Aula 03 – Introdução ao tidyverse - Curso R para iniciantes

Adhemar Ranciaro Neto

Aula 03 – Introdução ao tidyverse

Nesta aula, vamos conhecer o conjunto de pacotes `tidyverse`, que oferece ferramentas modernas e consistentes para manipulação e visualização de dados.

O que é o tidyverse?

O tidyverse é um ecossistema de pacotes para ciência de dados em R, incluindo:

- `readr`: leitura de arquivos
- `dplyr`: manipulação de dados
- `ggplot2`: visualização
- `tidyr`: organização de dados
- `tibble`: substituição moderna para data frames

Carregando o pacote

```
# Instalar (se necessário)
 install.packages("tidyverse")

library(tidyverse)
```

Lendo dados com `readr::read_csv()`

```
# Exemplo de leitura
vendas <- read_csv("vendas.csv")


# Visualizando os dados
head(vendas)
```

Usando `tibble` no lugar de data.frame

```
tibble(
  produto = c("A", "B", "C"),
  preco = c(10, 15, 12),
  vendido = c(100, 80, 50)
)
```

Manipulação de dados com `dplyr`

Selecionar colunas

```
select(vendas, produto, valor)

# select(variavel, coluna1, coluna2, coluna3 ...)
```

Filtrar linhas

```
filter(vendas, valor > 100)

vendas_filtradas <- filter(vendas, valor > 1000)
```

Criar colunas

```
vendas <- mutate(vendas, valor_total = valor * quantidade)
```

Ordenar dados com arrange()

Útil para classificar vendas por valor decrescente, comum em relatórios de performance.

```
arrange(vendas, desc(valor_total))
```

###Renomear colunas com rename() Para padronizar nomes em datasets financeiros.

```
rename(vendas, preco_unitario = valor)
vendas <- rename(vendas, preco_unitario = valor, Estado=uf)
```

Agrupar e resumir

```
vendas_resumo <- vendas %>%
  group_by(produto) %>%
  summarise(total = sum(valor_total))

vendas_resumo_decr <- arrange( vendas %>%
  group_by(produto) %>%
  summarise(total = sum(valor_total)), desc(total))

print(vendas_resumo)
print(vendas_resumo_decr)
```

```
vendas_resumo <- vendas %>%
  group_by(produto) %>%
  summarise(
    total_vendas = sum(valor_total),
    media_preco = mean(valor_total),
    max_venda = max(valor_total),
    contagem = n(),
    desvio_padrao = sd(valor_total),
    qtde_vendida = sum(quantidade)
  )
```

```r
print(vendas_resumo)

arrange(vendas_resumo, desc(qtde_vendida))
```

Pipes (%>%)

O operador %>% permite encadear comandos de forma mais legível:

```r
vendas %>%
  filter(quantidade > 10) %>%
  mutate(receita = preco_unitario * quantidade) %>%
  group_by(produto) %>%
  summarise(total_receita = sum(receita))
```

Mini Desafios

1. Importe o arquivo de vendas.
2. Filtre apenas as vendas acima de R$ 200.
3. Calcule a receita por produto.
4. Selecione apenas as colunas de produto e receita.

Tarefa para casa

1. Baixe uma planilha (.csv) com dados de despesas.
2. Importe para o R usando `read_csv()`.
3. Calcule o total de despesas por categoria.

Resumo da Aula

- tidyverse agrupa pacotes úteis para análise de dados
- Aprendemos a usar `read_csv`, `select`, `filter`, `mutate`, `group_by`, `summarise`
- Usamos pipes para encadear operações

Apêndice – Entendendo o Operador Pipe (%>%)

O operador **pipe** (%>%) permite encadear operações de forma mais legível. Ele envia o resultado da esquerda como **primeiro argumento** da função da direita.

Exemplo

```r
vendas %>%
  filter(valor > 200) %>%
  mutate(receita = valor * quantidade) %>%
  group_by(produto) %>%
  summarise(total_receita = sum(receita))
```

É equivalente a escrever de forma aninhada:

```
summarise(
  group_by(
    mutate(
      filter(vendas, valor > 200),
      receita = valor * quantidade
    ),
    produto
  ),
  total_receita = sum(receita)
)
```

Regras básicas

1. %>% envia o resultado da esquerda para o **primeiro argumento** da função à direita.
2. Se quiser colocar o objeto em outro lugar, use o **ponto (.)**:

```
c("R", "Python") %>% paste("é ótimo!", .)
# Resultado: "é ótimo! R" "é ótimo! Python"
```

Benefícios do pipe

- Código mais **legível**
- Facilita **sequência de operações**
- Muito usado com `dplyr`, `ggplot2`, `stringr` e outros pacotes do tidyverse

Parte Extra – Manipulação de Dados com Tidyverse (Aprofundamento)

Nesta seção adicionamos técnicas essenciais para **agregação**, **colapso de categorias**, **junções (merge/joins)**, **append/empilhamento**, **operações úteis do dia a dia** e **boas práticas**. Os exemplos usam datasets do `dplyr`/`tidyr` e o pacote `nycflights13` quando conveniente.

Pacotes

```
# Instale se necessário:
# install.packages(c("tidyverse", "nycflights13", "janitor", "lubridate", "forcats"))

library(tidyverse)
library(nycflights13)
library(janitor)
library(lubridate)
library(forcats)

# install.packages("conflicted")
# library(conflicted)
# conflict_prefer("filter", "dplyr")
# conflict_prefer("lag", "dplyr")
```

1) Agregação com `group_by()` + `summarise()` e amigos

```
# Exemplo com 'flights' (nycflights13)
dados <- flights |>
  select(year, month, day, carrier, origin, dest, distance, air_time, dep_delay, arr_delay)
```

```r
# Média de atraso por companhia
agg1 <- dados |>
  group_by(carrier) |>
  summarise(
    n_voos = n(),
    atraso_médio_chegada = mean(arr_delay, na.rm = TRUE),
    atraso_médio_partida = mean(dep_delay, na.rm = TRUE),
    .groups = "drop"
  )

# Agregando por múltiplas chaves
agg2 <- dados |>
  group_by(origin, carrier) |>
  summarise(
    n_voos = n(),
    dist_média = mean(distance, na.rm = TRUE),
    p95_atraso_chegada = quantile(arr_delay, 0.95, na.rm = TRUE),
    .groups = "drop"
  )

# Usando across() para resumir várias colunas ao mesmo tempo
agg3 <- dados |>
  group_by(carrier) |>
  summarise(
    across(c(distance, dep_delay, arr_delay, air_time),
           list(media = ~mean(.x, na.rm = TRUE), sd = ~sd(.x, na.rm = TRUE)),
           .names = "{.col}_{.fn}"),
    .groups = "drop"
  )

# Contagens rápidas: count()/tally()/add_tally()
contagens <- dados |>
  count(carrier, sort = TRUE) # equivale a group_by(carrier) |> summarise(n = n())
```

Dicas rápidas de agregação
- `n()` conta linhas dentro de `summarise()` ou `mutate()` no contexto de grupos.
- `n_distinct(x)` conta valores distintos de `x`.
- Para pesos: `weighted.mean(x, w, na.rm = TRUE)`.
- Para remover a informação de agrupamento ao final: use `.groups = "drop"` no `summarise()`.

2) "Colapsar" categorias com `forcats`

Às vezes precisamos **agrupar** ou **reduzir** categorias raras para análises e visualizações.

```r
# Exemplo com 'starwars'
cat_df <- starwars |>
  select(name, species, homeworld) |>
  clean_names()

# Colapsando espécies raras em "Outras"
cat_df2 <- cat_df |>
  mutate(
    species_lumped = fct_lump_n(f = factor(species), n = 5, other_level = "Outras")
  )
```

```r
# Colapsando manualmente categorias (mapeamento controlado)
cat_df3 <- cat_df |>
  mutate(
    species_colapsada = fct_collapse(
      factor(species),
      Human = c("Human"),
      Droid = c("Droid"),
      Wookiee = c("Wookiee"),
      # Tudo que não estiver explícito pode virar "Outras"
      Outras = setdiff(levels(factor(species)), c("Human","Droid","Wookiee"))
    )
  )
```

Quando usar: antes de gráficos (ex.: `ggplot2`) para evitar dezenas de barras finas; para sumarizações mais estáveis; para facilitar modelos.

3) "Merge" no Tidyverse: família `*_join()` vs. `merge()` base

`merge()` (base R) funciona, mas no dia a dia recomendam-se os *joins* do `dplyr`, que são mais legíveis e consistentes.

- `left_join(x, y, by=)` – mantém todas as linhas de `x` e adiciona colunas de `y`.
- `inner_join(x, y, by=)` – mantém apenas chaves presentes em **ambos**.
- `full_join(x, y, by=)` – união total, preenchendo com `NA` quando faltar.
- `right_join(x, y, by=)` – análogo ao `left_join`, mas mantendo linhas de `y`.
- `semi_join(x, y, by=)` – filtra `x` para linhas com chave presente em `y` (não adiciona colunas).
- `anti_join(x, y, by=)` – filtra `x` para chaves **não** presentes em `y`.

```r
# Tabelas de exemplo
tab1 <- tibble(id = 1:5, nota = c(7.5, 8.0, 6.9, 9.2, 7.8))
tab2 <- tibble(id = c(3,4,5,6), turma = c("A","B","B","C"))

left <- left_join(tab1, tab2, by = "id")
inner <- inner_join(tab1, tab2, by = "id")
full  <- full_join(tab1, tab2, by = "id")
semi  <- semi_join(tab1, tab2, by = "id")  # mantém id 3,4,5
anti  <- anti_join(tab1, tab2, by = "id")  # mantém id 1,2
```

Várias chaves, sufixos e conflitos de nomes

```r
x <- tibble(id = c(1,1,2), ano = c(2023,2024,2024), valor = c(10, 12, 9))
y <- tibble(id = c(1,2,2), ano = c(2024,2024,2023), valor = c(11, 10, 8))

j <- left_join(x, y, by = c("id","ano"), suffix = c("_x","_y"))
j
```

Boas práticas: sempre conferir cardinalidade da chave antes do join (1:1, 1:n, n:n). Quando precisar, use `distinct()` para garantir chaves únicas.

4) "Append"/empilhar/concatenar linhas e colunas

- **Linhas**: `bind_rows()` (equivale ao `rbind`, mas é mais tolerante a colunas ausentes).
- **Colunas**: `bind_cols()` (equivale ao `cbind` com reciclagem/avisos).
- **Operações de linhas em dplyr (v1.1+)**: `rows_append()`, `rows_insert()`, `rows_update()`, `rows_upsert()`, `rows_patch()`, `rows_delete()` – fazem mutating joins "declarativos".

```
a <- tibble(id = 1:3, x = c("a","b","c"))
b <- tibble(id = 4:5, x = c("d","e"))

linhas <- bind_rows(a, b)

colunas <- bind_cols(
  tibble(id = 1:3),
  tibble(y = c(10, 20, 30)),
  tibble(z = letters[1:3])
)

# rows_* (exige dplyr >= 1.1.0)
base <- tibble(id = 1:3, valor = c(100, 200, 300))
novas <- tibble(id = c(2,4), valor = c(250, 400))

atualizado <- rows_update(base, novas, by = "id")  # atualiza id=2; ignora id=4
inserido   <- rows_insert(base, novas, by = "id", conflict = "ignore") # insere id=4
upsert     <- rows_upsert(base, novas, by = "id")  # atualiza 2 e insere 4
```

5) Transformações úteis do dia a dia

Seleção/renomeação/organização

```
dados2 <- dados |>
  clean_names() |>
  rename(atraso_saida = dep_delay, atraso_chegada = arr_delay) |>
  relocate(carrier, origin, dest, .before = distance) |>
  select(carrier, origin, dest, starts_with("atraso"), ends_with("time"), everything())
```

Criações condicionais e vetorizadas

```
dados3 <- dados2 |>
  mutate(
    long_haul = if_else(distance >= 3000, "sim", "nao", missing = "NA"),
    atraso_cat = case_when(
      atraso_chegada <= 0 ~ "pontual",
      atraso_chegada <= 30 ~ "leve",
      atraso_chegada <= 120 ~ "moderado",
      TRUE ~ "severo"
    )
  )
```

Datas/tempos com `lubridate`

```
dados4 <- dados |>
  mutate(
    data = make_date(year, month, day),
    ym = floor_date(data, "month"),
```

```
    trimestre = quarter(data, with_year = TRUE)
  ) |>
  group_by(ym) |>
  summarise(n_voos = n(), atraso_médio = mean(arr_delay, na.rm = TRUE), .groups = "drop")
```

Tidyr para "arrumar" dados

- `pivot_longer()` – de colunas para linhas.
- `pivot_wider()` – de linhas para colunas.
- `separate()`/`unite()` – dividir e unir colunas.
- `nest()`/`unnest()` – trabalhar com listas por grupo.

```
# Ex.: largura -> longo
wide <- tibble(id = 1:3, jan = c(10,20,30), fev = c(5,7,9))
long <- wide |>
  pivot_longer(cols = jan:fev, names_to = "mes", values_to = "valor")

# Ex.: longo -> largura
wide2 <- long |>
  pivot_wider(names_from = mes, values_from = valor)
```

Operações em grupo adicionais

```
# pegar top-N por grupo
top_por_carrier <- dados |>
  group_by(carrier) |>
  slice_max(order_by = distance, n = 3, with_ties = FALSE) |>
  ungroup()

# lags/leads por grupo
lags <- dados |>
  arrange(carrier, month, day) |>
  group_by(carrier) |>
  mutate(
    atraso_lag = lag(arr_delay),
    atraso_lead = lead(arr_delay)
  ) |>
  ungroup()
```

Operações de conjunto com data frames

```
df1 <- tibble(id = c(1,2,3), x = c("a","b","c"))
df2 <- tibble(id = c(3,4),   x = c("c","d"))

uniao      <- union(df1, df2)      # elementos de df1   df2
intersecao <- intersect(df1, df2)  # elementos em comum
diferenca  <- setdiff(df1, df2)    # em df1 e não em df2
```

6) Comparando com `merge()` (base R)

```
# Equivalências (aproximadas):
# left_join(x, y, by="chave")  ~ merge(x, y, by="chave", all.x = TRUE)
# inner_join(x, y, by="chave") ~ merge(x, y, by="chave")  # all=FALSE padrão
# full_join(x, y, by="chave")  ~ merge(x, y, by="chave", all = TRUE)
# right_join(x, y, by="chave") ~ merge(x, y, by="chave", all.y = TRUE)
```

```r
merge_left <- merge(tab1, tab2, by = "id", all.x = TRUE)
```

7) Checklist de boas práticas

- Verifique **tipos** antes de unir: use `glimpse()` e `skimr::skim()` (se usar `skimr`).
- Garanta **unicidade** de chaves quando sua lógica exigir (use `count(chave)` + `filter(n > 1)`).
- Trate **faltantes** (`NA`) explicitamente: `replace_na()`, `coalesce()`.
- Documente suposições no código e use nomes claros para objetos.
- Para performance em dados grandes, considere: `arrow`, `duckdb`, `dtplyr` (data.table com gramática dplyr).

8) Mini–exercícios guiados (com respostas)

(a) Crie uma tabela com o atraso médio de chegada por `origin` e `month`.
(b) Selecione os 5 destinos com maior distância média.
(c) Faça um `left_join()` entre um resumo de `flights` por `carrier` e a tabela `airlines`, trazendo o nome da companhia.

```r
# Respostas (exemplos)
resp_a <- flights |>
  group_by(origin, month) |>
  summarise(atraso_médio = mean(arr_delay, na.rm = TRUE), .groups = "drop")

resp_b <- flights |>
  group_by(dest) |>
  summarise(dist_média = mean(distance, na.rm = TRUE), .groups = "drop") |>
  slice_max(dist_média, n = 5)

resp_c <- flights |>
  group_by(carrier) |>
  summarise(n_voos = n(), atraso_médio = mean(arr_delay, na.rm = TRUE), .groups = "drop") |>
  left_join(airlines, by = "carrier")
```

Apêndice – Tabela de referência rápida (cheatsheet de verbos)

- **Selecionar/Ordenar**: `select()`, `rename()`, `relocate()`, `arrange()`, `distinct()`
- **Criar/Transformar**: `mutate()`, `if_else()`, `case_when()`, `across()`, `rowwise()`
- **Agregação**: `group_by()`, `summarise()`, `count()`, `tally()`
- **Joins**: `left_join()`, `inner_join()`, `full_join()`, `right_join()`, `semi_join()`, `anti_join()`
- **Empilhar/Combinar**: `bind_rows()`, `bind_cols()`, `rows_*()`
- **Arrumar dados**: `pivot_longer()`, `pivot_wider()`, `separate()`, `unite()`, `nest()`, `unnest()`
- **Fatores**: `fct_lump_n()`, `fct_collapse()`

- **Datas**: `make_date()`, `floor_date()`, `quarter()`
- **Conjuntos**: `union()`, `intersect()`, `setdiff()`