$dp
#> [1] 70.21396
#>
#> $Rebanho_A$n
#> [1] 5
#>
#> $Rebanho_A$cv
#> [1] 8.398799
#>
#>
#> $Rebanho_B
#> $Rebanho_B$media
#> [1] 722
#>
#> $Rebanho_B$dp
#> [1] 60.16644
#>
#> $Rebanho_B$n
#> [1] 5
#>
#> $Rebanho_B$cv
#> [1] 8.333301
#>
#>
#> $Rebanho_C
#> $Rebanho_C$media
```

```
#> [1] 930
#>
#> $Rebanho_C$dp
#> [1] 60.41523
#>
#> $Rebanho_C$n
#> [1] 5
#>
#> $Rebanho_C$cv
#> [1] 6.496261
```

```
# sapply() - Simplifica para vetor/matriz
cat("\n\nMédias dos rebanhos (sapply):\n")
```

```
#>
#>
#> Médias dos rebanhos (sapply):
```

```
medias_rebanhos <- sapply(rebanhos, mean)
print(round(medias_rebanhos, 1))
```

```
#> Rebanho_A Rebanho_B Rebanho_C
#>       836       722       930
```

```
# tapply() - Aplicar por grupos
animais <- data.frame(
  dep_leite = c(850, 750, 920, 680, 820, 650, 880, 700),
  raca = c("Holandês", "Jersey", "Holandês", "Jersey",
           "Holandês", "Jersey", "Holandês", "Jersey")
)

cat("\n\nMédia de DEP por raça (tapply):\n")
```

```
#>
#>
#> Média de DEP por raça (tapply):
```

```
media_raca <- tapply(animais$dep_leite, animais$raca, mean)
print(round(media_raca, 1))
```

```
#> Holandês   Jersey
#>    867.5    695.0
```

3.2.2 Opções de Pacotes (purrr/tidyverse)

3.2.2.1 Família `map()` - Sintaxe Moderna e Consistente A família `map()` do pacote **purrr** oferece sintaxe mais consistente e previsível que `apply()`.

```
library(purrr)

# ==========================================
# FAMÍLIA MAP - PURRR
```

```r
# ==========================================

# map() - Retorna lista
cat("Médias dos rebanhos (map):\n")
```

```
#> Médias dos rebanhos (map):
```

```r
medias_map <- map(rebanhos, mean)
print(medias_map)
```

```
#> $Rebanho_A
#> [1] 836
#>
#> $Rebanho_B
#> [1] 722
#>
#> $Rebanho_C
#> [1] 930
```

```r
# map_dbl() - Retorna vetor numérico
cat("\n\nMédias dos rebanhos como vetor (map_dbl):\n")
```

```
#>
#>
#> Médias dos rebanhos como vetor (map_dbl):
```

```r
medias_fazendas <- map_dbl(rebanhos, mean)
print(round(medias_fazendas, 1))
```

```
#> Rebanho_A Rebanho_B Rebanho_C
#>       836       722       930
```

```r
# map_chr() - Retorna vetor de caracteres
cat("\n\nClassificação dos rebanhos (map_chr):\n")
```

```
#>
#>
#> Classificação dos rebanhos (map_chr):
```

```r
classificacao <- map_chr(rebanhos, ~{
  media <- mean(.x)
  case_when(
    media > 900 ~ "Excelente",
    media > 850 ~ "Ótimo",
    media > 800 ~ "Bom",
    TRUE ~ "Regular"
  )
})
print(classificacao)
```

```
#>   Rebanho_A   Rebanho_B   Rebanho_C
```

```
#>       "Bom"   "Regular" "Excelente"
```

```
# map_df() - Retorna data frame
cat("\n\nEstatísticas completas (map_df):\n")
```

```
#>
#>
#> Estatísticas completas (map_df):
```

```
estatisticas_df <- map_df(rebanhos, ~{
  tibble(
    n = length(.x),
    media = mean(.x),
    dp = sd(.x),
    min = min(.x),
    max = max(.x)
  )
}, .id = "rebanho")
print(estatisticas_df)
```

```
#> # A tibble: 3 x 6
#>   rebanho       n media    dp   min   max
#>   <chr>     <int> <dbl> <dbl> <dbl> <dbl>
#> 1 Rebanho_A     5   836  70.2   750   920
#> 2 Rebanho_B     5   722  60.2   650   800
#> 3 Rebanho_C     5   930  60.4   870  1020
```

3.3 Loops vs. Vetorização

Vetorização é quando operações são aplicadas automaticamente a todos os elementos sem loop explícito.

Por que evitar loops em R?

R é uma linguagem **vetorizada**, o que significa que muitas operações funcionam automaticamente em vetores inteiros. Operações vetorizadas são:

- **Mais rápidas** (otimizadas internamente em C/Fortran)
- **Mais legíveis** (menos linhas de código)
- **Mais idiomáticas** (o "jeito R" de fazer)

Quando usar loops:

- Quando não existe alternativa vetorizada
- Para operações que dependem de iterações anteriores
- Em simulações e processos iterativos

```
# ============================================
# COMPARAÇÃO: LOOP vs VETORIZAÇÃO
# ============================================

deps <- c(850, 750, 920, 680, 820)
```

```r
# COM LOOP (lento, verboso)
deps_padronizadas_loop <- numeric(length(deps))
for (i in 1:length(deps)) {
  deps_padronizadas_loop[i] <- (deps[i] - mean(deps)) / sd(deps)
}
cat("Com loop:\n")
```

```
#> Com loop:
```

```r
print(round(deps_padronizadas_loop, 2))
```

```
#> [1]  0.50 -0.58  1.26 -1.34  0.17
```

```r
# VETORIZADO (rápido, limpo)
deps_padronizadas_vet <- (deps - mean(deps)) / sd(deps)
cat("\nVetorizado:\n")
```

```
#>
#> Vetorizado:
```

```r
print(round(deps_padronizadas_vet, 2))
```

```
#> [1]  0.50 -0.58  1.26 -1.34  0.17
```

```r
# Resultados idênticos
cat("\nResultados iguais?", all.equal(deps_padronizadas_loop, deps_padronizadas_vet))
```

```
#>
#> Resultados iguais? TRUE
```

Hierarquia de preferência no R:

1. **Vetorização** (quando possível) - `deps * 2`
2. **Família apply/map** - `sapply()`, `map_dbl()`
3. **Loop for** - quando não há alternativa
4. **while/repeat** - apenas quando necessário

Regra de ouro: Se você pode vetorizar, vetorize!

```r
valores <- 1:5

# Loop for (didático, mas não idiomático)
soma <- 0
for (v in valores) {
  soma <- soma + v
}
cat("Soma com loop:", soma, "\n")
```

```
#> Soma com loop: 15
```

```r
# Vetorizado (preferido em R!)
cat("Soma vetorizada:", sum(valores), "\n")
```

```
#> Soma vetorizada: 15
```

3.4 Funções: empacotando lógica reutilizável

O que são funções?

Funções são blocos de código que realizam uma tarefa específica e podem ser reutilizados. São fundamentais para: - **Organizar** código em partes lógicas - **Reutilizar** lógica sem repetir código - **Documentar** intenções através de nomes descritivos - **Facilitar** manutenção e debugging

Estrutura de uma função:

```
nome_funcao <- function(argumento1, argumento2 = valor_padrao) {
  # corpo da função
  resultado <- alguma_operacao
  return(resultado)  # return é opcional (retorna última expressão)
}
```

Boas práticas: - Use nomes descritivos que indiquem o que a função faz - Valide entradas com `stop()`, `stopifnot()` ou `if` - Documente com comentários o que a função faz e quais são os argumentos - Retorne sempre o mesmo tipo de objeto

Exemplo prático: calculadora de IMC

```
# Fórmula: IMC = peso(kg) / altura(m)^2
imc <- function(peso, altura) {
  # Validação: altura não pode ser zero ou negativa
  if (any(altura <= 0)) stop("Altura deve ser > 0")

  # Cálculo vetorizado (funciona com um ou vários valores)
  peso / (altura ^ 2)
}

# Testando com múltiplos valores
imc(c(70, 80), c(1.70, 1.80))
```

```
#> [1] 24.22145 24.69136
```

```
# Função para classificar IMC usando case_when()
classificar_imc <- function(imc) {
  dplyr::case_when(
    imc < 18.5              ~ "Abaixo do peso",
    imc >= 18.5 & imc < 25  ~ "Normal",
    imc >= 25   & imc < 30  ~ "Sobrepeso",
    imc >= 30               ~ "Obesidade"
  )
}

# Combinando as duas funções
val <- imc(80, 1.75)
classificar_imc(val)
```

```
#> [1] "Sobrepeso"
```

Princípio DRY (Don't Repeat Yourself): se você copiou e colou código mais de 2 vezes, provavelmente deveria criar uma função!

---

4 Introdução ao Tidyverse (45 min)

Vamos aplicar `dplyr` no dataset **palmerpenguins** e criar um pequeno pipeline.

```
library(dplyr)
library(palmerpenguins)

# Remover linhas com NAs nas colunas essenciais
peng <- penguins |>
  filter(!is.na(species),
         !is.na(bill_length_mm),
         !is.na(bill_depth_mm),
         !is.na(flipper_length_mm),
         !is.na(body_mass_g))

# Selecionar só o que precisamos
peng_sel <- peng |>
  select(species, island, bill_length_mm, bill_depth_mm, flipper_length_mm, body_mass_g)

# Criar nova variável (razão do bico) e reordenar
peng_feat <- peng_sel |>
  mutate(raz_bico = bill_length_mm / bill_depth_mm) |>
  arrange(species, desc(raz_bico))

# Resumo por espécie
resumo <- peng_feat |>
  group_by(species) |>
  summarize(
    n = n(),
    media_flipper = mean(flipper_length_mm),
    sd_flipper    = sd(flipper_length_mm),
    media_massa   = mean(body_mass_g)
  )
resumo
```

```
#> # A tibble: 3 x 5
#>   species       n media_flipper sd_flipper media_massa
#>   <fct>     <int>         <dbl>      <dbl>       <dbl>
#> 1 Adelie      151          190.       6.54       3701.
#> 2 Chinstrap    68          196.       7.13       3733.
#> 3 Gentoo      123          217.       6.48       5076.
```

4.1 Pipe: %>% vs. |>

```
# Ambos funcionam; escolha um padrão para a turma.
# Exemplo com |> (pipe nativo do R >= 4.1):
penguins |>
  tidyr::drop_na(bill_length_mm) |>
  dplyr::summarize(media = mean(bill_length_mm))
```

```
#> # A tibble: 1 x 1
#>   media
#>   <dbl>
#> 1  43.9
```

---

5 Datas com `lubridate` (10 min)

Datas aparecem em **quase todos** os projetos. Vamos ilustrar rapidamente.

```
library(lubridate)

# Criação e parsing
ymd("2025-11-18")
```

```
#> [1] "2025-11-18"
```

```
dmy("18/11/2025")
```

```
#> [1] "2025-11-18"
```

```
mdy("11-18-2025")
```

```
#> [1] "2025-11-18"
```

```
# Componentes
hoje <- today()
year(hoje)
```

```
#> [1] 2025
```

```
month(hoje)
```

```
#> [1] 11
```

```
wday(hoje, label = TRUE, abbr = FALSE)
```

```
#> [1] Monday
#> 7 Levels: Sunday < Monday < Tuesday < Wednesday < Thursday < ... < Saturday
```

```
# Operações simples
hoje + days(14)
```

```
#> [1] "2025-11-17"
```

```r
interval(ymd("2025-11-01"), ymd("2025-11-18"))
```

```
#> [1] 2025-11-01 UTC--2025-11-18 UTC
```

Integrando no pipeline: quando houver colunas de data, transforme-as e derive **mês/ano** para agregações.

---

6 Exercícios Práticos (20–25 min)

Dataset: `palmerpenguins::penguins`

6.1 Exercício 1 — Condicionais

1. Crie um vetor de 8 notas qualquer.
2. Classifique com `ifelse()` como **Aprovado/Recuperação** (corte em 7).
3. Depois, crie uma classificação mais rica usando `case_when()` com 4 faixas.

```r
# Seu código aqui
```

6.2 Exercício 2 — Funções

1. Escreva uma função `zscore(x)` que centraliza e escala (média 0, desvio 1).
2. Aplique em `bill_length_mm` **removendo NAs** antes.
3. Faça um segundo argumento opcional `na_rm = TRUE` dentro da função.

```r
# Seu código aqui
```

6.3 Exercício 3 — Pipeline `dplyr`

1. Crie `peng3` filtrando linhas completas nas 4 medidas principais.
2. Calcule, por espécie, média e desvio da nadadeira (`flipper_length_mm`).
3. Ordene do maior para o menor e mostre as 5 primeiras linhas.

```r
# Seu código aqui
```

6.4 Exercício 4 — Datas com `lubridate`

1. Crie um vetor com 5 datas em formato "dd/mm/aaaa".
2. Converta com `dmy()` e extraia `month()` (com rótulo).
3. Some 30 dias à primeira data e compute o intervalo até a última.

```r
# Seu código aqui
```

---

7 Boas Práticas e Debugging (20 min)

- Use **nomes descritivos** em `snake_case`.
- Comente o **porquê** (não só o que) no código.
- Valide entradas em funções (`stop()` para erros previsíveis).
- Leia mensagens de erro **de baixo para cima** (stack trace).
- Mantenha scripts curtos e reutilizáveis.

8 Ferramentas úteis

```r
# message(), warning(), stop() para sinalizar eventos
# browser() para inspecionar dentro de uma função (quando eval=TRUE)
# traceback() após um erro
```

> **IA como apoio (responsável):** use ChatGPT/Claude para **explicar erros** e sugerir melhorias, mas sempre **entenda e teste** o código.

---

9 Commit do Dia

1. Salve como `scripts/02_logica_funcoes.R` ou `materiais/dia2_logica_funcoes.Rmd` (este arquivo).

2. No **Terminal do RStudio**:

```r
git add scripts/02_logica_funcoes.R
git commit -m "Dia 2: lógica, funções e tidyverse (com lubridate)"
git push origin main
```

> Lembre-se: você está trabalhando **no SEU fork**. O repositório original permanece protegido.

---

10 Checklist de encerramento

- ☐ Dominou operadores lógicos e relacionais
- ☐ Entendeu diferenças entre `if/else`, `ifelse()` e `case_when()`
- ☐ Compreendeu por que vetorização é preferível a loops
- ☐ Criou suas primeiras funções com validação
- ☐ Aplicou pipeline básico com `dplyr`
- ☐ Explorou manipulação de datas com `lubridate`
- ☐ Realizou commit e push no seu fork

---

11 Referências rápidas

- **`dplyr` cheatsheet**: https://posit.co/resources/cheatsheets/
- **R for Data Science (2e)**: https://r4ds.hadley.nz/
- **Happy Git with R**: https://happygitwithr.com/
- **palmerpenguins**: https://allisonhorst.github.io/palmerpenguins/
- **lubridate**: https://lubridate.tidyverse.org/

Nos vemos no Dia 3 para transformação de dados e visualização com ggplot2!