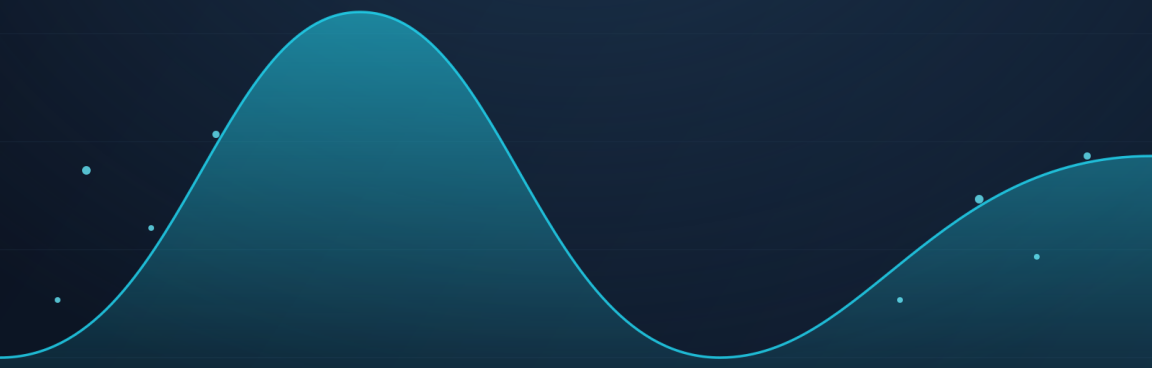
ESTATÍSTICA • R • DADOS



Estatística Aplicada com R

para quem conhece Python

Teoria estatística e aplicação prática em R,
módulo a módulo — da descritiva à bayesiana.

16 capítulos completos | código R pronto p/ rodar | dados reais + sintéticos

Sérgio Berlotto Jr.

TEORIA & APLICAÇÃO PRÁTICA EM R

Estatística Aplicada com R

para quem conhece Python

Dezesseis capítulos de estatística aplicada — da descritiva à bayesiana — com teoria matemática, premissas a verificar antes de usar cada técnica, e implementação completa em R.

Sérgio Berlotto Jr.

Estatística Aplicada com R — para quem conhece Python

© 2026 Sérgio Berlotto Jr. Todos os direitos reservados.

Nenhuma parte desta obra pode ser reproduzida ou transmitida por qualquer forma ou meio, eletrônico ou mecânico, incluindo fotocópia e gravação, sem permissão por escrito do autor, exceto citações breves em resenhas e materiais de apoio ao próprio estudo.

Os exemplos de código, datasets e scripts complementares citados ao longo do texto estão disponíveis no repositório público:
github.com/berlotto/estatistica-aplicada-r-materiais

Primeira edição — 2026.

Sumário

FUNDAMENTOS

00 Sintaxe Básica do R — Para quem já sabe Python e Pandas 12

01 Gráficos com ggplot2 31

02 Estatística Descritiva 52

03 Dados Faltantes 71

MODELAGEM

04 Correlação e Regressão — Estatística Bivariada 82

05 Regressão Logística — GLM Binomial 116

06 Testes Não-Paramétricos 132

07 Cluster e PCA — Estatística Multivariada 156

INFERÊNCIA

08 Distribuições e Intervalos de Confiança — Estatística Inferencial I 179

09 Testes de Hipótese — Estatística Inferencial II 212

APLICAÇÕES

10 Séries Temporais 248

11 Estatística Espacial 270

TÓPICOS AVANÇADOS

12 GLM Generalizado — Poisson e Gamma 297

13 Análise de Sobrevivência 309

14 Estatística Bayesiana — Introdução 320

15 Delineamento Experimental — DOE 331

Resumo dos capítulos

Uma sinopse do que cada capítulo cobre, para você navegar direto ao que precisa. A sequência segue as dependências conceituais: Fundamentos → Modelagem → Inferência → Aplicações → Tópicos Avançados.

00 Sintaxe Básica do R · Para quem já sabe Python e Pandas

Fecha a lacuna de sintaxe de superfície para quem já pensa em Python/Pandas: vetorização, atribuição com `<-`, estruturas de dado do R, indexação, o pipe (`%>%` e `|>`), a gramática do dplyr e fórmulas estatísticas (`y ~ x`).

01 Gráficos com ggplot2

A gramática dos gráficos (Grammar of Graphics) que fundamenta o ggplot2: dados, estética, geometrias, escalas e camadas. Construção em camadas, facetas, temas e customização — termina com uma biblioteca pessoal de templates reutilizáveis.

02 Estatística Descritiva

A base de tudo: distribuição de frequência, histograma simples e acumulado, leitura de forma e assimetria, média vs. mediana, dispersão via variância/desvio padrão/quartis/percentis, e coeficiente de variação.

03 Dados Faltantes

Missing estrutural vs. os três mecanismos de Rubin (MCAR, MAR, MNAR), consequência de listwise deletion, imputação simples e múltipla (`mice`), e como reportar missing com transparência.

04 Correlação e Regressão · Estatística Bivariada

Correlação de Pearson, regressão linear simples e múltipla, multicolinearidade (VIF), regressão polinomial, diagnóstico formal (Breusch-Pagan, distância de Cook, DFBETAS, Box-Cox) e intervalo de confiança vs. de predição.

05 Regressão Logística · GLM Binomial

Regressão logística (GLM binomial): log-odds, função sigmoide, máxima verossimilhança, odds ratio, matriz de confusão, curva ROC/AUC e separação quase perfeita como sinal de vazamento de dado.

06 Testes Não-Paramétricos

A lógica geral dos testes não-paramétricos: Mann-Whitney, Wilcoxon signed-rank, Kruskal-Wallis, correlação de Spearman, e o critério de decisão entre a via paramétrica e a não-paramétrica.

07 Cluster e PCA · Estatística Multivariada

Cluster hierárquico (métricas de distância, linkage, dendrogramas) e PCA (autovalores/autovetores, scree plot, biplots) — o primeiro módulo com múltiplas variáveis simultâneas.

08 Distribuições e Intervalos de Confiança · Estatística Inferencial I

Amostragem, Teorema de Bayes, distribuições discretas e contínuas, Teorema do Limite Central, intervalos de confiança e determinação de tamanho de amostra via power analysis (`pwr`).

09 Testes de Hipótese · Estatística Inferencial II

O núcleo da estatística aplicada: lógica de teste de hipótese, p-valor, teste t, ANOVA com Tukey HSD, qui-quadrado, Teste Exato de Fisher, tamanho de efeito e bootstrap/permutação.

10 Séries Temporais

Séries temporais com `tsibble` / `fable` : decomposição, estacionariedade, ACF/PACF, ARIMA automático, diagnóstico de resíduos e avaliação de previsão.

11 Estatística Espacial

Estatística espacial com `sf` : Moran's I, Geary's C, LISA, matrizes de vizinhança, krigagem e regressão espacial.

12 GLM Generalizado · Poisson e Gamma

GLM como família unificada: regressão de Poisson (contagens, sobredispersão) e regressão Gamma (contínua positiva assimétrica).

13 Análise de Sobrevivência

Análise de sobrevivência: censura à direita, Kaplan-Meier, log-rank test, modelo de Cox e verificação de riscos proporcionais.

14 Estatística Bayesiana · Introdução

Estatística bayesiana: prior/verossimilhança/posterior, conjugação Beta-Binomial, intervalo de credibilidade, MCMC via `brms` .

15 Delineamento Experimental · DOE

Delineamento experimental: DIC, DBC, Quadrado Latino, dimensionamento de repetições via `pwr.anova.test()` e o pacote `agricolae` .

Como usar este livro

Todo o código em R deste livro é executável e foi escrito para ser copiado e rodado exatamente como aparece — nenhum trecho é pseudocódigo. Cada capítulo cita explicitamente os pacotes CRAN necessários na primeira vez em que são usados (`install.packages(...)`).

Datasets e scripts complementares

Os datasets e os scripts auxiliares referenciados ao longo dos capítulos estão publicados neste repositório, na mesma estrutura de pastas usada no texto:

```
github.com/berlotto/estatistica-aplicada-r-materiais

├── dados/
│   ├── varejo_sintetico.csv
│   └── painel_regional_varejo.csv
└── scripts/
    ├── gerar_datasets_sinteticos.R
    └── 01_templates_ggplot2.R
```

Clone ou baixe o repositório e mantenha a pasta `dados/` e `scripts/` no mesmo diretório do seu projeto R — os caminhos relativos usados no código (`read.csv("dados/varejo_sintetico.csv")` , por exemplo) já assumem essa estrutura. Datasets reais têm sua fonte específica documentada no cabeçalho do respectivo capítulo (INMET, IBGE/SIDRA, UCI Machine

Learning Repository, entre outros). Onde um dataset real com a estrutura necessária não estava disponível, o capítulo constrói um dataset sintético com estrutura conhecida, documentada com transparência.

PARTE 1

Fundamentos

Sintaxe do R para quem já pensa em Python/Pandas, visualização com ggplot2, estatística descritiva e o tratamento correto de dados faltantes — a base sobre a qual todo o resto se apoia.

MÓDULO 00 · FUNDAMENTOS

Sintaxe Básica do R

Para quem já sabe Python e Pandas

1. Modelo mental: R é vetorizado por padrão, Python não é

Em Python, uma lista não sabe fazer aritmética elemento a elemento (`[1,2,3] * 2` duplica a lista, não multiplica cada item) — você precisa de NumPy/Pandas para isso. Em R, **todo objeto numérico é um vetor**, e operações aritméticas são vetorizadas por padrão, sem biblioteca nenhuma:

```
x <- c(1, 2, 3)
x * 2        # 2 4 6 — elemento a elemento, sem precisar de numpy
x + c(10, 20, 30)   # 11 22 33
```

Não existe "escalar" de verdade em R: `5` é um vetor de tamanho 1. Essa é a razão pela qual R não precisa de uma distinção `int` vs `np.array` como Python/NumPy — tudo já é array. É também a raiz do erro mais comum de quem vem de Python: escrever um `for` explícito onde uma operação vetorizada já resolveria (ver seção 7).

Indexação começa em 1, não em 0. `x[1]` é o primeiro elemento. Isso é puramente convencional (herdado do Fortran/S), mas é a fonte mais comum de erro off-by-one na transição.

---

2. Atribuição: `<-` em vez de `=`

```
receita_media <- mean(varejo$receita_mensal)
```

`<-` é o operador idiomático de atribuição em R (não `=`). `=` também funciona para atribuição no nível superior, mas a convenção da comunidade — e o que você vai ver em 100% do código dos módulos 1-13 — é `<-`. A razão histórica é que `=` fica reservado para passar argumentos nomeados dentro de uma chamada de função:

```
# = aqui é argumento nomeado, não atribuição
round(3.14159, digits = 2)

# <- é atribuição de variável
resultado <- round(3.14159, digits = 2)
```

Misturar os dois (`=` para atribuir) roda, mas confunde leitura porque o mesmo símbolo passaria a significar duas coisas diferentes dependendo do contexto. Existe também `->` (atribuição da esquerda para direita, ex.: `mean(x) -> media`) — raro, aparece ocasionalmente ao final de uma pipe longa, mas não é o padrão usado no curso.

Atalho no RStudio: `Alt` + `-` digita `<-` automaticamente.

---

3. Estruturas de dado que você vai ver

R	Equivalente conceitual em Python	Observação
`vector` (`c(1, 2, 3)`)	`list` homogênea / `np.array` 1D	Todo elemento precisa ser do mesmo tipo — misturar força coerção (`c(1, "a")` vira tudo string)
`list`	`list` heterogênea / `dict` sem chave obrigatória	Pode misturar tipos e tamanhos: `list(nome = "Loja A", vendas = c(100, 200))`

R	Equivalente conceitual em Python	Observação
`data.frame` / `tibble`	`pandas.DataFrame`	`tibble` (pacote `tibble`, parte do `tidyverse`) é a versão mais estrita e com `print` mais legível do `data.frame` base
`factor`	categórico do Pandas (`pd.Categorical`)	Guarda níveis (`levels`) fixos — crítico para ordem correta em boxplot/ANOVA, ver Módulo 1 e 9
`NA`	`None` / `np.nan` (parcialmente)	`NA` é o "faltante" universal do R — existe `NA_integer_`, `NA_real_`, `NA_character_` internamente, mas na prática você só escreve `NA`
`NULL`	`None`	Representa "ausência do objeto", não "valor faltante dentro de um vetor" — `length(NULL)` é `0`, `length(NA)` é `1`
`NaN`	`np.nan`	Resultado de operação matematicamente indefinida (`0/0`), diferente de `NA` (dado que não foi coletado)

```
x <- c(10, NA, 30)
mean(x)                    # NA — R propaga NA por padrão, não ignora
silenciosamente
mean(x, na.rm = TRUE)
# 20 — remove NA antes de calcular (equivalente a dropna implícito do
Pandas)
```

Esse comportamento (propagar `NA` em vez de ignorar) é intencional — R força você a decidir explicitamente o que fazer com o faltante, em vez de assumir por você. Pandas historicamente ignora `NaN` em `.mean()` por padrão, o que é a direção oposta.

---

4. Indexação: `[ ]`, `[[ ]]` e `$`

Três operadores de acesso, com comportamento diferente — essa é provavelmente a maior fonte de confusão vinda do Pandas.

```
varejo$receita_mensal        # coluna inteira, por nome — equivalente a
varejo['receita_mensal']
varejo[["receita_mensal"]]   # mesma coisa que $, mas aceita nome como
string/variável
varejo["receita_mensal"]     # retorna um data.frame de 1 coluna, NÃO o
vetor — diferença sutil e comum de errar
varejo[1, ]                  # primeira linha, todas as colunas —
equivalente a varejo.iloc[0]
varejo[, "porte"]            # todas as linhas, coluna porte
varejo[varejo$porte == "Grande", ]   # filtro lógico — equivalente a
varejo[varejo['porte']=='Grande'] no Pandas
```

Regra prática para listas (inclusive `data.frame`, que é uma lista de colunas por baixo): - `[ ]` (colchete simples) sempre retorna o **mesmo tipo de contêiner** — uma lista fatiada com `[ ]` continua sendo lista, mesmo com 1 elemento. - `[[ ]]` (colchete duplo) **extrai o conteúdo** de dentro do contêiner — de uma lista com `[[ ]]` você tira o vetor/objeto de dentro, não uma lista de 1 elemento. - `$` é açúcar sintático para `[[ ]]` por nome (não funciona com posição numérica, não funciona com nome guardado em variável).

```
lista <- list(a = 1:3, b = "texto")
lista["a"]     # list de 1 elemento — ainda é lista
lista[["a"]]   # 1 2 3 — o vetor de fato
lista$a        # 1 2 3 — mesma coisa que [["a"]], só que sintaticamente
fixo (não aceita variável)

nome_col <- "a"
lista[[nome_col]]   # funciona — [[ ]] aceita variável com o nome
lista$nome_col      # NÃO funciona como esperado — $ procura literalmente
a coluna "nome_col", não o valor da variável
```

Essa última linha é o erro mais comum ao escrever função genérica que recebe nome de coluna como parâmetro (ver Módulo 1, seção sobre NSE em `aes()`, e Módulo 2 na função `cv()`).

---

5. Funções: `function(...) { }`

```
cv <- function(x, na_rm = TRUE) {
  (sd(x, na.rm = na_rm) / mean(x, na.rm = na_rm)) * 100
}

cv(varejo$receita_mensal)
```

Pontos que diferem de Python:

- **Retorno implícito**: a última expressão avaliada no corpo da função é o valor retornado — não precisa de `return()` explícito (embora `return()` exista e seja usado para saída antecipada, ex.: dentro de um `if` no meio da função).
- **Argumentos com default** funcionam igual a Python (`na_rm = TRUE` acima) — parâmetro opcional se não passado.
- **Argumentos nomeados podem vir em qualquer ordem** na chamada: `cv(na_rm = FALSE, x = varejo$receita_mensal)` funciona igual a `cv(varejo$receita_mensal, FALSE)`.
- `...` **(reticências/dots)**: captura um número variável de argumentos extras e repassa adiante — equivalente a `*args, **kwargs` combinados

em um símbolo só. Aparece com frequência em wrappers, ex. `moda <- function(x, ...) { table(x, ...) }`.

- Funções são **objetos de primeira classe** como em Python — podem ser passadas como argumento (é assim que `sapply(dados, sd)` funciona: `sd` é passada como valor, não chamada ali).

---

6. Pipe: `%>%` e `|>`

O pipe é a construção sintática mais usada no material inteiro (dezenas de ocorrências por módulo). Ele resolve o mesmo problema do method chaining do Pandas (`df.query(...).groupby(...).agg(...)`), mas para funções que não são métodos de objeto.

```
# Sem pipe — leitura de dentro pra fora, difícil de acompanhar
summarise(group_by(filter(varejo, porte == "Grande"), regiao), receita_med
ia = mean(receita_mensal))

# Com pipe — leitura de cima pra baixo, na ordem em que as operações
acontecem
varejo %>%
  filter(porte == "Grande") %>%
  group_by(regiao) %>%
  summarise(receita_media = mean(receita_mensal))
```

Mecanicamente: `x %>% f(y)` é reescrito como `f(x, y)` — o pipe insere o lado esquerdo como **primeiro argumento** da função do lado direito. Encadear pipes é encadear essa substituição sucessivamente.

Existem dois pipes no ecossistema R, e o material mistura predominantemente o primeiro:

Operador	Origem	Observação
`%>%`	pacote `magrittr`, carregado junto com `dplyr` / `tidyverse`	O que você vê na esmagadora maioria do código dos módulos 1-13. Aceita `.` como placeholder explícito para casos em que o valor não deveria ir para o primeiro argumento (`x %>% f(y, .)`)
`|>`	nativo do R (a partir da versão 4.1, 2021), sem precisar de pacote	Aparece uma única vez em todo o material (Módulo 10). Mais simples internamente, mas com menos recursos (placeholder `_` exige nome de argumento explícito, sintaxe mais nova e mais rígida)

Na prática, para este curso: leia os dois, mas **escreva** `%>%` — é o que mantém consistência com o resto do código já produzido, e é o que domina em código R real de produção hoje.

---

7. Controle de fluxo: `if`, `for`, e por que evitá-los quando dá

```
if (media_area > 200) {
  print("Área média acima do limiar")
} else {
  print("Área média dentro do esperado")
}

for (regiao in unique(varejo$regiao)) {
  cat(regiao, ":", mean(varejo$receita_mensal[varejo$regiao == regiao]),
"\n")
}
```

Sintaxe é próxima de Python (parênteses na condição, chaves para bloco — sem dois-pontos, sem indentação obrigatória, embora indentar seja convenção universal). A diferença de fundo é de **uso**, não de sintaxe: como tudo em R é vetorizado (seção 1), um `for` explícito é frequentemente sintoma de estar resolvendo o problema do jeito Python dentro de R. Prefira, nesta ordem:

1. Operação vetorizada direta (`varejo$receita_mensal * 1.1`)
2. `ifelse()` vetorizado para condicional elemento a elemento — **não** `if()` dentro de loop
3. Família `apply` / `sapply` (seção 8) ou verbos `dplyr` (seção 9) para agregação por grupo
4. `for` explícito só quando cada iteração depende do resultado da anterior (ex.: simulação sequencial, replicação Monte Carlo com estado acumulado — usado nos módulos 6 e 7)

Armadilha clássica: `if()` espera um único `TRUE` / `FALSE` , não um vetor — passar um vetor gera erro (ou warning, dependendo da versão) porque R não sabe qual dos elementos usar como condição. `ifelse(condicao_vetor, valor_se_true, valor_se_false)` é o equivalente vetorizado, e é o que você deve usar para gerar uma coluna nova condicional:

```
varejo$porte_binario <- ifelse(varejo$porte == "Grande", "Grande", "Não-
grande")
```

`case_when()` do `dplyr` (seção 9) é a extensão natural disso para mais de duas categorias.

---

8. Família `apply`: `sapply`, `lapply`, `apply`

Equivalente funcional ao `map()` /list comprehension do Python — aplica uma função a cada elemento de uma estrutura, sem loop explícito.

```
# sapply: aplica função a cada elemento de um vetor/lista, tenta
simplificar o resultado (retorna vetor se possível)
sapply(varejo[c("area_m2", "receita_mensal", "ticket_medio")], sd)
#    area_m2 receita_mensal   ticket_medio
#     134.82        2481.09          17.63

# lapply: igual, mas sempre retorna list (nunca simplifica) — mais
previsível, pior pra imprimir
lapply(varejo[c("area_m2", "receita_mensal")], mean)

# apply: aplica por linha (MARGIN=1) ou por coluna (MARGIN=2) de uma
matriz/data.frame
apply(varejo[c("area_m2", "num_funcionarios")], 2, mean)   # MARGIN=2 =
por coluna
```

Diferença-chave entre os três: `sapply` tenta devolver algo "achatado" (vetor nomeado, como no exemplo acima); `lapply` sempre devolve `list` (útil quando cada resultado tem tamanho diferente, então não daria pra simplificar em vetor/matriz de qualquer forma); `apply` é para estrutura retangular (matriz/data.frame) quando você quer aplicar por linha ou por coluna explicitamente, controlado pelo argumento `MARGIN`.

Na prática do curso: para agregação de data.frame por grupo, os verbos `dplyr` (próxima seção) são mais legíveis e mais usados que `apply` / `sapply` — a família `apply` aparece principalmente para aplicar a mesma função a várias colunas de uma vez (como no exemplo `sd` acima) ou dentro de funções auxiliares mais genéricas (Módulos 7, 8, 9, 10).

---

9. `dplyr` — a gramática de manipulação de dados

Se ggplot2 é a "Grammar of Graphics", `dplyr` é a gramática equivalente para transformação de dados — um pequeno conjunto de verbos combináveis via pipe, cada um resolvendo uma operação específica. É o que substitui a maior parte do que você faria com indexação `[ ]` / `.loc` / `.groupby` do Pandas.

Verbo `dplyr`	Equivalente Pandas	O que faz
`filter(condicao)`	`df.query(...)` / `df[condicao]`	Mantém linhas que satisfazem a condição
`select(colunas)`	`df[['col1','col2']]`	Mantém só as colunas listadas
`mutate(nova_col = expr)`	`df.assign(nova_col=...)`	Cria/sobrescreve coluna
`group_by(coluna)`	`df.groupby('coluna')`	Agrupa para agregação seguinte — não retorna resultado sozinho, precisa de `summarise` depois
`summarise(estatistica = fn(col))`	`.agg(...)` após `groupby`	Colapsa cada grupo em uma linha de resumo
`arrange(coluna)`	`df.sort_values('coluna')`	Ordena linhas

Verbo `dplyr`	Equivalente Pandas	O que faz
`left_join(outro_df, by = "chave")`	`df.merge(outro, on='chave', how='left')`	Junção de tabelas
`pivot_longer(cols, names_to, values_to)`	`pd.melt(df, ...)`	Formato largo → longo
`case_when(cond1 ~ val1, cond2 ~ val2, TRUE ~ default)`	`np.select([...], [...], default=...)`	`ifelse` encadeado para múltiplas condições

```
varejo %>%
  filter(porte != "Pequena") %>%
  group_by(regiao, tipo_loja) %>%
  summarise(
    receita_media = mean(receita_mensal),
    n_lojas = n(),
    .groups = "drop"
  ) %>%
  arrange(desc(receita_media))
```

Detalhe que confunde quem vem do Pandas: dentro de `filter()`, `mutate()`, `summarise()`, os nomes de coluna são escritos **sem aspas e sem `$`** — é a mesma NSE (Non-Standard Evaluation) mencionada no Módulo 1 para `aes()`. O `dplyr` resolve `regiao` procurando essa coluna dentro do data.frame que veio pelo pipe. `n()` (dentro de `summarise`) conta linhas do grupo atual, sem argumento — equivalente a `.size()` do Pandas agrupado.

`.groups = "drop"` em `summarise()` depois de `group_by()` com mais de uma coluna evita um warning do `dplyr` sobre agrupamento residual — boa prática, não estritamente obrigatório, mas aparece no código dos módulos por essa razão.

---

10. Fórmulas estatísticas: `y ~ x`

Sintaxe exclusiva de R, sem equivalente direto em Pandas/statsmodels na mesma forma compacta — mas é como praticamente toda função de modelo (`lm`, `glm`, `aov`, `wilcox.test`, testes do Módulo 6, 9, 12, 13, 15) especifica variável resposta e preditoras.

```
modelo <- lm(receita_mensal ~ area_m2, data =
varejo)                        # y ~ x — regressão simples
modelo2 <- lm(receita_mensal ~ area_m2 + fluxo_clientes_mensal, data =
varejo)  # y ~ x1 + x2 — múltipla
modelo3 <- lm(receita_mensal ~ area_m2 * tipo_loja, data =
varejo)           # * inclui termo de interação
modelo4 <- lm(receita_mensal ~ ., data =
varejo)                              # . = "todas as outras colunas"
modelo5 <- lm(receita_mensal ~ area_m2 - 1, data =
varejo)                   # -1 remove o intercepto
```

Leitura: `~` separa resposta (esquerda) de preditoras (direita) — "receita_mensal é modelada em função de area_m2". `+` adiciona preditor (aditivo, sem interação); `*` adiciona preditor **e** a interação entre eles (equivalente a escrever `area_m2 + tipo_loja + area_m2:tipo_loja`, onde `:` isolado adiciona *só* a interação, sem os efeitos principais); `.` é atalho para "todas as colunas do `data` exceto a resposta"; `-1` (ou `+ 0`) força o modelo a passar pela origem, removendo o intercepto — raro, usado só quando há razão teórica específica.

A fórmula não é avaliada imediatamente — ela é um objeto R (`class(receita_mensal ~ area_m2)` retorna `"formula"`) que a função de modelo interpreta internamente, decidindo como montar a matriz de design.

---

11. Namespace: `pacote::funcao()`

```
broom::tidy(modelo)
car::vif(modelo)
dunn.test::dunn.test(varejo$receita_mensal, varejo$regiao)
```

`::` acessa uma função de um pacote **sem precisar rodar `library(pacote)` antes** — usado tipicamente quando (a) você só vai usar uma função isolada daquele pacote e não quer carregar o namespace inteiro, ou (b) dois pacotes carregados têm função de mesmo nome, e `::` desambigua explicitamente qual delas você quer (ex.: `dplyr::filter()` vs `stats::filter()`, que existem os dois e fazem coisas completamente diferentes). Nos módulos 3, 3B, 5, 8, 9, 13 aparece principalmente pela primeira razão — chamar uma função pontual de pacote auxiliar sem poluir o ambiente com `library()`.

---

12. Inspeção e saída: `print`, `cat`, `str`, `head`, `summary`

```
varejo                  # no console/RMarkdown, digitar o objeto chama
print() implicitamente
print(varejo)
# explícito — necessário dentro de função ou loop, onde o auto-print não
dispara

cat("Média:", mean(varejo$receita_mensal), "\n")   # concatena e imprime
sem aspas, sem formatação automática
str(varejo)              # estrutura do objeto — tipos de coluna,
primeiras linhas — equivalente a df.info()
head(varejo, 5)           # primeiras 5 linhas — igual df.head()
summary(varejo$receita_mensal)   # min/Q1/mediana/média/Q3/max —
equivalente a df.describe() por coluna

paste("Loja", 1:3)                    # "Loja 1" "Loja 2" "Loja 3" —
concatena com espaço por padrão
paste0("Loja_", 1:3)
# "Loja_1" "Loja_2" "Loja_3" — paste com separador vazio
sprintf("Receita: R$ %.2f", 1234.5)
# "Receita: R$ 1234.50" — formatação estilo printf, usado quando precisão
importa
```

`print()` vs `cat()`: `print()` mostra a representação do objeto (com aspas em string, índices, etc.); `cat()` concatena e imprime texto cru, sem essas marcações — o que você normalmente quer para mensagem legível ao usuário.

---

13. Diferenças-chave vindas de Python (Pandas)

Situação	Python/Pandas	R
Índice de vetor/lista	Começa em 0	Começa em 1
Atribuição	`=`	`<-` (convenção; `=` funciona mas não é idiomático)
Selecionar coluna	`df['col']` ou `df.col`	`df$col`, `df[["col"]]`, ou `df["col"]` (retorna data.frame, não vetor!)
Faltante	`None` / `np.nan`, ignorado por padrão em `.mean()`	`NA`, propagado por padrão — precisa `na.rm = TRUE` explícito
Encadear operações	method chaining (`df.query().groupby().agg()`)	pipe (`df %>% filter() %>% group_by() %>% summarise()`)
Aplicar função a coluna(s)	`df.apply()`, `df['col'].map()`	`sapply()`, `lapply()`, ou verbos `dplyr`
Especificar modelo	`smf.ols('y ~ x', data=df)` (statsmodels já imita a fórmula do R)	`lm(y ~ x, data = df)` — a sintaxe de fórmula é nativa do R, statsmodels a copiou

Situação	Python/Pandas	R
Namespace de função	`import pacote; pacote.funcao()`	`library(pacote); funcao()` ou `pacote::funcao()` sem carregar tudo
Categórico	`pd.Categorical`, `.astype('catego`	`factor()`, com `levels` explícitos controlando ordem
Docstring/ajuda	`help(funcao)` / `funcao.__doc__`	`?funcao` ou `help(funcao)` — abre documentação formal, quase sempre com exemplo executável no final

---

14. Exercícios

Use `varejo_sintetico.csv`. Carregue `dplyr` antes de começar.

Exercício 1 — `<-` vs `=`, e leitura de erro comum

O código abaixo tem um bug de digitação. Identifique e corrija:

```
media_receita = mean(varejo$receita_mensal, digits = 2)
```

Solução:

```
media_receita <- round(mean(varejo$receita_mensal), digits = 2)
```

Dois problemas: (1) `mean()` não tem argumento `digits` — quem tem esse argumento é `round()`, então é preciso envolver o resultado; (2) usar `=` para atribuição no lugar de `<-` roda sem erro (não é bug de execução), mas foge

da convenção — o ponto pedagógico aqui é que R **aceita** `=` para atribuição de topo, então esse tipo de erro não é sinalizado pelo interpretador, só por revisão de código/convenção.

Exercício 2 — `[ ]` vs `[[ ]]`

Sem rodar no console, preveja o tipo de retorno (vetor ou data.frame) de cada linha, depois confirme com `class()`:

```
a <- varejo["ticket_medio"]
b <- varejo[["ticket_medio"]]
c <- varejo$ticket_medio
```

Solução:

```
class(a)   # "data.frame" — colchete simples preserva o contêiner
class(b)   # "numeric" — colchete duplo extrai o conteúdo
class(c)   # "numeric" — $ é equivalente a [[ ]] por nome fixo
```

`a` e `b` / `c` parecem a mesma coisa ao imprimir no console (ambos mostram os valores da coluna), mas `a` é uma tabela de 1 coluna e `b` / `c` são o vetor puro — a diferença aparece na hora de usar o resultado dentro de outra função que espera vetor (ex.: `mean(a)` falha, `mean(b)` funciona).

Exercício 3 — vetorização vs loop

Reescreva o loop abaixo como operação vetorizada, sem `for`:

```
resultado <- c()
for (i in 1:nrow(varejo)) {
  if (varejo$porte[i] == "Grande") {
    resultado[i] <- "Prioritária"
  } else {
    resultado[i] <- "Padrão"
  }
}
```

Solução:

```
resultado <- ifelse(varejo$porte == "Grande", "Prioritária", "Padrão")
```

Além de mais curto, a versão vetorizada evita o antipadrão de fazer `resultado[i] <- valor` dentro de loop sem pré-alocar o vetor (`resultado <- c()` cresce a cada iteração, recriando o vetor inteiro em memória a cada passo — caro para `n` grande). `ifelse()` roda em C internamente, sem esse custo.

Exercício 4 — pipe: reescrever sem `%>%` e depois com

Dado:

```
varejo %>%
  filter(regiao == "Sul") %>%
  group_by(tipo_loja) %>%
  summarise(receita_media = mean(receita_mensal), .groups = "drop") %>%
  arrange(desc(receita_media))
```

Reescreva sem usar `%>%` (chamadas aninhadas), depois explique qual versão você entenderia mais rápido lendo pela primeira vez e por quê.

Solução:

```
arrange(
  summarise(
    group_by(
      filter(varejo, regiao == "Sul"),
      tipo_loja
    ),
    receita_media = mean(receita_mensal), .groups = "drop"
  ),
  desc(receita_media)
)
```

A versão aninhada exige ler de dentro para fora (o `filter` mais interno acontece primeiro, mas está mais distante visualmente do início da leitura) — para 4 operações isso já é difícil de acompanhar; com 6-7 operações (comum nos módulos 6-9) fica impraticável. A versão com pipe segue a ordem de execução = ordem de leitura, de cima para baixo.

Exercício 5 — fórmula: interpretar sem rodar

Para cada fórmula abaixo, escreva em português o que o modelo está especificando (variável resposta, preditoras, se há interação, se há intercepto):

```
a: receita_mensal ~ area_m2 + tipo_loja
b: receita_mensal ~ area_m2 * porte
c: receita_mensal ~ . - regiao
```

Solução:

- `a` : resposta `receita_mensal` , preditoras `area_m2` e `tipo_loja` somadas de forma aditiva (sem interação) — o efeito de `area_m2` é assumido igual para todo `tipo_loja` .
- `b` : resposta `receita_mensal` , preditora `porte` e `area_m2` , **com** interação — o efeito de `area_m2` sobre `receita_mensal` pode variar conforme o `porte` (equivale a `area_m2 + porte + area_m2:porte`).
- `c` : resposta `receita_mensal` , todas as outras colunas do `data.frame` como preditoras, **exceto** `regiao` (removida explicitamente do atalho `.`).

Exercício 6 — `sapply` para diagnóstico rápido de missing

Usando `sapply` , produza um vetor nomeado com a contagem de `NA` por coluna de `varejo` , sem usar `dplyr` .

Solução:

```
sapply(varejo, function(col) sum(is.na(col)))
```

`is.na(col)` retorna vetor lógico (`TRUE` / `FALSE`) do mesmo tamanho da coluna; `sum()` sobre lógico conta os `TRUE` (R trata `TRUE` como `1` e `FALSE` como `0` em contexto aritmético — outro ponto de vetorização implícita).

`sapply` aplica essa função anônima (`function(col) ...` , definida inline sem nome) a cada coluna do data.frame e simplifica o resultado em vetor nomeado pelas colunas.

---

15. Entrega esperada do módulo

Nenhum script de entrega — este módulo é referência de consulta, não produz artefato próprio. Critério de conclusão é prático: rodar, sem consultar solução, os 6 exercícios acima.

---

16. Critério de domínio antes de avançar ao Módulo 1

- ☐ Sabe explicar a diferença entre `[ ]` , `[[ ]]` e `$` sem precisar testar no console
- ☐ Reconhece quando um `for` deveria ser vetorização/ `ifelse` / `apply` , e sabe reescrever
- ☐ Lê `%>%` fluentemente — consegue reescrever uma pipe de 3-4 passos como chamadas aninhadas e vice-versa
- ☐ Interpreta fórmula (`y ~ x1 * x2` , `y ~ .` , `y ~ x - 1`) sem precisar rodar o modelo para conferir
- ☐ Entende por que `NA` propaga por padrão em vez de ser ignorado, e sabe quando usar `na.rm = TRUE`
- ☐ Sabe quando usar `pacote::funcao()` em vez de `library(pacote)`

MÓDULO 01 · FUNDAMENTOS

Gráficos com ggplot2

1. Grammar of Graphics — o modelo conceitual

Matplotlib e a maior parte das libs de plot são **imperativas**: você dá uma sequência de comandos que desenham e modificam uma figura (`plt.plot(...)` , depois `plt.xlabel(...)` , depois `plt.legend(...)`). O estado do gráfico vive implícito, acumulado passo a passo.

O ggplot2 implementa a **Grammar of Graphics** (Leland Wilkinson, formalizada para R por Hadley Wickham): um gráfico é **declarado** como composição de camadas independentes que descrevem *o quê*, não *como desenhar*. Cada camada resolve uma pergunta específica:

Componente	Pergunta que resolve	Função típica
data	Quais dados?	`data.frame` / `tibble`
aes (aesthetics)	Quais variáveis mapeiam para quais propriedades visuais?	`aes(x=, y=, color=, size=...)`
geom	Qual forma geométrica representa cada observação?	`geom_point()` , `geom_bar()` , `geom_line()`
stat	Os dados precisam de transformação antes de plotar?	`stat_count()` , `stat_smooth()` (geralmente implícito dentro do geom)
scale	Como o domínio dos dados mapeia pro range visual (eixo, cor)?	`scale_x_continuous()` , `scale_color_brewer()`

Componente	Pergunta que resolve	Função típica
coord	Qual sistema de coordenadas?	`coord_cartesian()`, `coord_polar()`
facet	O gráfico se divide em subplots por alguma variável categórica?	`facet_wrap()`, `facet_grid()`
theme	Elementos não-dados: fundo, grid, fonte, legenda	`theme_minimal()`, `theme()`

A consequência prática: cada camada é independente e composável com `+`. Trocar um histograma por densidade é trocar `geom_histogram()` por `geom_density()` — o resto (data, aes, facet, theme) permanece intacto. Em matplotlib/seaborn isso costuma exigir reescrever boa parte da chamada.

---

2. Anatomia mínima de um gráfico

```
library(ggplot2)

ggplot(data = varejo, mapping = aes(x = area_m2, y = receita_mensal)) +
  geom_point()
```

Decompondo:

- `ggplot(data, aes(...))` inicializa o gráfico e declara o **mapeamento padrão** — herdado por todas as camadas seguintes, a menos que uma camada sobrescreva.
- `+ geom_point() adiciona a camada geométrica. Sem geom_*, ggplot()` sozinho só desenha um painel vazio com os eixos já escalados.
- Cada `aes()` pode também ser declarado **dentro** de um `geom_*` específico, o que restringe aquele mapeamento só àquela camada — útil quando você sobrepõe camadas com fontes de dados diferentes (ex.:

pontos de uma amostra + linha de um modelo ajustado a partir de outro data frame).

```
# aes local sobrescreve/complementa o global — comum em regressão + pontos
ggplot(varejo, aes(x = fluxo_clientes_mensal, y = receita_mensal)) +
  geom_point(alpha = 0.3) +                          # aes herdado do
ggplot()
  geom_smooth(method = "lm", color = "firebrick")    # também herda x, y
```

Sobre NSE (Non-Standard Evaluation): dentro de `aes()`, os nomes de coluna são escritos sem aspas e sem `$` — o ggplot2 resolve `area_m2` procurando essa coluna dentro do `data` declarado. Isso é diferente de indexação Pandas (`df['area_m2']`). Quando você precisar passar nome de coluna como **string** (por exemplo, dentro de uma função reutilizável — ver seção 13), use o pronome `.data[[ ]]`: `aes(x = .data[[nome_da_coluna]])`.

---

3. Setup

```
# Pacotes usados neste módulo
install.packages(c("ggplot2", "dplyr", "scales", "RColorBrewer"))

library(ggplot2)
library(dplyr)
library(scales)         # formatação de eixos (moeda, %, milhar)
library(RColorBrewer)   # paletas categóricas/sequenciais/divergentes

varejo <- read.csv("varejo_sintetico.csv", stringsAsFactors = TRUE)
painel <- read.csv("painel_regional_varejo.csv", stringsAsFactors = FALSE)
painel$mes_ano <- as.Date(painel$mes_ano)   # read.csv não infere Date
sozinho
```

`stringsAsFactors = TRUE` em `varejo` é proposital: `regiao`, `tipo_loja`, `porte` são categóricas com poucos níveis fixos — tratá-las como `factor` desde a leitura garante ordem de nível consistente em boxplots e barras (relevante já a partir da seção 5). Em `painel`, mantemos string porque `polo` e `cidade_nome` não têm ordenação natural relevante para os gráficos deste módulo.

4. Histograma e densidade — forma da distribuição

Pergunta que este gráfico responde: a variável é simétrica? Tem cauda longa? Multimodal? Isso antecede qualquer decisão de "média ou mediana", "teste paramétrico ou não" nos módulos 2, 4 e 7.

```
ggplot(varejo, aes(x = receita_mensal)) +
  geom_histogram(bins = 30, fill = "steelblue", color = "white") +
  labs(title = "Distribuição de receita mensal por loja",
       x = "Receita mensal (R$)", y = "Contagem de lojas")
```

- `bins` define o número de intervalos (alternativa: `binwidth` define a largura direto — mutuamente exclusivos, escolha um). Não existe valor "correto" universal: comece com `bins = 30` (default do ggplot2 é `bins = 30` com aviso), ajuste visualmente até a forma parar de mudar estruturalmente.
- `fill` colore a área da barra, `color` colore o contorno — nomenclatura ggplot2 é consistente: `fill` = preenchimento, `color` / `colour` = linha/contorno. Isso vale para todo geom com área (barras, boxplot, densidade).

Sobreposição de estatísticas de referência — média e mediana no mesmo gráfico, para visualizar o efeito de assimetria (o tema central do Módulo 2):

```
media_receita   <- mean(varejo$receita_mensal)
mediana_receita <- median(varejo$receita_mensal)

ggplot(varejo, aes(x = receita_mensal)) +
  geom_histogram(bins = 30, fill = "steelblue", color = "white") +
  geom_vline(xintercept = media_receita, color = "firebrick", linewidth =
1) +
  geom_vline(xintercept = mediana_receita, color = "darkgreen", linewidth
= 1, linetype = "dashed") +
  labs(title = "Receita mensal: média (vermelho) vs. mediana (verde
tracejado)",
       subtitle = "Distância entre as duas indica assimetria",
       x = "Receita mensal (R$)", y = "Contagem")
```

Densidade em vez de contagem: quando você quer comparar a *forma* de duas distribuições com N muito diferentes (ex.: comparar `receita_mensal` entre `porte` = Pequena vs. Grande, que têm contagens desiguais), contagem absoluta distorce a comparação visual. Normalize para densidade:

```
ggplot(varejo, aes(x = receita_mensal, fill = porte)) +
  geom_density(alpha = 0.4) +
  labs(title = "Densidade de receita mensal por porte de loja",
       x = "Receita mensal (R$)", y = "Densidade")
```

`alpha` controla transparência (0 = invisível, 1 = opaco) — essencial sempre que há sobreposição de geoms preenchidos.

---

5. Boxplot — comparação de grupos e outliers

O boxplot resume 5 números (mínimo não-outlier, Q1, mediana, Q3, máximo não-outlier) e plota outliers individualmente. É a ferramenta padrão para “essa variável numérica difere entre esses grupos categóricos?” — pergunta que antecede diretamente os testes dos módulos 4 e 7 (Mann-Whitney, Kruskal-Wallis, ANOVA).

```
ggplot(varejo, aes(x = porte, y = ticket_medio, fill = porte)) +
  geom_boxplot() +
  labs(title = "Ticket médio por porte de loja",
       x = "Porte", y = "Ticket médio (R$)") +
  theme(legend.position = "none")
# fill=porte já está no eixo x, legenda é redundante
```

Leitura do boxplot: a caixa é o IQR (Q3−Q1); a linha dentro da caixa é a mediana (não a média); os bigodes se estendem até 1.5×IQR a partir da caixa; pontos além disso são plotados individualmente como outliers candidatos. Um boxplot **não mostra a forma da distribuição dentro do IQR** — duas distribuições muito diferentes (bimodal vs. uniforme) podem gerar boxplots quase idênticos. Por isso boxplot e histograma se complementam, nunca um substitui o outro.

Boxplot cruzando duas categóricas (`porte` × `tipo_loja`) usando `fill` para a segunda dimensão:

```
ggplot(varejo, aes(x = porte, y = fluxo_clientes_mensal, fill =
tipo_loja)) +
  geom_boxplot() +
  labs(title = "Fluxo de clientes por porte e tipo de loja",
       x = "Porte", y = "Fluxo de clientes/mês", fill = "Tipo de loja")
```

`labs(fill = "Tipo de loja")` renomeia o título da legenda — por padrão o ggplot2 usa o nome literal da coluna (`tipo_loja`), quase nunca o rótulo que você quer entregar num documento final.

---

6. Scatter + regressão — relação entre duas variáveis

Antecede diretamente o Módulo 4 (correlação e regressão). O scatter é o primeiro teste visual de linearidade — premissa que a correlação de Pearson exige.

```
ggplot(varejo, aes(x = fluxo_clientes_mensal, y = receita_mensal)) +
  geom_point(alpha = 0.3, size = 1.5) +
  geom_smooth(method = "lm", se = TRUE, color = "firebrick") +
  labs(title = "Receita mensal vs. fluxo de clientes",
       x = "Fluxo de clientes/mês", y = "Receita mensal (R$)")
```

- `geom_smooth(method = "lm")` ajusta e desenha uma regressão linear simples sobre os dados plotados — útil para inspeção visual rápida, mas **não substitui** `lm()` explícito do Módulo 4 (aqui você não tem acesso aos coeficientes, R^2, resíduos).
- `se = TRUE` (default) desenha a banda de intervalo de confiança da reta. Em N grande essa banda fica visualmente insignificante — não confundir "banda estreita" com "relação forte": banda estreita só diz que a *estimativa da reta* é precisa, nada sobre o quanto os pontos se dispersam ao redor dela (isso é o R^2, não a CI da reta).
- Troque `method = "lm"` por `method = "loess"` para uma curva de suavização não-paramétrica — inspeção de não-linearidade sem assumir forma funcional, útil para o caso `taxa_conversao` vs. `fluxo_clientes_mensal` (relação saturante, não-linear por construção neste dataset):

```
ggplot(varejo, aes(x = fluxo_clientes_mensal, y = taxa_conversao)) +
  geom_point(alpha = 0.3) +
  geom_smooth(method = "loess", color = "darkorange") +
  geom_smooth(method = "lm", color = "steelblue", linetype = "dashed", se
= FALSE) +
  labs(title = "Taxa de conversão vs. fluxo: loess (laranja) expõe a
saturação que a reta (azul) esconde",
       x = "Fluxo de clientes/mês", y = "Taxa de conversão")
```

Este é o tipo de gráfico que justifica, visualmente, por que regressão polinomial (Módulo 4) é necessária em vez de forçar uma reta.

---

7. Gráficos de barras — categóricas

Duas funções, uso diferente — confusão comum vindo de `seaborn.countplot` vs. `seaborn.barplot`:

- **`geom_bar()`**: conta observações por categoria automaticamente (aplica `stat_count()` por baixo). Use quando seu data frame tem *uma linha por observação*.
- **`geom_col()`**: plota o valor de uma coluna já agregada diretamente (`stat = "identity"`, sem transformação). Use quando você já fez `group_by() %>% summarise()` antes.

```
# geom_bar — contagem direta, sem agregação prévia
ggplot(varejo, aes(x = regiao)) +
  geom_bar(fill = "steelblue") +
  labs(title = "Número de lojas por região", x = "Região", y = "Contagem")

# geom_col — dado já agregado
receita_por_regiao <- varejo %>%
  group_by(regiao) %>%
  summarise(receita_media = mean(receita_mensal))

ggplot(receita_por_regiao, aes(x = regiao, y = receita_media)) +
  geom_col(fill = "darkgreen") +
  labs(title = "Receita média mensal por região", x = "Região", y = "Recei
ta média (R$)")
```

Barras empilhadas vs. lado a lado (`position`):

```
ggplot(varejo, aes(x = regiao, fill = porte)) +
  geom_bar(position = "fill") +   # "fill" normaliza para proporção (0-1)
— bom p/ comparar composição
  labs(title = "Composição de porte de loja por região (proporção)",
       x = "Região", y = "Proporção", fill = "Porte") +
  scale_y_continuous(labels = percent)
```

`position = "stack"` (default) empilha em valor absoluto; `position = "dodge"` coloca lado a lado; `position = "fill"` normaliza pra 100% empilhado — a escolha depende se a pergunta é “quantos” ou “qual proporção”.

8. Séries temporais

Antecipa o Módulo 10, aqui só na camada de visualização (`geom_line`), sem decomposição/ARIMA ainda.

```
cidade_exemplo <- painel %>% filter(cidade_id == "CID01")

ggplot(cidade_exemplo, aes(x = mes_ano, y = indice_vendas)) +
  geom_line(color = "steelblue", linewidth = 0.7) +
  labs(title = paste("Índice de vendas —", unique(cidade_exemplo$cidade_no
me)),
       x = "Mês", y = "Índice de vendas")
```

Múltiplas séries sobrepostas com cor por grupo — útil para comparar padrão sazonal entre poucas cidades (evite mais que ~5-6 linhas no mesmo painel, vira "espaguete" ilegível; para mais grupos, use facet — seção 9):

```
tres_cidades <- painel %>% filter(cidade_id %in% c("CID01", "CID15", "CID3
0"))

ggplot(tres_cidades, aes(x = mes_ano, y = indice_vendas, color =
cidade_nome)) +
  geom_line(linewidth = 0.7) +
  labs(title = "Índice de vendas — 3 cidades", x = "Mês", y = "Índice de
vendas", color = "Cidade")
```

Eixo `Date` no `x` : o ggplot2 detecta a classe `Date` automaticamente e escolhe quebras (breaks) sensatas. Para controlar manualmente:

```
+ scale_x_date(date_breaks = "1 year", date_labels = "%Y")
```

9. Agrupamento visual: cor, forma, tamanho, facetas

Mapear uma terceira (ou quarta) variável para um atributo estético evita ter que fazer múltiplos gráficos separados manualmente:

```
ggplot(varejo, aes(x = area_m2, y = receita_mensal, color = porte, shape
= tipo_loja)) +
  geom_point(alpha = 0.5) +
  labs(title = "Receita vs. área, por porte (cor) e tipo (forma)",
       x = "Área (m²)", y = "Receita mensal (R$)")
```

Isso funciona até 3-4 variáveis simultâneas antes de virar ilegível. Além disso, **facetas** dividem o painel em subplots por uma variável categórica — mais legível que empilhar estéticas quando o número de grupos é maior que ~4:

```
ggplot(varejo, aes(x = fluxo_clientes_mensal, y = receita_mensal)) +
  geom_point(alpha = 0.3) +
  geom_smooth(method = "lm", se = FALSE, color = "firebrick") +
  facet_wrap(~ regiao) +
  labs(title = "Receita vs. fluxo, por região", x = "Fluxo de clientes/
mês", y = "Receita mensal (R$)")
```

- `facet_wrap(~var)` : distribui os subplots numa grade automática (linhas × colunas ajustadas pra caber).
- `facet_grid(var_linha ~ var_coluna)` : cruza **duas** variáveis categóricas, uma controla linhas, outra colunas — grade completa, sem "buracos":

```
ggplot(varejo, aes(x = ticket_medio)) +
  geom_histogram(bins = 20, fill = "steelblue") +
  facet_grid(porte ~ tipo_loja) +
  labs(title = "Distribuição de ticket médio por porte × tipo de loja")
```

`facet_grid` sempre gera `nrow(niveis_A) × nrow(niveis_B)` painéis (aqui, 3×3=9); `facet_wrap` é mais flexível quando você só tem uma variável ou quando o cruzamento completo teria células vazias/pouco povoadas.

Por padrão, todas as facetas compartilham a mesma escala de eixo — o que é correto para comparação justa entre painéis. Quando os grupos têm escalas muito diferentes e a prioridade é ver a *forma* de cada um individualmente, use `scales = "free"` (ou `"free_x"` / `"free_y"`), mas documente essa escolha explicitamente: liberar a escala facilita comparar forma e dificulta comparar magnitude — trade-off, não ganho gratuito.

10. Escalas: eixos e cor

10.1 Eixos numéricos e categóricos

```
ggplot(receita_por_regiao, aes(x = regiao, y = receita_media)) +
  geom_col(fill = "steelblue") +
  scale_y_continuous(labels = label_currency(prefix = "R$ ", big.mark = ".
", decimal.mark = ",")) +
  labs(title = "Receita média por região", x = "Região", y = "Receita
média")
```

`scales::label_currency()` (ou `dollar_format()` / `comma()`, dependendo da versão do pacote `scales`) formata os rótulos do eixo sem alterar o dado subjacente — separação limpa entre valor e apresentação, o mesmo princípio de `formatação de exibição` vs. `dado bruto` que você já aplica em relatórios Python.

`scale_x_continuous(breaks = seq(0, 20000, by = 5000))` controla manualmente onde ficam as marcações do eixo quando o default automático não é o que você quer.

10.2 Escalas de cor

Categórica (discreta) — `RColorBrewer`:

```
ggplot(varejo, aes(x = porte, y = receita_mensal, fill = porte)) +
  geom_boxplot() +
  scale_fill_brewer(palette = "Set2") +
  theme(legend.position = "none")
```

Famílias de paletas do `RColorBrewer` (chame `RColorBrewer::display.brewer.all()` para visualizar todas):

- **Sequential** (`"Blues"`, `"Greens"`, ...): para variável ordenada/contínua — do menor ao maior.

- **Diverging** (`"RdBu"` , `"PiYG"` , ...): para variável com ponto neutro relevante (ex.: desvio em torno de zero, correlação de -1 a 1).
- **Qualitative** (`"Set2"` , `"Dark2"` , ...): para categorias sem ordem — a família certa para `porte` , `regiao` , `tipo_loja` .

Contínua — gradiente numérico mapeado pra cor (ex.: mapa de calor, heatmap de correlação no Módulo 4):

```
ggplot(varejo, aes(x = area_m2, y = fluxo_clientes_mensal, color =
receita_mensal)) +
  geom_point(alpha = 0.6) +
  scale_color_viridis_c(option = "plasma") +
  labs(title = "Receita mensal mapeada em cor", color = "Receita (R$)")
```

`viridis` (`scale_color_viridis_c()` para contínua, `_d()` para discreta) é perceptualmente uniforme e seguro para daltonismo — prefira a `RColorBrewer` sempre que a variável for contínua e a decisão não tiver restrição específica de paleta corporativa.

---

11. Temas e personalização

Temas prontos (aplicam um conjunto coerente de fundo, grid, fonte de uma vez):

```
p <- ggplot(varejo, aes(x = porte, y = receita_mensal, fill = porte)) +
  geom_boxplot()

p + theme_minimal()   # fundo branco, sem borda, grid leve — bom default
para relatório
p + theme_bw()        # fundo branco, com borda no painel
p + theme_classic()   # sem grid, eixos como linha — estilo "publicação"
p + theme_void()      # remove tudo — útil pra mapas/gráficos decorativos
```

Personalização fina via `theme()` — cada elemento visual do gráfico (que **não** é dado) é acessível individualmente:

```
ggplot(varejo, aes(x = porte, y = receita_mensal, fill = porte)) +
  geom_boxplot() +
  scale_fill_brewer(palette = "Set2") +
  labs(title = "Receita mensal por porte de loja",
       subtitle = "Dataset sintético — 1.300 lojas",
       x = "Porte", y = "Receita mensal (R$)",
       caption =
"Fonte: dados sintéticos, projeto Estatística Aplicada em R") +
  theme_minimal() +
  theme(
    legend.position = "none",
    plot.title = element_text(face = "bold", size = 14),
    plot.subtitle = element_text(color = "gray40"),
    panel.grid.minor = element_blank(),     # remove grid secundário —
reduz ruído visual
    axis.title = element_text(face = "bold")
  )
```

Padrão útil: `element_text()` para texto, `element_line()` para linhas (grid, eixo), `element_rect()` para retângulos (fundo), `element_blank()` para remover o elemento completamente. Essa é a mesma lógica em qualquer canto do `theme()` — uma vez que você entende os 4 tipos de `element_*`, toda a superfície de customização do ggplot2 fica previsível.

Dica prática: defina seu próprio tema como função (`tema_estudo()`) uma vez e reuse em todos os gráficos do projeto — é exatamente o que a seção 13 entrega como template.

---

12. Coordenadas: polares e cortes de eixo

12.1 `coord_flip()` — inverter eixos

Útil quando rótulos de categoria são longos e ficam ilegíveis na horizontal:

```
ggplot(varejo, aes(x = reorder(regiao, receita_mensal, FUN = median), y =
receita_mensal)) +
  geom_boxplot(fill = "steelblue") +
  coord_flip() +
  labs(title = "Receita mensal por região (ordenado pela mediana)", x =
"", y = "Receita mensal (R$)")
```

`reorder(regiao, receita_mensal, FUN = median)` reordena os níveis do fator pela estatística escolhida — sem isso, `ggplot2` usa a ordem alfabética/factor original, quase nunca a ordem que comunica melhor.

12.2 `coord_polar()` — pizza e alternativas

Gráfico de pizza é `geom_bar()` (empilhado, proporção) + `coord_polar(theta = "y")`:

```
composicao_tipo <- varejo %>% count(tipo_loja)

ggplot(composicao_tipo, aes(x = "", y = n, fill = tipo_loja)) +
  geom_col(width = 1) +
  coord_polar(theta = "y") +
  scale_fill_brewer(palette = "Set2") +
  labs(title = "Composição de lojas por tipo", fill = "Tipo") +
  theme_void()
```

Ressalva teórica, não só estética: pizza (ângulo/área) é comprovadamente mais difícil de julgar quantitativamente pelo olho humano que comprimento de barra. Prefira `geom_col()` + `coord_flip()` ordenado sempre que a decisão for sua — use `coord_polar` quando o público/contexto exigir especificamente esse formato.

12.3 `coord_cartesian()` vs. `xlim()` / `ylim()` — zoom sem descartar dados

```
# ERRADO para "dar zoom": xlim()/ylim() FILTRAM os dados antes de qualquer
stat
# (geom_smooth recalculado só com os pontos dentro do range — distorce a
regressão)
ggplot(varejo, aes(fluxo_clientes_mensal, receita_mensal)) +
  geom_point() + geom_smooth(method = "lm") +
  xlim(0, 10000)   # descarta linhas fora do range ANTES do ajuste da reta

# CORRETO: coord_cartesian() só corta a JANELA DE VISUALIZAÇÃO, o ajuste
usa todos os dados
ggplot(varejo, aes(fluxo_clientes_mensal, receita_mensal)) +
  geom_point() + geom_smooth(method = "lm") +
  coord_cartesian(xlim = c(0, 10000))
```

Essa distinção parece cosmética mas muda o resultado estatístico do gráfico — é um erro comum o suficiente para merecer menção explícita neste ponto do módulo.

---

13. Diferenças-chave vindas de Python (matplotlib/ seaborn)

Conceito	Python (matplotlib/ seaborn)	R (ggplot2)
Modelo	Imperativo — comandos sequenciais mutam um estado (`fig` , `ax`)	Declarativo — camadas compostas com `+` , sem estado mutável explícito
Adicionar elemento	`ax.set_title(...)` , `ax.set_xlabel(...)`	`+ labs(title=..., x=...)`
Cor por grupo	`hue=` (seaborn) itera e desenha múltiplas séries	`aes(color=)` mapeia variável → cor numa única chamada declarativa

Conceito	Python (matplotlib/ seaborn)	R (ggplot2)
Subplots por grupo	`plt.subplots()` + loop manual, ou `sns.FacetGrid`	`facet_wrap()` / `facet_grid()` nativo, sem loop
Nome de coluna	String, `df['coluna']`	Símbolo sem aspas dentro de `aes()`; string só via `.data[[ ]]`
Reuso/composição	Função que recebe `ax` e desenha nele	Objeto `ggplot` é composável: `p + geom_smooth()` adiciona camada a um gráfico já pronto

O ponto que mais rende produtividade na transição: em ggplot2 você constrói um objeto (`p <- ggplot(...) + geom_point()`), guarda em variável, e **continua adicionando camadas depois** (`p + facet_wrap(~regiao)`, `p + theme_bw()`) sem redesenhar do zero — o equivalente mais próximo em matplotlib exigiria manter referência ao `Axes` e chamar métodos nele, que é exatamente o que `hue` / `FacetGrid` do seaborn tentam abstrair, mas de forma menos uniforme que a gramática de camadas.

---

14. Exercícios

Use `varejo_sintetico.csv` e `painel_regional_varejo.csv`. Carregue os pacotes e dados como na seção 3 antes de começar.

Exercício 1 — Histograma com referência

Construa um histograma de `gasto_marketing`. Adicione uma linha vertical na média e outra na mediana. Sem calcular nada numericamente ainda (isso é o Módulo 2) — só pela forma do histograma e pela distância entre as duas

linhas, essa variável parece simétrica ou assimétrica? Por quê o dataset foi construído assim (dica: releia o comentário sobre `gasto_marketing` no script gerador)?

Solução:

```
ggplot(varejo, aes(x = gasto_marketing)) +
  geom_histogram(bins = 40, fill = "steelblue", color = "white") +
  geom_vline(xintercept = mean(varejo$gasto_marketing), color = "firebrick
") +
  geom_vline(xintercept = median(varejo$gasto_marketing), color = "darkgre
en", linetype = "dashed") +
  labs(title = "Gasto com marketing: média (vermelho) vs. mediana
(verde)",
       x = "Gasto com marketing (R$)", y = "Contagem")
```

A distribuição é assimétrica à direita (cauda longa) — construída deliberadamente como log-normal, mais o efeito de ~1% de outliers (campanhas atípicas de alto investimento, ver `idx_outlier_mkt` no script gerador). A média fica visivelmente à direita da mediana porque é puxada pela cauda — o padrão clássico que justifica preferir mediana como medida de tendência central quando há assimetria forte (tema central do Módulo 2).

Exercício 2 — Boxplot comparativo

Construa um boxplot de `ticket_medio` por `porte`, ordenado do menor para o maior valor mediano (não pela ordem alfabética/factor padrão). Use uma paleta `RColorBrewer` qualitativa e remova a legenda redundante.

Solução:

```
ggplot(varejo, aes(x = reorder(porte, ticket_medio, FUN = median), y =
ticket_medio, fill = porte)) +
  geom_boxplot() +
  scale_fill_brewer(palette = "Set2") +
  labs(title = "Ticket médio por porte (ordenado pela mediana)", x = "Port
e", y = "Ticket médio (R$)") +
  theme_minimal() +
  theme(legend.position = "none")
```

Exercício 3 — Scatter facetado com regressão

Plote `receita_mensal` (y) vs. `fluxo_clientes_mensal` (x), com pontos semitransparentes, reta de regressão linear (`method = "lm"`) e facetado por `regiao`. Alguma região visualmente parece ter inclinação (slope) diferente das demais?

Solução:

```
ggplot(varejo, aes(x = fluxo_clientes_mensal, y = receita_mensal)) +
  geom_point(alpha = 0.25) +
  geom_smooth(method = "lm", color = "firebrick") +
  facet_wrap(~ regiao) +
  labs(title = "Receita vs. fluxo de clientes, por região",
       x = "Fluxo de clientes/mês", y = "Receita mensal (R$)")
```

Como `receita_mensal` foi gerada pela mesma fórmula estrutural em todas as regiões (o script gerador não diferencia `regiao` no termo de receita — só `tipo_loja` modula levemente o fluxo), a inclinação deve parecer visualmente semelhante entre painéis. Isso é, por si, uma checagem útil: se você *visse* inclinações claramente diferentes aqui, seria sinal de possível interação `fluxo × regiao` a investigar formalmente no Módulo 4 (o que, neste dataset sintético, não deveria aparecer por construção).

Exercício 4 — Barras de proporção

Construa um gráfico de barras mostrando a **proporção** de lojas com `teve_reclamacao_grave == 1`, agrupado por `tipo_loja`. (Dica: agregue com `dplyr` antes — `group_by(tipo_loja) %>% summarise(prop = mean(teve_reclamacao_grave))` — e use `geom_col()`, não `geom_bar()`.) Formate o eixo Y como percentual.

Solução:

```
prop_reclamacao <- varejo %>%
  group_by(tipo_loja) %>%
  summarise(prop_reclamacao_grave = mean(teve_reclamacao_grave))

ggplot(prop_reclamacao, aes(x = tipo_loja, y = prop_reclamacao_grave,
fill = tipo_loja)) +
  geom_col() +
  scale_y_continuous(labels = percent) +
  scale_fill_brewer(palette = "Set2") +
  labs(title = "Proporção de lojas com reclamação grave, por tipo",
       x = "Tipo de loja", y = "Proporção") +
  theme_minimal() +
  theme(legend.position = "none")
```

Exercício 5 — Série temporal facetada

Usando `painel_regional_varejo.csv`, selecione as cidades `CID01`, `CID10`, `CID25`, `CID40` e plote `indice_vendas` ao longo de `mes_ano`, uma linha por cidade, **facetado** por `cidade_nome` (não sobreposto por cor — compare o resultado com a versão sobreposta da seção 8 e note a diferença de legibilidade quando o padrão sazonal de cada série é o foco, não a comparação direta de nível).

Solução:

```
cidades_selecionadas <- painel %>% filter(cidade_id %in% c("CID01", "CID10
", "CID25", "CID40"))

ggplot(cidades_selecionadas, aes(x = mes_ano, y = indice_vendas)) +
  geom_line(color = "steelblue") +
  facet_wrap(~ cidade_nome, scales = "free_y") +
  labs(title = "Índice de vendas — 4 cidades", x = "Mês", y = "Índice de
vendas")
```

`scales = "free_y"` é justificado aqui: `nivel_base_cidade` no script gerador depende de `populacao_mil`, então cidades grandes e pequenas têm níveis de índice em escalas muito diferentes — sem escala livre, a sazonalidade das cidades menores fica visualmente achatada perto de zero.

Exercício 6 (desafio) — Coordenadas polares com dado agregado

Usando `painel_regional_varejo.csv`, agregue `indice_renda_media` médio por `polo` (nível cidade, não painel — cuidado para não duplicar linhas por causa do merge com o painel mensal: agregue com `cidades %>% group_by(polo) %>% summarise(...)` reconstruindo o nível-cidade via `distinct()`, ou releia o dataset de cidades separadamente). Construa um gráfico de barras em coordenadas polares (`coord_polar(theta = "x")`, sem `theta = "y"` desta vez) — um "gráfico radial" por polo.

Solução:

```
indice_por_polo <- painel %>%
  distinct(cidade_id, polo, indice_renda_media) %>%   # 1 linha por
cidade, não por cidade-mês
  group_by(polo) %>%
  summarise(renda_media = mean(indice_renda_media))

ggplot(indice_por_polo, aes(x = polo, y = renda_media, fill = polo)) +
  geom_col() +
  coord_polar(theta = "x") +
  scale_fill_brewer(palette = "Set2") +
  labs(title = "Índice de renda média por polo regional (radial)", x =
"", y = "") +
  theme_minimal()
```

A armadilha didática deste exercício é justamente o `distinct()` — sem ele, cada cidade entra 84 vezes (uma por mês) na agregação, mas como o valor de `indice_renda_media` é constante por cidade no painel (não varia mês a mês), a média sai numericamente correta mesmo com a duplicação. O ponto é o hábito: **sempre confirme a granularidade da linha antes de agregar** — nem sempre a duplicação é numericamente inofensiva como neste caso específico.

---

15. Entrega esperada do módulo

Biblioteca pessoal de templates reutilizáveis: arquivo `templates_ggplot2.R` (entregue junto com este documento) com 4 funções — histograma, boxplot, scatter+regressão, série temporal — parametrizadas para reuso direto nos módulos seguintes sem reescrever `aes()` / `geom_*` do zero a cada vez.

16. Critério de domínio antes de avançar ao Módulo 2

- ☐ Consegue explicar de cabeça as 4 camadas mínimas de um gráfico ggplot2 (data, aes, geom, e pelo menos uma de scale/coord/facet/theme) sem consultar este documento
- ☐ Sabe quando usar `geom_bar()` vs. `geom_col()` sem hesitar
- ☐ Entende a diferença entre `xlim()` / `ylim()` e `coord_cartesian(xlim=)` e por que ela importa estatisticamente, não só visualmente
- ☐ Consegue construir, sem copiar código, um histograma + boxplot + scatter+regressão + série temporal usando os dois datasets deste projeto
- ☐ Todos os 6 exercícios rodam sem erro e produzem gráfico com título, eixos rotulados e legenda com nome legível (não o nome cru da coluna)

MÓDULO 02 · FUNDAMENTOS

Estatística Descritiva

1. Distribuição de frequência

1.1 Variável categórica: frequência absoluta, relativa e acumulada

```r
library(dplyr)

varejo <- read.csv("varejo_sintetico.csv", stringsAsFactors = TRUE)

# Frequência absoluta
tabela_abs <- table(varejo$porte)
tabela_abs

# Frequência relativa (proporção)
tabela_rel <- prop.table(tabela_abs)
round(tabela_rel, 3)

# Combinando as duas + acumulada, como data frame legível
freq_porte <- varejo %>%
  count(porte, name = "freq_abs") %>%
  mutate(
    freq_rel = freq_abs / sum(freq_abs),
    freq_rel_acum = cumsum(freq_rel)
  )
freq_porte
```

`table()` conta ocorrências por nível de fator — equivalente a `value_counts()` do Pandas, mas retorna um objeto `table` (não um `Series`), o que importa se você for encadear com outras funções. `prop.table()` normaliza um `table` existente para proporções — não recalcula do zero, então sempre aplique sobre a tabela de contagem, não sobre o vetor original.

Frequência acumulada só faz sentido diretamente quando a variável categórica é **ordinal** (tem ordem natural) ou quando você a calcula sobre uma variável contínua discretizada em classes (seção 1.2). Para `porte` especificamente a ordem `Pequena < Media < Grande` é natural — por isso o

Módulo 1 já carregou esse fator com `levels` explícitos nessa ordem (seção 3 do Módulo 1). Para `regiao` ou `tipo_loja`, que não têm ordem natural, frequência acumulada não tem interpretação útil.

1.2 Variável contínua: classes de intervalo

Para variável contínua, "frequência" exige primeiro decidir em quantos intervalos (classes) dividir os dados — exatamente a mesma decisão que `bins` no `geom_histogram()` do Módulo 1. A regra clássica para número de classes é a **regra de Sturges**:

$$k = 1 + \log_2(n) \approx 1 + 3.322 \times \log_{10}(n)$$

Para `n = 1300`, isso dá aproximadamente `k ≈ 11`. Sturges assume distribuição aproximadamente normal e degrada em amostras muito grandes ou muito assimétricas — trate como ponto de partida, não como lei.

```
k <- 1 + 3.322 * log10(nrow(varejo))
k  # ~11.4 -> arredondar para 11 ou 12

varejo$classe_receita <- cut(varejo$receita_mensal, breaks = 11)

tabela_freq_receita <- varejo %>%
  count(classe_receita, name = "freq_abs") %>%
  mutate(
    freq_rel = freq_abs / sum(freq_abs),
    freq_rel_acum = cumsum(freq_rel)
  )
tabela_freq_receita
```

`cut(x, breaks = 11)` divide o range de `x` em 11 intervalos de largura igual e retorna um fator com rótulos tipo `"(52000,78000]"`. Repare que essa tabela de frequência **é** um histograma em forma de tabela — `geom_histogram()` do Módulo 1 faz exatamente esse binning por baixo dos panos antes de desenhar as barras. A frequência acumulada aqui responde perguntas como "que proporção das lojas fatura até R$ X?" — a base conceitual do que vira, formalizado, um percentil (seção 3.4).

2. Medidas de tendência central

2.1 Média aritmética

$$ \bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i $$

```
mean(varejo$receita_mensal)
```

Propriedade central da média: ela minimiza a soma dos quadrados dos desvios — é o valor c que minimiza $\sum(x_i - c)^2$. Essa propriedade é ótima para dados aproximadamente simétricos, mas é exatamente o que a torna **sensível a outliers**: um único valor extremo entra na soma elevado (indiretamente, via desvio grande) e puxa a média na sua direção, mesmo representando uma fração mínima da amostra.

2.2 Mediana

Valor que divide a amostra ordenada ao meio: 50% dos valores abaixo, 50% acima.

```
median(varejo$receita_mensal)
```

Para `n` ímpar, é o valor central da amostra ordenada; para `n` par, é a média dos dois valores centrais. A mediana tem **breakdown point** de 50% — você pode corromper até quase metade dos dados com valores arbitrariamente extremos sem que a mediana se mova de forma descontrolada, porque ela só depende da *posição* do valor central, não da sua *magnitude*. A média tem breakdown point de 0%: um único valor levado ao infinito leva a média junto.

2.3 Moda

Armadilha de R para quem vem de outra linguagem: `mode()` em R **não** calcula a moda estatística — retorna o tipo de armazenamento do objeto (`"numeric"`, `"character"`, etc.). É um dos erros de iniciante mais comuns na linguagem. Não existe função nativa para moda estatística; você constrói:

```
moda <- function(x) {
  freq <- table(x)
  as.numeric(names(freq)[freq == max(freq)])
}

moda(varejo$num_funcionarios)
```

Para variável **contínua**, moda no sentido "valor que mais se repete" quase nunca faz sentido — cada valor tende a ser único. Nesse caso, moda vira um conceito sobre a **forma da densidade**: a distribuição é unimodal, bimodal, trimodal? Isso se lê no histograma (Módulo 1) ou formalmente pelo pico da curva de densidade estimada:

```
dens <- density(varejo$area_m2)
dens$x[which.max(dens$y)]  # localização do pico principal da densidade
```

`area_m2` neste dataset é um caso didático interessante: foi gerada como **mistura de 3 normais** (uma por `porte`, médias 80/220/480 — ver `gerar_datasets_sinteticos.R`). Olhando `area_m2` ignorando `porte`, a distribuição tende a ser multimodal — potencialmente com 3 picos visíveis, um por porte de loja. Isso é o caso em que média e mediana **globais** (sem estratificar por `porte`) descrevem mal os dados: nenhuma das duas captura "existem três grupos distintos aqui". A média/mediana só são resumos honestos quando a distribuição é (aproximadamente) unimodal — esse é o ponto central do Exercício 2 abaixo.

2.4 Média aparada (trimmed mean)

Meio-termo entre média (usa tudo, sensível a outlier) e mediana (ignora magnitude, usa só posição): remove uma fração dos valores mais extremos de cada ponta antes de calcular a média.

```
mean(varejo$gasto_marketing, trim = 0.05)  # remove 5% de cada extremo
antes da média
```

Útil quando você quer robustez a outliers sem descartar toda informação de magnitude como a mediana faz.

2.5 Quando usar cada uma

Situação	Métrica recomendada	Por quê
Distribuição aproximadamente simétrica, sem outliers	Média	Usa toda a informação, é o estimador mais eficiente sob normalidade
Distribuição assimétrica ou com outliers	Mediana	Robusta, não distorcida pela cauda
Precisa de robustez mas quer reter mais informação que a mediana	Média aparada	Compromisso
Variável categórica ou distribuição multimodal	Moda (ou descrição por grupo)	Média/mediana de categoria não tem significado; em multimodal, nenhuma medida única de “centro” é honesta — estratifique

Situação	Métrica recomendada	Por quê
Comunicar para não-técnico com dados de renda/preço (classicamente assimétricos)	Mediana	É o que órgãos como IBGE reportam para renda — a média é sistematicamente mais alta que a experiência típica da população

---

3. Medidas de dispersão

Tendência central sozinha nunca é suficiente: duas distribuições podem ter média idêntica e formas completamente diferentes. Dispersão quantifica "o quão espalhados" os dados estão em torno do centro.

3.1 Amplitude (range)

Amplitude = $x_{max} - x_{min}$

```
range(varejo$receita_mensal)          # retorna c(min, max)
diff(range(varejo$receita_mensal))    # a amplitude propriamente dita
```

Métrica mais fraca de dispersão: usa só 2 pontos da amostra inteira, então é maximamente sensível a outliers e não diz nada sobre como os dados se distribuem *entre* os extremos.

3.2 Variância — populacional vs. amostral

Populacional (quando você tem os dados de **toda** a população, *N* observações):

$$ \sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2 $$

Amostral (quando os dados são uma **amostra** de n observações de uma população maior — o caso quase universal em análise aplicada):

$$ s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2 $$

Por que $n-1$ e não n (correção de Bessel): a média amostral $\bar{x}$ é, por construção, o valor que minimiza $\sum(x_i - c)^2$ dentro da própria amostra. Isso significa que $\sum(x_i - \bar{x})^2$ é sistematicamente **menor ou igual** a $\sum(x_i - \mu)^2$ (a soma usando a média populacional real, que você não conhece). Dividir por n nessa soma já reduzida produziria um estimador **viesado para baixo** — subestimaria a variância real da população. Dividir por $n-1$ (os "graus de liberdade" que sobram depois de gastar um grau para estimar $\bar{x}$) corrige exatamente esse viés, tornando s^2 um estimador não-viesado de σ^2.

```
var(varejo$receita_mensal)   # var() do R SEMPRE usa n-1 (amostral) — não
existe argumento pra mudar

# Verificação manual
x <- varejo$receita_mensal
n <- length(x)
sum((x - mean(x))^2) / (n - 1)   # == var(x)
sum((x - mean(x))^2) / n         # variância populacional — sempre menor
que a amostral
```

Gotcha vindo de Python: `numpy.var()` usa `ddof=0` por padrão (populacional), enquanto `pandas.Series.var()` usa `ddof=1` (amostral, igual ao R). É uma fonte clássica de números levemente diferentes ao portar código de `numpy` puro para R sem prestar atenção nesse detalhe — `numpy.var(x, ddof=1)` reproduz o comportamento do R.

3.3 Desvio padrão

$$ s = \sqrt{s^2} $$

```
sd(varejo$receita_mensal)   # sempre amostral (n-1), consistente com var()
```

Vantagem sobre a variância: está na **mesma unidade** da variável original (variância está em unidade ao quadrado — "reais ao quadrado" não tem interpretação intuitiva). Por isso desvio padrão é a medida de dispersão default para comunicação; variância é mais usada internamente em cálculos (ANOVA, regressão) porque tem propriedades algébricas mais simples (variâncias de variáveis independentes se somam; desvios padrão não).

3.4 Quartis e percentis

```
quantile(varejo$receita_mensal, probs = c(0.25, 0.5, 0.75))
quantile(varejo$receita_mensal, probs = seq(0, 1, 0.1))  # decis
```

Percentil *p* é o valor abaixo do qual *p*% dos dados caem. Quartis são os percentis 25, 50 (= mediana) e 75. **Detalhe que gera confusão vindo de outras linguagens:** existe mais de um algoritmo válido para calcular percentil quando o valor exato não coincide com uma observação da amostra (interpolação necessária). O R implementa 9 tipos diferentes (`quantile(x, type = 1:9)`) — o default é `type = 7`. Excel usa um método equivalente ao `type = 7` também; `numpy` / `pandas`, com `interpolation="linear"` (o default de ambos), também correspondem ao `type = 7` do R — na prática, os valores default costumam bater entre R e Python. O risco aparece quando alguém muda o `interpolation` no Python ou usa outro pacote estatístico (SAS usa `type = 3` por padrão) sem perceber — vale conferir sempre que os números "quase batem" mas não exatamente entre ferramentas.

3.5 IQR e a regra de outliers

$$IQR = Q_3 - Q_1$$

```
IQR(varejo$receita_mensal)
```

O IQR é a base da regra prática de detecção de outliers usada no boxplot do Módulo 1: um ponto é candidato a outlier se cai fora de $[Q_1 - 1.5 \times IQR, Q_3 + 1.5 \times IQR]$. Isso não é um teste estatístico formal (não tem p-valor associado) — é uma convenção descritiva de Tukey, robusta porque Q_1 e Q_3 não são puxados por outliers do mesmo jeito que média/desvio padrão seriam.

```
Q1 <- quantile(varejo$gasto_marketing, 0.25)
Q3 <- quantile(varejo$gasto_marketing, 0.75)
iqr <- Q3 - Q1
limite_superior <- Q3 + 1.5 * iqr

outliers <- varejo[varejo$gasto_marketing > limite_superior, ]
nrow(outliers)
```

---

4. Coeficiente de variação

$$ CV = \frac{s}{\bar{x}} \quad \text{(frequentemente expresso em \%: } CV \times 100\text{)} $$

```
cv <- function(x) sd(x) / mean(x)

cv(varejo$area_m2)
cv(varejo$receita_mensal)
```

Desvio padrão sozinho não permite comparar dispersão entre variáveis de **escalas diferentes** — dizer que `receita_mensal` (desvio padrão na casa dos milhares de reais) é "mais dispersa" que `area_m2` (desvio padrão na casa das dezenas de m²) não significa nada, porque as unidades e magnitudes são incomparáveis. O CV resolve isso normalizando o desvio padrão pela própria média, produzindo um número **adimensional** — permite comparar, por exemplo, se `num_funcionarios` é relativamente mais ou menos disperso que `fluxo_clientes_mensal`, mesmo estando em escalas completamente diferentes.

Duas ressalvas importantes:

1. **CV explode ou perde sentido quando a média está próxima de zero.** Se $\bar{x} \to 0$, $CV \to \infty$ mesmo que o desvio padrão absoluto seja pequeno. Não use CV em variáveis centradas em zero (ex.: `vendas_incrementais_fidelidade` do grupo controle, construída com média ≈ 0 — CV aqui seria instável e enganoso).
2. **CV exige escala de razão (ratio scale)**, isto é, um zero absoluto com significado real (R$, m², contagem — todos válidos). Não faz sentido em escala intervalar sem zero absoluto — o exemplo clássico é temperatura em Celsius/Fahrenheit: 10°C não é "duas vezes mais quente" que 5°C, então CV de temperatura em Celsius muda dependendo até de qual escala (Celsius vs. Fahrenheit) você usa, o que denuncia que a métrica não tem interpretação estável ali.

---

5. Sumarizando tudo: `summary()` e agregação por grupo

```
summary(varejo$receita_mensal)
# Min., 1st Qu., Median, Mean, 3rd Qu., Max. — 6 números, visão rápida

summary(varejo)  # aplica a cada coluna do data frame automaticamente
```

`summary()` sobre um data frame inteiro é o equivalente direto de `df.describe()` do Pandas — mas com diferenças relevantes: `summary()` do R inclui contagem de `NA`s automaticamente (relevante aqui: `receita_pre_campanha`, `receita_pos_campanha` e `vendas_incrementais_fidelidade` têm `NA` para lojas que não participaram do respectivo subconjunto — ver `gerar_datasets_sinteticos.R`), mas **não inclui desvio padrão** por padrão — diferente de `describe()`, que traz `std`. Para reproduzir a tabela completa que um Pandas `describe()` entregaria, monte manualmente:

```
resumo_completo <- varejo %>%
  summarise(
    n = n(),
    media = mean(receita_mensal),
    mediana = median(receita_mensal),
    dp = sd(receita_mensal),
    cv = sd(receita_mensal) / mean(receita_mensal),
    q1 = quantile(receita_mensal, 0.25),
    q3 = quantile(receita_mensal, 0.75),
    iqr = IQR(receita_mensal),
    min = min(receita_mensal),
    max = max(receita_mensal)
  )
resumo_completo
```

Estatística descritiva por grupo — o equivalente de `df.groupby(col).describe()`:

```
resumo_por_porte <- varejo %>%
  group_by(porte) %>%
  summarise(
    n = n(),
    media = mean(receita_mensal),
    mediana = median(receita_mensal),
    dp = sd(receita_mensal),
    cv = sd(receita_mensal) / mean(receita_mensal)
  )
resumo_por_porte
```

Esse padrão — `group_by() %>% summarise()` — é a ferramenta central para resolver o problema da seção 2.3: quando a variável global é multimodal por causa de um fator latente (`porte`), a estatística descritiva correta é sempre **estratificada por esse fator**, nunca a estatística global sozinha.

> ***Dados faltantes:*** *este dataset tem* `NA` *real em* `receita_pre_campanha`, `receita_pos_campanha` *e* `vendas_incrementais_fidelidade` *(visível no* `summary()` *acima). O tratamento formal disso — mecanismos MCAR/MAR/MNAR, consequência de* `na.omit()`*, imputação — está no* ***Módulo 3****, pensado para rodar logo em seguida a este.*

6. Diferenças-chave vindas de Python (Pandas)

Conceito	Python (Pandas/NumPy)	R
Resumo rápido de uma coluna	`df['col'].describe()` (inclui `std`, não inclui `NA` count por padrão)	`summary(df$col)` (inclui contagem de `NA`, não inclui `sd`)
Variância amostral (n-1)	`df['col'].var()` (ddof=1 default) — cuidado: `numpy.var()` puro usa ddof=0	`var(df$col)` — sempre n-1, sem opção
Moda estatística	`df['col'].mode()`	Não existe nativo — `mode()` retorna tipo do objeto. Função custom necessária
Percentil/ quantil	`df['col'].quantile(0.9)` (interpolation="linear" default)	`quantile(df$col, 0.9)` (type=7 default) — batem na maioria dos casos
Estatística por grupo	`df.groupby('col').describe()`	`df %>% group_by(col) %>% summarise(...)` — mais verboso, mas explícito sobre quais estatísticas você quer
Contagem de categorias	`df['col'].value_counts()`	`table(df$col)` ou `df %>% count(col)`
Tabela de frequência relativa	`df['col'].value_counts(normalize=True)`	`prop.table(table(df$col))`
Média ignorando missing	`df['col'].mean()` (ignora `NaN` por padrão)	`mean(df$col, na.rm = TRUE)` (R **não** ignora por padrão — retorna `NA` se não pedir)

Conceito	Python (Pandas/NumPy)	R
Descartar linhas com missing	`df.dropna()`	`na.omit(df)`

O padrão geral: Pandas empacota muita coisa em métodos de conveniência (`.describe()` , `.value_counts(normalize=True)`) que fazem várias contas de uma vez; o R (fora de `summary()`) tende a exigir que você componha as peças (`mean` , `sd` , `quantile` dentro de um `summarise()`) — mais digitação, mas força você a ser explícito sobre exatamente quais estatísticas está reportando, o que reduz erro de "esqueci que `describe()` não trazia isso".

---

7. Exercícios

Use `varejo_sintetico.csv` e `painel_regional_varejo.csv` . Carregue `dplyr` antes de começar.

Exercício 1 — Tabela de frequência

Construa a tabela de frequência absoluta, relativa e relativa acumulada de `tipo_loja` . A frequência acumulada faz sentido de ser interpretada aqui? Justifique com base na natureza da variável.

Solução:

```
freq_tipo <- varejo %>%
  count(tipo_loja, name = "freq_abs") %>%
  mutate(
    freq_rel = freq_abs / sum(freq_abs),
    freq_rel_acum = cumsum(freq_rel)
  )
freq_tipo
```

A acumulada é calculável, mas **não tem interpretação substantiva**: `tipo_loja` (Rua/Shopping/Quiosque) não tem ordem natural — a acumulada depende inteiramente da ordem arbitrária dos níveis do fator (alfabética por default), não de nenhuma propriedade real dos dados. Frequência acumulada só é uma leitura válida para variável ordinal ou contínua discretizada.

Exercício 2 — Multimodalidade e o limite da média/mediana

Calcule média, mediana e desvio padrão de `area_m2` ignorando `porte`. Depois repita a mesma conta estratificando por `porte` (`group_by`). Compare os dois resultados e explique por que a estatística global é uma descrição pobre desses dados.

Solução:

```
# Global — ignora a estrutura latente
varejo %>% summarise(media = mean(area_m2), mediana = median(area_m2), dp
= sd(area_m2))

# Estratificado — revela a estrutura real
varejo %>%
  group_by(porte) %>%
  summarise(media = mean(area_m2), mediana = median(area_m2), dp = sd(area
_m2), n = n())
```

`area_m2` foi construída como mistura de 3 normais com médias bem separadas (80/220/480 m² para Pequena/Media/Grande — ver `gerar_datasets_sinteticos.R`). A média e o desvio padrão globais descrevem um "meio-termo" que não corresponde a nenhuma loja típica real: nenhuma loja tem área "próxima da média global", porque as lojas se concentram em três agrupamentos distintos, não ao redor de um único centro. O desvio padrão global também fica artificialmente inflado — ele está capturando majoritariamente a distância *entre grupos*, não a dispersão *dentro* de cada grupo (que é bem menor, como o `dp` estratificado revela). Esse é

exatamente o tipo de estrutura que o Módulo 7 (cluster/PCA) formaliza — aqui você já está vendo, numericamente, o motivo pelo qual clusterização existe como técnica.

Exercício 3 — Variância: populacional vs. amostral, verificação manual

Para a variável `ticket_medio`, calcule manualmente (sem usar `var()`) a variância amostral e a variância populacional. Confirme que a versão amostral bate com `var(varejo$ticket_medio)`. Qual das duas é numericamente maior, e isso é sempre verdade?

Solução:

```
x <- varejo$ticket_medio
n <- length(x)

var_amostral <- sum((x - mean(x))^2) / (n - 1)
var_populacional <- sum((x - mean(x))^2) / n

var_amostral
var_populacional
var_amostral == var(x)  # TRUE
```

A variância amostral é sempre maior (ou, no limite, igual) à populacional, porque o numerador é idêntico e o denominador amostral (`n-1`) é sempre menor que o populacional (`n`) para `n > 1` — dividir a mesma soma por um número menor produz um resultado maior. A diferença relativa entre as duas encolhe conforme `n` cresce (com `n = 1300`, a diferença é pequena; seria bem mais visível com `n = 10`).

Exercício 4 — IQR e detecção de outliers

Para `gasto_marketing`, calcule Q1, Q3, IQR e o limite superior pela regra de Tukey (`Q3 + 1.5*IQR`). Quantas lojas são candidatas a outlier por esse critério? Esse número é compatível com a proporção de ~1% de campanhas atípicas inseridas deliberadamente no script gerador (`idx_outlier_mkt`)?

Solução:

```
Q1 <- quantile(varejo$gasto_marketing, 0.25)
Q3 <- quantile(varejo$gasto_marketing, 0.75)
iqr <- Q3 - Q1
limite_sup <- Q3 + 1.5 * iqr

outliers_mkt <- varejo[varejo$gasto_marketing > limite_sup, ]
nrow(outliers_mkt)
nrow(outliers_mkt) / nrow(varejo)  # proporção sobre o total
```

O número tende a ficar **acima** de 1%, e isso não é erro: a regra de Tukey não detecta exclusivamente os outliers "plantados" (`idx_outlier_mkt` , ~1% das lojas) — como `gasto_marketing` já é log-normal *por construção* (cauda longa mesmo sem outlier nenhum), uma fração adicional de valores legitimamente altos, mas não anômalos, também cai acima do limite. Isso ilustra uma limitação real da regra: ela assume implicitamente uma distribuição aproximadamente simétrica: em dados já assimetricamente distribuídos por natureza, ela superestima o número de outliers "reais". Vale comparar com o boxplot de `gasto_marketing` do Módulo 1 (Exercício 1) — os pontos além dos bigodes no gráfico são exatamente esses mesmos candidatos.

Exercício 5 — Coeficiente de variação comparando escalas

Calcule o coeficiente de variação de `area_m2` , `num_funcionarios` , `fluxo_clientes_mensal` e `receita_mensal` . Qual dessas quatro variáveis é **relativamente** mais dispersa (maior CV)? Essa resposta seria possível só olhando os desvios padrão brutos?

Solução:

```
varejo %>%
  summarise(
    cv_area = sd(area_m2) / mean(area_m2),
    cv_func = sd(num_funcionarios) / mean(num_funcionarios),
    cv_fluxo = sd(fluxo_clientes_mensal) / mean(fluxo_clientes_mensal),
    cv_receita = sd(receita_mensal) / mean(receita_mensal)
  )
```

Não seria possível comparar diretamente pelos desvios padrão brutos — `sd(fluxo_clientes_mensal)` está na casa das centenas/milhares e `sd(num_funcionarios)` está na casa das unidades, magnitudes incomparáveis por estarem em escalas/unidades diferentes. O CV normaliza isso. Espera-se que `num_funcionarios` e `area_m2` tenham CV relativamente mais alto que `receita_mensal`: as três primeiras variáveis herdam quase toda sua variação da mistura por `porte` (multiplicativa em relação à média de cada grupo), enquanto `receita_mensal` soma vários termos com ruído aditivo próprio (`rnorm(n_lojas, 0, 1800)`), o que tende a suavizar um pouco a dispersão relativa no agregado.

Exercício 6 (desafio) — Descritiva de painel: estabilidade por cidade

Usando `painel_regional_varejo.csv`, calcule para cada `cidade_id` a média, desvio padrão e coeficiente de variação de `indice_vendas` ao longo dos 84 meses. Ordene o resultado do maior para o menor CV. As cidades com maior CV são as de maior ou menor `populacao_mil`? Que hipótese isso sugere sobre estabilidade de vendas e porte de cidade?

Solução:

```
painel <- read.csv("painel_regional_varejo.csv", stringsAsFactors = FALSE)

estabilidade_cidade <- painel %>%
  group_by(cidade_id, cidade_nome, populacao_mil) %>%
  summarise(
    media_vendas = mean(indice_vendas),
    dp_vendas = sd(indice_vendas),
    cv_vendas = dp_vendas / media_vendas,
    .groups = "drop"
  ) %>%
  arrange(desc(cv_vendas))

head(estabilidade_cidade, 10)
cor(estabilidade_cidade$populacao_mil, estabilidade_cidade$cv_vendas)
```

O script gerador constrói `indice_vendas` com um termo de ruído mensal proporcional ao nível-base da cidade (`rnorm(n_meses, 0, base * 0.04)`) — ou seja, o ruído é multiplicativo em relação ao nível, o que por si só tende a manter o CV parecido entre cidades grandes e pequenas. Mas a correlação entre `populacao_mil` e `cv_vendas` vale conferir diretamente: se ela sair negativa (ainda que fraca), é evidência de que cidades maiores tendem a ter vendas relativamente mais estáveis — hipótese plausível em geral (mais pontos de venda agregados tendem a suavizar flutuação idiossincrática, uma manifestação prática da Lei dos Grandes Números que o Módulo 8 vai formalizar). Esse exercício antecipa diretamente a lógica do Módulo 8: por que agregados maiores têm variância amostral relativa menor.

---

8. Entrega esperada do módulo

Script `descritiva_varejo.R` que recebe `varejo_sintetico.csv` , e produz automaticamente:

1. Tabela de frequência de cada variável categórica (`regiao` , `tipo_loja` , `porte`)
2. Tabela-resumo (média, mediana, dp, CV, Q1, Q3, IQR, min, max) de todas as variáveis numéricas, calculada tanto globalmente quanto estratificada por `porte`
3. Lista de candidatos a outlier (regra de Tukey) para `gasto_marketing` e `receita_mensal`

---

9. Critério de domínio antes de avançar ao Módulo 4

- ☐ Explica de cabeça por que a variância amostral divide por `n-1` e não por `n`
- ☐

Sabe por que `mode()` do R não serve para calcular moda estatística, e sabe escrever a função alternativa

- ☐ Consegue decidir, para uma distribuição dada, se média ou mediana é a métrica de centro mais honesta — e justificar com a forma da distribuição, não por hábito
- ☐ Entende por que o coeficiente de variação existe e quando ele não deve ser usado (média perto de zero, escala sem zero absoluto)
- ☐ Sabe identificar quando uma estatística "global" está escondendo uma estrutura de subgrupos (o caso de `area_m2` por `porte`) e sabe corrigir isso com `group_by()`
- ☐ Todos os 6 exercícios rodam sem erro e a interpretação escrita conecta o número calculado com a forma como a variável foi construída (ou, para dados reais em módulos futuros, com o fenômeno que ela mede)

MÓDULO 03 · FUNDAMENTOS

Dados Faltantes

1. Missing por desenho vs. missing de verdade

O `varejo_sintetico.csv` já tem `NA` real em três colunas: `receita_pre_campanha`, `receita_pos_campanha` e `vendas_incrementais_fidelidade`.

```
library(dplyr)
varejo <- read.csv("varejo_sintetico.csv", stringsAsFactors = TRUE)

varejo %>%
  summarise(
    n_total = n(),
    n_na_pre = sum(is.na(receita_pre_campanha)),
    n_na_pos = sum(is.na(receita_pos_campanha)),
    n_na_fidelidade = sum(is.na(vendas_incrementais_fidelidade))
  )
```

Antes de qualquer técnica, uma distinção que este dataset ilustra bem, mas que **não** é o caso geral: aqui, o `NA` existe porque a variável **não se aplica** à linha — `receita_pre_campanha` só existe para lojas que participaram da campanha (`participou_campanha == 1`); para as demais, o valor não é "desconhecido", é **inexistente por construção**. Confira em `gerar_datasets_sinteticos.R` : `participou_campanha` é sorteado por `rbinom(n_lojas, 1, 0.19)` , independente de qualquer outra variável da loja.

Isso é **missing por desenho** (*structurally missing*) — não existe "recuperar" esse dado, porque ele nunca existiu para essas linhas. É diferente de dado real onde uma pergunta de formulário não foi respondida, um sensor falhou, ou um município deixou de reportar um ano ao IBGE — aí sim entra a distinção que a literatura de dados faltantes (Rubin, 1976) formaliza em três mecanismos.

2. Os três mecanismos de Rubin: MCAR, MAR, MNAR

- **MCAR (Missing Completely At Random)** — a probabilidade de faltar não depende de nada: nem do próprio valor faltante, nem de nenhuma outra variável observada. É o caso mais benigno, e o mais raro na prática real. O missing por desenho deste dataset **se comporta como MCAR** em relação às outras variáveis (o sorteio de participação é independente de tudo), mas não é MCAR no sentido estrito da literatura — é ausência estrutural, não estocástica.
- **MAR (Missing At Random)** — a probabilidade de faltar depende de **outras variáveis observadas**, mas não do próprio valor que falta. Exemplo típico em dado público brasileiro: municípios menores reportam menos ao IBGE — a chance de faltar depende de `populacao` (variável observada) — mas, dentro de municípios do mesmo porte, o motivo de faltar não tem relação com o valor real do indicador em si.
- **MNAR (Missing Not At Random)** — a probabilidade de faltar depende do **próprio valor que está faltando**. Exemplo: uma loja com receita muito baixa deixa de reportar exatamente porque a receita é ruim (constrangimento, risco de fechamento, ou simplesmente porque fechou). É o mecanismo mais perigoso porque é **impossível de detectar só olhando os dados observados** — exige conhecimento de domínio sobre o processo de coleta.

Por que a distinção importa na prática: os métodos de tratamento da seção 4 assumem, no mínimo, MAR. Sob MNAR, qualquer imputação — mesmo sofisticada — tende a produzir estimativas viesadas, porque o próprio mecanismo que gera o buraco está correlacionado com o valor que você está tentando reconstruir. Antes de imputar qualquer coisa, a pergunta correta não é “qual técnica uso”, é “eu sei, ou posso argumentar, qual desses três mecanismos gerou esse buraco?” — e essa resposta vem de

conhecimento do processo que gerou o dado, não de nenhum teste estatístico rodado sobre os dados observados (por definição, você não pode testar diretamente se o missing é MNAR usando só o que está observado).

---

3. Consequência de ignorar: listwise deletion

O comportamento default de muitas funções em R (e o mais comum entre iniciantes) é simplesmente descartar qualquer linha com `NA` em qualquer coluna usada:

```
mean(varejo$receita_pre_campanha)                # NA -- R se recusa a
calcular com NA presente
mean(varejo$receita_pre_campanha, na.rm = TRUE)   # ignora os NA, calcula
só sobre os observados

nrow(varejo)                                       # 1300
nrow(na.omit(varejo))
# cai bastante -- toda linha com QUALQUER NA em QUALQUER coluna some
```

`na.omit()` sobre o data frame inteiro é **listwise deletion**: se uma linha tem `NA` em `vendas_incrementais_fidelidade` só porque a loja não participou daquele teste específico, ela é descartada inteira — perdendo também `receita_mensal`, `area_m2` e todo o resto de informação boa daquela linha, mesmo que essas colunas estivessem completas. Duas consequências:

1. **Perda de poder estatístico** — menos linhas usadas do que as disponíveis, mesmo quando a maior parte da informação da linha era válida.
2. **Viés, se o missing não for MCAR** — se a chance de faltar está relacionada a alguma variável (MAR) ou ao próprio valor (MNAR), a amostra que sobra depois do `na.omit()` deixa de representar a população original. Sob MAR/MNAR, a subamostra completa é sistematicamente diferente da amostra original — não é “menos dados”, é “dados diferentes”.

Regra prática: nunca aplicar `na.omit()` no data frame inteiro por reflexo. Trate `NA` coluna a coluna, com `na.rm = TRUE` nas funções que precisam, e só remova linha quando a análise específica exigir aquela coluna especificamente.

---

4. Tratamento

4.1 Imputação simples

Aceitável para preencher lacunas quando o objetivo é descritivo, o missing é MCAR/MAR e a proporção é pequena — **não** para inferência formal, onde a incerteza da imputação precisa ser propagada (seção 4.2):

```
receita_pre_imputada <- varejo$receita_pre_campanha
receita_pre_imputada[is.na(receita_pre_imputada)] <- median(receita_pre_im
putada, na.rm = TRUE)
```

Prefira **mediana** a média — mesmo motivo do Módulo 2: robustez a assimetria e outliers. Mas repare no problema conceitual específico deste dataset: imputar `receita_pre_campanha` para lojas que nunca participaram da campanha não faz sentido de domínio — não existe um valor "verdadeiro" escondido para essas linhas, porque o missing é estrutural, não uma falha de coleta. É um exemplo de aplicar a técnica certa no lugar errado. Imputação simples só faz sentido quando o dado realmente existiria mas não foi capturado.

4.2 Imputação múltipla

Para dado real com missing genuíno (não estrutural) sob MAR, o padrão mais robusto é **imputação múltipla**: em vez de preencher cada buraco com um único valor fixo (que finge uma certeza que não existe), gera-se **várias** versões plausíveis do dataset completo — cada uma com valores imputados ligeiramente diferentes, a partir de um modelo estatístico das relações entre

as variáveis —, analisa-se cada versão separadamente, e combinam-se os resultados (regras de Rubin). Isso propaga corretamente a incerteza adicional que vem de não saber o valor real, em vez de escondê-la.

Em R, o pacote de referência é `mice` (*Multivariate Imputation by Chained Equations*):

```
# install.packages("mice")
library(mice)

imputado <- mice(varejo[, c("receita_mensal", "area_m2", "num_funcionarios
", "receita_pre_campanha")],
                 m = 5, method = "pmm", printFlag = FALSE)

varejo_completo <- complete(imputado, 1)  # uma das 5 versões imputadas

# Para análise formal (ex: regressão) que propaga a incerteza corretamente
entre as 5 versões:
# modelo <- with(imputado, lm(receita_mensal ~ area_m2 +
num_funcionarios))
# pool(modelo) -- combina os 5 ajustes pelas regras de Rubin
```

`method = "pmm"` (*predictive mean matching*) é o método default mais seguro para variáveis contínuas — imputa usando valores realmente observados em casos parecidos, em vez de gerar valores sintéticos que podem cair fora do range plausível da variável. `m = 5` é o número de imputações (datasets completos gerados); 5 é o mínimo tradicionalmente recomendado, `m` maior reduz variância de simulação ao custo de mais computação.

Este dataset não tem missing genuíno que justifique aprofundar em `mice` além do nível conceitual acima — é a ferramenta certa para quando você encontrar isso em dado real (INMET, IBGE, DATASUS costumam ter, especialmente séries históricas com estações ou anos faltantes).

4.3 Como reportar

Independente do método escolhido, todo relatório com dado real deve declarar: quantas linhas/colunas tinham `NA`, qual mecanismo foi assumido (e por quê), e qual tratamento foi aplicado. Reportar um resultado descritivo

sem mencionar que 15% dos dados foram descartados por `na.omit()` silencioso é, na prática, esconder uma decisão metodológica que muda a interpretação do resultado.

---

5. Diferenças-chave vindas de Python (Pandas)

Conceito	Python (Pandas)	R
Média ignorando missing	`df['col'].mean()` (ignora `NaN` por padrão)	`mean(df$col, na.rm = TRUE)` (R **não** ignora por padrão — retorna `NA` se não pedir)
Descartar linhas com missing	`df.dropna()`	`na.omit(df)`
Descartar só se coluna específica tiver NA	`df.dropna(subset=['col'])`	`df[!is.na(df$col), ]`
Preencher com valor fixo	`df['col'].fillna(valor)`	`df$col[is.na(df$col)] <- valor`
Contar missing por coluna	`df.isna().sum()`	`colSums(is.na(df))`
Imputação múltipla	`sklearn.impute.IterativeImputer` (experimental) ou `statsmodels.imputation.mice`	`mice` — pacote de referência, mais maduro e mais usado no ecossistema R

O ponto de atenção mais comum na migração: Pandas ignora `NaN` por padrão na maioria dos agregadores (`.mean()` , `.sum()`); R **não** ignora por padrão (`mean()` , `sd()` , `sum()` retornam `NA` se houver qualquer `NA` no vetor,

a menos que você passe `na.rm = TRUE` explicitamente). É um erro de transição comum: código que "funcionava" em Python silenciosamente retorna `NA` em R até que o `na.rm = TRUE` seja adicionado.

---

6. Exercícios

Exercício 1 — Mapeando o missing

Para cada coluna de `varejo_sintetico.csv`, calcule a proporção de `NA`. Quais colunas têm missing, e para cada uma, o valor faltante é estrutural (não se aplica) ou seria, em princípio, um dado real que poderia existir?

Solução:

```
prop_na <- colSums(is.na(varejo)) / nrow(varejo)
prop_na[prop_na > 0]
```

Três colunas têm `NA`: `receita_pre_campanha`, `receita_pos_campanha` (ambas ~83.5%, ligadas a `participou_campanha`) e `vendas_incrementais_fidelidade` (~75.4%, ligada a `participou_teste_fidelidade`). Todas as três são missing estrutural — a variável não existe para quem não integrou o respectivo subconjunto experimental, não é um dado real não coletado.

Exercício 2 — Mecanismo e efeito da deleção ingênua

Para `vendas_incrementais_fidelidade`: (a) argumente qual mecanismo (estrutural, MCAR, MAR ou MNAR) melhor descreve o missing, com base em como a variável foi gerada; (b) compare `nrow(varejo)` com o número de linhas restantes após um `na.omit()` sobre `varejo[, c("loja_id", "receita_mensal", "vendas_incrementais_fidelidade")]`, quantificando quanto de `receita_mensal` — uma coluna completa — seria perdido por tabela por uma análise que nada tem a ver com fidelidade.

Solução:

```
sub <- varejo[, c("loja_id", "receita_mensal", "vendas_incrementais_fideli
dade")]
nrow(varejo)
nrow(na.omit(sub))
nrow(varejo) - nrow(na.omit(sub))
(nrow(varejo) - nrow(na.omit(sub))) / nrow(varejo)
```

O mecanismo é missing por desenho: `participou_teste_fidelidade` é sorteado por `rbinom(n_lojas, 1, 0.23)`, independente de qualquer característica da loja (incluindo `receita_mensal`) — a ausência não carrega informação sobre o valor que faltaria nem sobre nenhuma outra variável. Comportamentalmente equivalente a MCAR (seguro ignorar sem viés), mas conceitualmente distinto (o valor não existe, não é desconhecido). O ponto pedagógico do item (b): um `na.omit()` no data frame inteiro descartaria ~75% das linhas mesmo para uma análise que só precisa de `receita_mensal`, completo em praticamente todas elas — demonstração concreta de por que listwise deletion irrestrito quase sempre é a escolha errada.

Exercício 3 — Imputação simples e seu limite conceitual

Impute `receita_pre_campanha` pela mediana dos valores observados. Depois, argumente por que essa imputação específica não faz sentido de domínio para as linhas onde `participou_campanha == 0`, mesmo estando tecnicamente correta.

Solução:

```
mediana_pre <- median(varejo$receita_pre_campanha, na.rm = TRUE)
receita_pre_imputada <- ifelse(is.na(varejo$receita_pre_campanha),
mediana_pre, varejo$receita_pre_campanha)
```

Tecnicamente a imputação roda sem erro. Mas para uma loja que nunca participou da campanha, não existe “receita antes da campanha que ela não fez” — a pergunta em si não faz sentido para essas linhas. Preencher com a mediana cria um número que parece válido mas não representa nada real; é diferente de imputar, por exemplo, uma temperatura de estação

meteorológica que falhou por um dia (onde o valor real existiu, só não foi medido). A lição: sempre confirmar que o missing é do tipo "existe mas não foi capturado" antes de imputar — nunca imputar por reflexo só porque a função pede um número.

Exercício 4 — MAR simulado

Crie artificialmente uma versão MAR de `receita_mensal`: torne o valor `NA` para lojas de porte "Pequena" com probabilidade 40%, e mantenha as demais completas. Calcule a média de `receita_mensal` (a) sobre os dados originais completos e (b) sobre os dados com o missing artificial, usando `na.rm = TRUE`. As duas médias batem? Por quê?

Solução:

```
set.seed(42)
varejo_mar <- varejo
idx_pequena <- which(varejo_mar$porte == "Pequena")
idx_na <- sample(idx_pequena, size = round(0.4 * length(idx_pequena)))
varejo_mar$receita_mensal[idx_na] <- NA

mean(varejo$receita_mensal)
mean(varejo_mar$receita_mensal, na.rm = TRUE)
```

As médias **não** batem — a segunda vem sistematicamente mais alta, porque o missing foi induzido preferencialmente em lojas pequenas (que têm receita menor por construção), e `na.rm = TRUE` simplesmente as remove do cálculo, mudando a composição da amostra restante. Isso é a demonstração direta de por que MAR gera viés quando tratado com deleção simples: a chance de faltar dependeu de uma variável observada (`porte`), que por sua vez está relacionada ao valor da variável de interesse (`receita_mensal`) — mesmo sem o missing depender diretamente do próprio valor de receita (o que seria MNAR).

---

7. Entrega esperada do módulo

Script `dados_faltantes_varejo.R` que recebe `varejo_sintetico.csv` e produz:

1. Tabela de proporção de `NA` por coluna
2. Classificação do mecanismo assumido (estrutural/MCAR/MAR/MNAR) para cada coluna com missing, com justificativa de uma linha
3. Demonstração do efeito de `na.omit()` irrestrito vs. tratamento coluna a coluna, quantificado em linhas/proporção perdida

---

8. Critério de domínio antes de avançar

- ☐ Distingue missing estrutural dos três mecanismos de Rubin (MCAR/MAR/MNAR)
- ☐ Sabe por que MNAR não é detectável só a partir dos dados observados
- ☐ Sabe por que `na.omit()` aplicado ao data frame inteiro é geralmente a escolha errada, e usa `na.rm = TRUE` coluna a coluna por padrão
- ☐ Entende a diferença entre imputação simples e múltipla, e por que a múltipla é necessária para inferência formal
- ☐ Todos os 4 exercícios rodam sem erro e a interpretação escrita justifica o mecanismo assumido, não só descreve o número

---

PARTE 2

Modelagem

Correlação, regressão linear e logística, testes não-paramétricos e as duas técnicas centrais de estatística multivariada: cluster e PCA.

MÓDULO 04 · MODELAGEM

Correlação e Regressão

Estatística Bivariada

1. Correlação de Pearson

1.1 Definição

Correlação de Pearson mede a **força e direção de uma relação linear** entre duas variáveis contínuas:

$$ r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} = \frac{\text{cov}(x,y)}{s_x \, s_y} $$

r varia entre -1 e $+1$. É, por construção, o **coeficiente de covariância normalizado** — a covariância sozinha depende da escala das variáveis (covariância entre `receita_mensal` em R$ e `area_m2` em m² não é comparável com covariância entre duas variáveis em outras unidades); dividir pelo produto dos desvios padrão remove essa dependência de escala, assim como o CV do Módulo 2 remove a dependência de escala do desvio padrão.

```
library(dplyr)

varejo <- read.csv("varejo_sintetico.csv", stringsAsFactors = TRUE)

cor(varejo$fluxo_clientes_mensal, varejo$receita_mensal)
```

Interpretação de magnitude — não existe corte universal, mas a convenção mais usada (Cohen, 1988, para ciências sociais/comportamentais — em outras áreas os limiares mudam):

|r|	Interpretação convencional
< 0.10	Desprezível
0.10 – 0.30	Fraca
0.30 – 0.50	Moderada
0.50 – 0.70	Forte
> 0.70	Muito forte

Trate esses limiares como referência de linguagem comum, não como regra estatística — o que conta como "forte" depende do fenômeno (correlações de 0.3 são notáveis em ciências sociais, e triviais em física).

1.2 Premissas

Pearson assume:

1. **Ambas as variáveis são intervalares ou de razão** (contínuas, com distância numérica com significado real) — não use Pearson em ordinal (para isso existe Spearman, Módulo 6).
2. **Relação linear** — Pearson mede especificamente a força de uma reta. Uma relação forte, porém curva (como `taxa_conversao` vs. `fluxo_clientes_mensal` neste dataset, ver seção 6), pode gerar um *r* baixo mesmo com dependência quase determinística entre as variáveis.
3. **Ausência de outliers extremos** — igual à média, *r* não é robusto: um único ponto excêntrico em ambas as variáveis pode inflar ou destruir a correlação de toda a amostra.
4. **Homocedasticidade da relação** (para inferência formal sobre *r*, via `cor.test()`) — a dispersão de *y* em torno da reta deve ser razoavelmente constante ao longo de *x*.

1.3 Verificação visual antes de confiar no número

Nunca reporte um *r* sem antes ter olhado o scatter (Módulo 1, seção 6). O exemplo didático clássico é o **Quarteto de Anscombe**: quatro conjuntos de dados com exatamente o mesmo *r* (0.816), mesma média, mesma variância — um é linear de fato, um é uma curva perfeita mal servida por uma reta, um tem um outlier que sozinho cria a correlação, um é vertical com um ponto de alavancagem. O número sozinho não distingue essas quatro situações; o gráfico distingue instantaneamente.

```
ggplot(varejo, aes(x = fluxo_clientes_mensal, y = receita_mensal)) +
  geom_point(alpha = 0.3) +
  labs(title = "Verificação visual antes de reportar r")
```

1.4 Significância da correlação (prévia do Módulo 9)

`cor()` retorna só o número. Para saber se um *r* observado é estatisticamente distinguível de zero (ou seja, se a relação linear observada é improvável de ser só ruído amostral), usa-se `cor.test()`:

```
cor.test(varejo$fluxo_clientes_mensal, varejo$receita_mensal)
```

Isso retorna *r*, um intervalo de confiança para *r* e um p-valor. A teoria completa de teste de hipótese, p-valor e erro Tipo I/II só é formalizada no Módulo 9 — por ora, use como regra prática: com `n` grande (como os 1300 desta amostra), correlações mesmo pequenas (`r = 0.08`) tendem a dar "estatisticamente significativas", o que é outro lembrete de que significância estatística **não é o mesmo que relevância prática**. Um *r* de 0.08 é irrelevante para qualquer decisão de negócio, mesmo que o p-valor seja baixo.

1.5 Correlação ≠ causalidade

`gasto_marketing` e `fluxo_clientes_mensal` neste dataset são fortemente correlacionados **por construção** — não porque marketing causa fluxo diretamente no modelo gerador, mas porque ambos são impulsionados pelo mesmo fator latente (`porte` da loja: lojas grandes têm mais fluxo *e* mais orçamento de marketing, lojas pequenas têm menos dos dois). Isso é **confundimento** (confounding): uma terceira variável não observada (ou não incluída no modelo) cria uma associação entre duas variáveis que não têm relação causal direta entre si. O Exercício 6 deste módulo usa o dataset de painel para você experimentar exatamente esse padrão com dados espaciais — tema que retorna, formalizado, no Módulo 11.

---

2. Matriz de correlação

Quando há mais de duas variáveis numéricas de interesse, computar pares um a um é ineficiente — use uma matriz de correlação:

```
vars_numericas <- varejo %>%
  select(area_m2, num_funcionarios, fluxo_clientes_mensal, ticket_medio,
         gasto_marketing, receita_mensal, taxa_conversao)

matriz_cor <- cor(vars_numericas)
round(matriz_cor, 2)
```

`cor()` aplicado a um data frame (ou matriz) numérico retorna a matriz completa de correlações par a par — a diagonal é sempre 1 (correlação de uma variável com ela mesma), a matriz é simétrica.

Visualização — uma matriz numérica de 7×7 é difícil de ler rápido; `corrplot` ou `GGally::ggpairs` resolvem isso:

```
install.packages(c("corrplot", "GGally"))
library(corrplot)

corrplot(matriz_cor, method = "color", type = "upper",
         addCoef.col = "black", tl.col = "black", tl.srt = 45,
         col = colorRampPalette(c("firebrick", "white", "steelblue"))(200)
)
```

`corrplot` usa por padrão uma paleta divergente (ver Módulo 1, seção 10.2) — correto aqui porque correlação tem um ponto neutro (zero) com significado real, diferente de uma variável só positiva.

`GGally::ggpairs()` vai além: gera uma grade com scatter para cada par, histograma/densidade na diagonal, e o *r* numérico — é o jeito mais rápido de fazer a checagem visual da seção 1.3 para várias variáveis de uma vez, mas fica pesado com muitas variáveis ou `n` muito grande (considere amostrar linhas para o gráfico se `n` > alguns milhares).

```
GGally::ggpairs(vars_numericas, progress = FALSE)
```

Cuidado com múltiplas comparações: quanto mais variáveis na matriz, mais pares você está testando implicitamente — com 7 variáveis já são 21 pares. Se você fosse rodar `cor.test()` em cada par e procurar "o que deu significativo", a chance de achar algo por puro acaso amostral cresce com o número de testes (o problema de comparações múltiplas, que retorna formalmente no Módulo 9). Matriz de correlação é uma ferramenta exploratória, não uma bateria de testes de hipótese confirmatórios.

---

3. Regressão linear simples

3.1 O modelo

Regressão linear simples modela uma variável **resposta** (*y*) como função linear de uma variável **preditora** (*x*):

$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$

- β_0 (intercepto): valor esperado de y quando $x = 0$.
- β_1 (coeficiente angular/slope): variação esperada em y para cada unidade de aumento em x.
- ε_i (resíduo/erro): tudo que o modelo não explica — a distância vertical entre o ponto observado e a reta.

O que o modelo estima, exatamente: não a relação "verdadeira" entre as variáveis (essa você nunca observa diretamente), mas a reta que melhor resume a relação linear presente **nesta amostra específica**, sob o critério de mínimos quadrados.

3.2 Método dos mínimos quadrados (OLS — Ordinary Least Squares)

O ajuste escolhe $\hat{\beta}_0$ e $\hat{\beta}_1$ que minimizam a soma dos quadrados dos resíduos:

$$ SSE = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{n}\left(y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i\right)^2 $$

Por que quadrado e não valor absoluto? Duas razões práticas: (1) elevar ao quadrado penaliza desproporcionalmente erros grandes — um resíduo de 10 pesa 100, um resíduo de 2 pesa só 4, o que faz o ajuste "se esforçar mais" para evitar erros grandes; (2) a solução em quadrados tem forma fechada (derivável analiticamente), o que a torna computacionalmente trivial — a alternativa (minimizar valor absoluto, regressão de mediana/quantílica) exige otimização numérica, não uma fórmula direta.

A solução fechada (derivada de $\partial SSE/\partial\beta_0 = 0$ e $\partial SSE/\partial\beta_1 = 0$) é:

$$ \hat{\beta}_1 = \frac{\sum(x_i-\bar{x})(y_i-\bar{y})}{\sum(x_i-\bar{x})^2} = \frac{\text{cov}(x,y)}{\text{var}(x)} $$

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

Repare a relação direta com a seção 1: $\hat{\beta}_1$ é a covariância normalizada pela variância de *x* (não pelo produto dos desvios padrão, como *r*) — é por isso que β_1 tem unidade (R$ por m², por exemplo), enquanto *r* é adimensional.

3.3 Ajustando no R

```
modelo_simples <- lm(receita_mensal ~ fluxo_clientes_mensal, data =
varejo)
summary(modelo_simples)
```

Sintaxe de fórmula do R: `y ~ x` lê-se "y explicado por x" (ou "y em função de x"). É a mesma notação usada em praticamente todo modelo estatístico do R (`lm`, `glm`, `aov`, `arima` com covariáveis) — vale internalizar cedo.

`summary()` de um objeto `lm` retorna, entre outras coisas:

- `Estimate` de cada coeficiente ($\hat{\beta}_0$, $\hat{\beta}_1$).
- `Std. Error`, `t value`, `Pr(>|t|)` — erro padrão e teste de hipótese de cada coeficiente contra zero. Teoria completa no Módulo 9; por ora, leia como "esse coeficiente é distinguível de zero, dado o ruído da amostra?".
- `R-squared` e `Adjusted R-squared` (seção 4).
- `F-statistic` — teste de que **pelo menos um** coeficiente é diferente de zero (no caso de regressão simples, equivalente ao teste do próprio β_1).

Extraindo coeficientes programaticamente (útil quando você quer usar o valor em código, não só ler o output):

```
coef(modelo_simples)                    # vetor nomeado: (Intercept),
fluxo_clientes_mensal
coef(modelo_simples)["fluxo_clientes_mensal"]
confint(modelo_simples)                 # intervalo de confiança de cada
coeficiente
```

3.4 Premissas do modelo linear (Gauss-Markov + normalidade)

Para que os coeficientes estimados sejam confiáveis (não só calculáveis — `lm()` sempre retorna algum número, mesmo com premissas violadas):

1. **Linearidade** — a relação real entre *x* e *y* é (aproximadamente) linear. Verificação: scatter (seção 1.3) e resíduos vs. ajustados (abaixo).
2. **Independência dos resíduos** — observações não podem ser autocorrelacionadas (viola-se rotineiramente em séries temporais e dados espaciais — Módulos 10 e 11 tratam exatamente essa violação).
3. **Homocedasticidade** — variância do resíduo constante ao longo de todos os valores de *x* (o oposto, heterocedasticidade, é comum quando a variável resposta cresce em escala — como receita: lojas maiores tendem a ter resíduo absoluto maior).
4. **Normalidade dos resíduos** — necessária para que os testes de hipótese sobre os coeficientes (t, F) e os intervalos de confiança sejam válidos. Não é necessária para o ajuste em si (OLS não assume normalidade para *calcular* $\hat{\beta}$) — só para a inferência sobre os coeficientes.

Diagnóstico visual direto no R — `plot()` aplicado a um objeto `lm` gera 4 gráficos de diagnóstico automaticamente:

```
par(mfrow = c(2, 2))
plot(modelo_simples)
par(mfrow = c(1, 1))
```

- **Residuals vs Fitted**: procura padrão (curva, funil) — padrão indica não-linearidade ou heterocedasticidade. Nuvem sem forma é o resultado desejado.
- **Q-Q Residuals**: pontos alinhados à diagonal indicam resíduos aproximadamente normais (teoria formal de teste de normalidade no Módulo 9).
- **Scale-Location**: outra visão de homocedasticidade — linha de tendência horizontal é o ideal.
- **Residuals vs Leverage**: identifica pontos de alavancagem (observações com *x* extremo que puxam desproporcionalmente a reta) via distância de Cook.

Este módulo introduz a leitura visual; o Módulo 9 formaliza os testes numéricos correspondentes (Shapiro-Wilk para normalidade, teste de homogeneidade de variância). A seção 7 deste módulo já adianta os testes numéricos específicos de diagnóstico de regressão (heterocedasticidade, pontos influentes) que não dependem da teoria de teste de hipótese do Módulo 9.

3.5 Intervalo de predição vs. intervalo de confiança

Duas perguntas diferentes que `predict()` responde de formas diferentes, e que se confundem com frequência:

- **Intervalo de confiança (IC) da média prevista**: "qual a faixa provável para a **média** de *y*, dado esse *x*?" — incerteza só sobre onde está a reta verdadeira.
- **Intervalo de predição (IP) de uma observação individual**: "qual a faixa provável para **uma loja específica nova** com esse *x*?" — soma a incerteza

sobre a reta **e** a variabilidade natural em torno dela (o próprio ε do modelo).

```
novo_dado <- data.frame(fluxo_clientes_mensal = 8000)

predict(modelo_simples, newdata = novo_dado, interval = "confidence")
predict(modelo_simples, newdata = novo_dado, interval = "prediction")
```

O IP é **sempre mais largo** que o IC para o mesmo x — muitas vezes bem mais largo — porque carrega uma fonte extra de incerteza que o IC não tem. Confundir os dois é um erro comum e caro na prática: se você quer comunicar "a receita média esperada para lojas com esse fluxo está entre X e Y", use IC; se quer comunicar "a receita **desta loja específica** deve cair entre X e Y", use IP. Usar IC para prever uma observação individual sistematicamente subestima a incerteza real — o intervalo reportado parece mais preciso do que realmente é.

---

4. Coeficiente de determinação (R^2)

4.1 O que R^2 mede

$$ R^2 = 1 - \frac{SSE}{SST} = \frac{SSR}{SST} $$

Onde $SST = \sum(y_i - \bar{y})^2$ (variância total de y), $SSE = \sum(y_i - \hat{y}_i)^2$ (variância não explicada, os resíduos) e $SSR = SST - SSE$ (variância explicada pelo modelo).

Interpretação direta: **a proporção da variância de y que o modelo explica**. $R^2 = 0.65$ significa que 65% da variação observada em `receita_mensal` é explicada linearmente por `fluxo_clientes_mensal` nesta amostra — os outros 35% são atribuídos a outras causas não incluídas no modelo (ou a ruído genuinamente aleatório).

Na regressão linear **simples** especificamente, $R^2 = r^2$ — o quadrado do próprio coeficiente de Pearson entre *x* e *y*. Essa igualdade só vale no caso de um único preditor; em regressão múltipla, R^2 não é o quadrado de nenhum *r* par a par isolado.

```
summary(modelo_simples)$r.squared
cor(varejo$fluxo_clientes_mensal, varejo$receita_mensal)^2   # deve bater
```

4.2 O que R² não mede

Esta é a parte que mais gera decisão ruim quando ignorada:

- **Não mede se o modelo está correto/bem especificado.** Um R² de 0.90 com uma relação verdadeiramente curva forçada numa reta ainda pode reportar R² alto se a curva for suave o bastante — o R² não denuncia erro de forma funcional sozinho, os resíduos denunciam (seção 3.4).
- **Não mede causalidade.** R² alto entre `gasto_marketing` e `receita_mensal` não prova que gastar mais em marketing *causa* mais receita — pode ser, de novo, o confundimento por `porte` (seção 1.5).
- **Não mede se o modelo generaliza fora da amostra.** R² é calculado *dentro* dos dados usados para ajustar o modelo — é sempre possível aumentar R² artificialmente adicionando mais preditores, mesmo preditores sem relação real com *y* (mais sobre isso na seção 4.3). Avaliação de generalização exige dados de validação/teste — fora do escopo deste módulo introdutório, mas é o motivo pelo qual "R² alto" nunca deveria ser, sozinho, critério de aceite de um modelo em produção.
- **Não diz nada sobre relevância prática do efeito.** Um R² baixo (0.05) pode coexistir com um coeficiente β_1 que, em magnitude, é enorme e financeiramente relevante — R² fala sobre proporção de variância explicada, não sobre o tamanho do efeito em unidades reais.
- **É sensível à amplitude (range) dos dados observados.** Restringir a amostra a uma faixa estreita de *x* tende a reduzir R² mesmo que a

relação subjacente seja idêntica — porque menos variação em *x* produz menos variação explicável em *y*.

4.3 R² ajustado

$$ R^2_{ajustado} = 1 - (1 - R^2)\frac{n-1}{n-p-1} $$

Onde *p* é o número de preditores no modelo. R² comum **nunca diminui** quando você adiciona uma variável ao modelo, mesmo uma variável aleatória sem relação nenhuma com *y* — pura mecânica da minimização por mínimos quadrados: mais graus de liberdade sempre permitem, no mínimo, ajuste igual. R² ajustado penaliza a adição de preditores que não contribuem proporcionalmente à perda de graus de liberdade, o que o torna a métrica correta para **comparar modelos com números diferentes de preditores** — nunca compare R² comum entre um modelo com 2 preditores e outro com 8.

```
summary(modelo_simples)$adj.r.squared
```

Na regressão simples, com $p = 1$ e n grande, a diferença entre R² e R² ajustado é pequena — a distinção importa de verdade a partir da regressão múltipla (seção 5).

---

5. Regressão linear múltipla

5.1 O modelo geral

$$y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \ldots + \beta_p x_{pi} + \varepsilon_i$$

```
modelo_multiplo <- lm(receita_mensal ~ fluxo_clientes_mensal + ticket_medi
o +
                        gasto_marketing + area_m2, data = varejo)
summary(modelo_multiplo)
```

5.2 Interpretação dos coeficientes parciais

A diferença crítica em relação à regressão simples: cada β_j na regressão múltipla é interpretado **"tudo o mais constante" (ceteris paribus)** — o efeito esperado de uma unidade a mais em x_j sobre y, **mantendo todos os outros preditores fixos**. Isso é estatisticamente diferente do β_1 de uma regressão simples de y contra x_j isolado, porque a regressão múltipla já "descontou" a parte de x_j que é explicada pelos outros preditores.

Consequência prática direta neste dataset: o β de `fluxo_clientes_mensal` na regressão múltipla (controlando por `gasto_marketing`) tende a ser **diferente** do β_1 da regressão simples do Exercício 2, porque parte do que a regressão simples atribuía a `fluxo_clientes_mensal` sozinho estava, na verdade, compartilhado com `gasto_marketing` (o confundimento via `porte`, seção 1.5).

5.3 Variáveis categóricas em regressão

O plano original deste módulo não previa explicitamente esta seção, mas ela é indispensável na prática: `porte`, `regiao` e `tipo_loja` são preditores potencialmente relevantes e são categóricos. O R lida com isso automaticamente quando a coluna é `factor` — cria variáveis **dummy (indicadoras)**, uma para cada nível exceto o primeiro (nível de referência/baseline):

```
modelo_categorico <- lm(receita_mensal ~ area_m2 + porte, data = varejo)
summary(modelo_categorico)
```

Com `porte` tendo níveis `Pequena`, `Media`, `Grande` (nessa ordem, definida no Módulo 1), o output mostra coeficientes `porteMedia` e `porteGrande` — **não** `portePequena`, porque `Pequena` vira automaticamente a categoria de

referência (o primeiro nível do factor). A interpretação: `porteMedia` é a diferença esperada em `receita_mensal` entre uma loja `Media` e uma loja `Pequena`, **de mesma área** (`area_m2` mantida constante); `porteGrande`, o mesmo raciocínio contra a referência `Pequena`.

Controlando qual nível é a referência — por padrão é o primeiro nível alfabético/factor; para forçar outro:

```
varejo$porte <- relevel(varejo$porte, ref = "Media")
```

Isso é puramente sobre qual comparação fica explícita no output — não muda o ajuste do modelo, os valores preditos, nem o R^2, apenas reparametriza qual contraste aparece direto na tabela de coeficientes.

5.4 Multicolinearidade

O problema: quando dois ou mais preditores são fortemente correlacionados entre si (não com y — entre eles), o modelo tem dificuldade em separar o efeito individual de cada um. Isso não é uma falha de cálculo do R — é uma limitação matemática real: se `fluxo_clientes_mensal` e `gasto_marketing` sobem e descem quase juntos (porque ambos seguem `porte`), o modelo não consegue distinguir com precisão "quanto da receita vem do fluxo" vs. "quanto vem do marketing" — existem infinitas combinações de β_{fluxo} e $\beta_{marketing}$ que ajustam quase igualmente bem aos dados.

Sintomas: coeficientes com erro padrão inflado (intervalos de confiança largos), coeficientes que mudam de sinal ou magnitude de forma instável quando você adiciona/remove outro preditor correlacionado, coeficientes individualmente não-significativos mesmo com o modelo como um todo (teste F) sendo altamente significativo.

O que multicolinearidade NÃO prejudica: a capacidade preditiva do modelo como um todo (R^2, previsões em novos dados dentro do mesmo padrão de correlação) permanece intacta. O problema é especificamente de **interpretação dos coeficientes individuais**, não de qualidade preditiva agregada.

Diagnóstico formal — VIF (Variance Inflation Factor):

$$ VIF_j = \frac{1}{1 - R_j^2} $$

Onde R_j^2 é o R^2 obtido regredindo o preditor x_j contra **todos os outros preditores** do modelo (não contra y). Quanto mais um preditor é "explicado" pelos demais, maior seu VIF.

```
install.packages("car")
library(car)

vif(modelo_multiplo)
```

Regra prática (não é lei, é convenção): $VIF > 5$ merece atenção, $VIF > 10$ é geralmente tratado como problema sério. Neste dataset, espera-se que `fluxo_clientes_mensal` e `gasto_marketing` apresentem VIF elevado no modelo com os 4 preditores — exatamente o padrão de confundimento por `porte` descrito na seção 1.5, agora quantificado.

O que fazer diante de VIF alto: (1) remover um dos preditores redundantes, guiado por qual é teoricamente mais próximo da causa real; (2) combinar os preditores correlacionados em um índice único (link direto com PCA, Módulo 7 — PCA é, entre outras coisas, uma técnica formal para lidar com exatamente esse tipo de redundância entre variáveis); (3) se o objetivo é só prever (não interpretar coeficientes individuais), aceitar a multicolinearidade e não fazer nada — o R^2 e as previsões continuam válidos.

5.5 Termos de interação (extensão)

Também fora do escopo literal do plano original, mas necessário para modelar corretamente este dataset: a fórmula geradora de `receita_mensal` no script (`gerar_datasets_sinteticos.R`) inclui o termo `0.9 * fluxo_clientes_mensal * (ticket_medio / 60)` — um produto entre duas variáveis, não uma soma. Isso é uma **interação**: o efeito de `fluxo_clientes_mensal` sobre `receita_mensal` depende do valor de `ticket_medio` (lojas com ticket médio maior convertem cada cliente adicional em mais receita que lojas com ticket médio baixo). Um modelo puramente aditivo (`fluxo + ticket` , sem interação) não consegue capturar isso — ele assume que o efeito de `fluxo` é o mesmo independente do `ticket` .

```
modelo_interacao <- lm(receita_mensal ~ fluxo_clientes_mensal * ticket_med
io +
                         gasto_marketing + area_m2, data = varejo)
summary(modelo_interacao)
```

Na sintaxe de fórmula do R, `a * b` expande automaticamente para `a + b + a:b` (os dois efeitos principais mais o termo de interação); `a:b` sozinho inclui só a interação, sem os efeitos principais — quase nunca é o que você quer (viola o **princípio da hierarquia**: interpretar uma interação sem os efeitos principais correspondentes tende a ser enganoso).

Comparação formal entre o modelo aditivo e o modelo com interação usa `anova()` (teste F de modelos aninhados — teoria completa no Módulo 9):

```
anova(modelo_multiplo, modelo_interacao)
```

Um p-valor baixo nesse teste indica que adicionar o termo de interação melhora o ajuste de forma que não é explicável só por acaso — esperado neste dataset, dado que a interação está na fórmula geradora.

---

6. Regressão polinomial

6.1 Quando a relação não é linear

`taxa_conversao` foi construída como função **saturante** de `fluxo_clientes_mensal` (ver `gerar_datasets_sinteticos.R`: $0.05 + 0.30(1 - e^{-fluxo/5000})$) — cresce rápido em fluxo baixo e achata perto de um teto, o padrão clássico de rendimento marginal decrescente. Uma reta força uma taxa de crescimento constante sobre uma relação que, na realidade, desacelera — o Exercício 3 do Módulo 1 já expôs isso visualmente comparando `loess` com `lm`.

Regressão polinomial estende o modelo linear incluindo potências do preditor:

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \ldots + \beta_k x_i^k + \varepsilon_i$$

Ponto conceitual importante: isso continua sendo um **modelo linear** no sentido estatístico — "linear" se refere à linearidade nos **coeficientes** β, não à forma geométrica da curva resultante. x^2 é só mais uma coluna de preditor computada a partir de x; o ajuste continua sendo OLS padrão.

6.2 Ajustando no R

```
modelo_quad <- lm(taxa_conversao ~ poly(fluxo_clientes_mensal, degree = 2)
, data = varejo)
summary(modelo_quad)
```

`poly(x, degree = k)` por padrão gera **polinômios ortogonais** — as colunas resultantes (x, x^2, ..., transformadas) são construídas para serem não-correlacionadas entre si. Isso resolve de fábrica a multicolinearidade severa que existiria entre x e x^2 brutos (que são naturalmente muito correlacionados quando x não está centrado em zero). O custo: os

coeficientes de `poly()` ortogonal não têm interpretação direta em unidades originais — servem para teste de significância e para gerar previsões, não para leitura direta tipo "cada m² adiciona X reais".

Para coeficientes interpretáveis em unidade original (ao custo de reintroduzir a colinearidade entre x e x^2, que aqui é aceitável porque o objetivo passa a ser previsão, não interpretação de cada coeficiente isolado):

```
modelo_quad_raw <- lm(taxa_conversao ~ poly(fluxo_clientes_mensal, degree
= 2, raw = TRUE), data = varejo)
```

6.3 Visualizando o ajuste

```
ggplot(varejo, aes(x = fluxo_clientes_mensal, y = taxa_conversao)) +
  geom_point(alpha = 0.25) +
  geom_smooth(method = "lm", formula = y ~ poly(x, 2), color =
"firebrick") +
  labs(title = "Regressão polinomial (grau 2): taxa de conversão vs.
fluxo",
       x = "Fluxo de clientes/mês", y = "Taxa de conversão")
```

6.4 Riscos: overfitting e extrapolação

Overfitting: aumentar o grau do polinômio sempre aumenta R^2 (mesma lógica da seção 4.3 — mais parâmetros, ajuste igual ou melhor dentro da amostra), mas um polinômio de grau alto tende a "perseguir" ruído específico da amostra em vez da tendência real subjacente — a curva fica ondulada de forma artificial entre os pontos observados. Compare visualmente grau 2 contra grau 6 no mesmo scatter para ver esse efeito diretamente.

Extrapolação: este é o risco mais sério e menos intuitivo. Um polinômio ajustado dentro do range observado de x pode ter comportamento completamente absurdo **fora** desse range — uma curva de grau par pode disparar para $+\infty$ ou $-\infty$ logo após o último ponto observado, mesmo que dentro dos dados o ajuste pareça excelente. Isso é qualitativamente

diferente do risco de extrapolação numa reta simples (que pelo menos continua crescendo/decrescendo de forma previsível) — polinômios de grau ≥ 2 podem inverter direção de forma artificial fora da amostra. **Nunca use um modelo polinomial para prever fora do range de `fluxo_clientes_mensal` observado nos dados de treino.**

```
range(varejo$fluxo_clientes_mensal)   # limite seguro de previsão
```

Comparação de graus via `anova()` (mesma lógica de modelos aninhados da seção 5.5):

```
modelo_linear <- lm(taxa_conversao ~ fluxo_clientes_mensal, data = varejo)
modelo_cubico <- lm(taxa_conversao ~ poly(fluxo_clientes_mensal, degree =
3), data = varejo)

anova(modelo_linear, modelo_quad, modelo_cubico)
```

Se o termo cúbico não melhora o ajuste de forma estatisticamente distinguível do quadrático, prefira o modelo mais simples (grau 2) — princípio da parcimônia: entre modelos com poder explicativo equivalente, o mais simples generaliza melhor e é mais fácil de interpretar e comunicar.

---

7. Diagnóstico de resíduos — checklist prático e testes formais

Antes de aceitar qualquer modelo de regressão (simples, múltiplo ou polinomial) deste módulo, rode sempre:

```
par(mfrow = c(2, 2))
plot(modelo_multiplo)
par(mfrow = c(1, 1))

# Se houver preditores categóricos ou múltiplos preditores, vale plotar
# resíduos contra cada preditor individualmente também:
varejo$residuos <- resid(modelo_multiplo)
ggplot(varejo, aes(x = ticket_medio, y = residuos)) +
  geom_point(alpha = 0.3) +
  geom_hline(yintercept = 0, color = "firebrick") +
  labs(title = "Resíduos vs. ticket_medio — checando padrão não
capturado")
```

Se `residuos` mostrar um padrão sistemático contra `ticket_medio` no modelo aditivo da seção 5.2 (sem a interação da seção 5.5), isso é exatamente o sinal visual de que falta o termo de interação — o resíduo "sabe" que há estrutura não capturada, mesmo que o R^2 geral pareça aceitável. A leitura visual dos 4 gráficos de `plot(modelo)` é o primeiro passo, mas é subjetiva — as subseções seguintes formalizam numericamente os dois problemas mais comuns que a leitura visual só sugere: heterocedasticidade e pontos que distorcem o modelo sozinhos.

7.1 Teste de Breusch-Pagan (heterocedasticidade)

O gráfico Scale-Location (seção 3.4) sugere heterocedasticidade visualmente — "parece" que a dispersão dos resíduos cresce com os valores ajustados. O teste de Breusch-Pagan formaliza essa suspeita: ele regride os resíduos ao quadrado contra os preditores do modelo e testa se essa regressão auxiliar tem poder explicativo significativo (se tiver, a variância do erro não é constante — ela depende de *x*).

```
install.packages("lmtest")
library(lmtest)

bptest(modelo_multiplo)
```

H_0: homocedasticidade (variância constante). Um p-valor baixo (< 0.05) rejeita H_0 — evidência de heterocedasticidade. A teoria formal de teste de hipótese e p-valor só fecha no Módulo 9; por ora, leia como "esse padrão que pareceu visualmente suspeito é estatisticamente improvável de ser só ruído amostral".

O que fazer diante de heterocedasticidade confirmada: (1) transformar *y* (log é o mais comum quando a variável resposta é financeira/positiva e cresce em escala — ver Box-Cox, seção 7.3, para escolher a transformação certa em vez de chutar log por hábito); (2) usar erros padrão robustos a heterocedasticidade (`sandwich::vcovHC()`), que corrigem a inferência sem mudar os coeficientes estimados; (3) para dados em painel/série temporal, modelos que já assumem variância não constante (fora do escopo deste módulo). A heterocedasticidade não invalida os coeficientes estimados em si (ainda são não-viesados sob OLS), mas invalida os erros padrão, testes t e intervalos de confiança reportados por `summary()` — eles assumem homocedasticidade para serem corretos.

7.2 Pontos influentes: distância de Cook e DFBETAS

O gráfico Residuals vs. Leverage (seção 3.4) já cita distância de Cook visualmente. Aqui ela é calculada e usada com um critério numérico, não só lida no gráfico:

```
cook <- cooks.distance(modelo_multiplo)

# Regra prática mais comum: pontos com distância de Cook > 4/n merecem
investigação
limite_cook <- 4 / nrow(varejo)
pontos_influentes <- which(cook > limite_cook)
length(pontos_influentes)

varejo[pontos_influentes, ] %>% select(loja_id, receita_mensal,
fluxo_clientes_mensal, ticket_medio)
```

Distância de Cook mede **o quanto os coeficientes do modelo mudariam se aquela observação específica fosse removida** — não é sobre o resíduo ser grande sozinho, é sobre a combinação de resíduo grande **e** alavancagem alta (*x* atípico) que dá a uma única observação poder desproporcional sobre o ajuste inteiro. Um outlier em *y* com *x* típico tem influência limitada; o mesmo outlier com *x* extremo pode puxar a reta inteira.

DFBETAS vai um passo além: em vez de medir o impacto agregado sobre o modelo (Cook), mede **quanto cada coeficiente individual mudaria** se aquela observação fosse removida — útil quando você quer saber não só "essa observação é influente" mas "essa observação está distorcendo especificamente o coeficiente de `gasto_marketing`, ou de `area_m2`":

```
dfbetas_modelo <- dfbetas(modelo_multiplo)
head(dfbetas_modelo)

# Regra prática: |DFBETA| > 2/sqrt(n) merece atenção para aquele
coeficiente específico
limite_dfbeta <- 2 / sqrt(nrow(varejo))
which(abs(dfbetas_modelo[, "gasto_marketing"]) > limite_dfbeta)
```

O que fazer diante de pontos influentes confirmados: nunca remover automaticamente só por ultrapassar o limiar — investigar primeiro se é erro de entrada de dado (aí sim corrigir/remover), ou uma observação real e válida que só é rara (aí o ponto pode estar revelando algo genuíno sobre o fenômeno, e removê-lo às cegas seria descartar informação real para deixar o modelo mais confortável). Reportar o resultado com e sem o ponto influente, quando a decisão não for óbvia, é mais honesto que decidir silenciosamente.

7.3 Transformação de variável: Box-Cox

Quando heterocedasticidade ou não-normalidade dos resíduos persiste, a transformação de *y* costuma resolver — mas "tentar log porque geralmente funciona" é um chute. Box-Cox estima, a partir dos próprios dados, qual transformação (dentro de uma família parametrizada por λ) estabiliza melhor a variância:

$$ y^{(\lambda)} = \begin{cases} \dfrac{y^\lambda - 1}{\lambda} & \lambda \neq 0 \\ \ln(y) & \lambda = 0 \end{cases} $$

```
library(MASS)

bc <- boxcox(modelo_multiplo, plotit = TRUE)
lambda_otimo <- bc$x[which.max(bc$y)]
lambda_otimo
```

O gráfico mostra a log-verossimilhança para uma faixa de λ; o pico indica o valor ótimo. Valores de referência: λ ≈ 0 sugere log; λ ≈ 0.5 sugere raiz quadrada; λ ≈ 1 sugere que **nenhuma transformação é necessária** (o intervalo de confiança em torno do λ ótimo, mostrado no gráfico, geralmente inclui 1 nesse caso). Box-Cox só se aplica a *y* estritamente positivo — para variáveis com zeros ou negativos, existe a extensão Yeo-Johnson (`car::powerTransform`, não coberta aqui).

---

8. Diferenças-chave vindas de Python

Conceito	Python (statsmodels / sklearn)
Correlação simples	`df['x'].corr(df['y'])` ou `scipy.stats.pearsonr(x, y)` (retorna r e p-valor junto)

Conceito	Python (statsmodels / sklearn)
Matriz de correlação	`df.corr()`
Regressão linear (inferência)	`statsmodels.formula.api.ols("y ~ x", data=df).fit()`
Regressão linear (predição/ML)	`sklearn.linear_model.LinearRegression()` — sem tabela de p-valor por padrão
R^2	`model.rsquared` (statsmodels) / `model.score()` (sklearn)
R^2 ajustado	`model.rsquared_adj` (statsmodels)
Variável categórica	`pd.get_dummies(df, columns=['col'])` manual, ou `C(col)` na fórmula do statsmodels
VIF	`statsmodels.stats.outliers_influence.variance_inflation_factor` (uma linha por variável, mais verboso)
Regressão polinomial	`sklearn.preprocessing.PolynomialFeatures(degree=k)` + pipeline
Comparação de modelos aninhados (F-test)	`statsmodels` permite via `anova_lm(model1, model2)`
Fórmula de interação	`"y ~ x1 * x2"` no statsmodels (mesma sintaxe, na prática)
Heterocedasticidade (Breusch-Pagan)	`statsmodels.stats.diagnostic.het_breuschpagan`
Distância de Cook	`model.get_influence().cooks_distance` (statsmodels)
Box-Cox	`scipy.stats.boxcox`

Conceito	Python (statsmodels / sklearn)
Saída de modelo em formato tabular (tibble/DataFrame)	`model.summary2().tables[1]` (statsmodels, mais manual)

Nota sobre `broom`: vale a pena adotar cedo. `broom::tidy(modelo)` retorna a tabela de coeficientes como tibble (em vez do objeto `summary()` que é incômodo de manipular programaticamente); `broom::glance(modelo)` retorna métricas de ajuste (R^2, AIC, BIC) em uma linha; `broom::augment(modelo)` retorna o data frame original com colunas extras já anexadas (`.fitted`, `.resid`, `.cooksd`, `.hat`) — substitui boa parte do código manual das seções 7.1–7.2 por uma única chamada. Isso se conecta ao ecossistema `tidymodels`, que padroniza fluxo de treino/avaliação de qualquer modelo (regressão, classificação, etc.) — mencionado aqui como próximo passo natural, não coberto em profundidade neste plano.

O padrão mais relevante para quem vem de Python/sklearn: `sklearn.LinearRegression()` é otimizado para **predição** (não expõe erro padrão, p-valor ou intervalo de confiança de coeficiente por padrão — é preciso ir para `statsmodels` para isso). `lm()` do R já nasce do lado de **inferência**: todo `summary(lm())` traz erro padrão, teste t e p-valor de cada coeficiente de fábrica, porque a tradição estatística do R prioriza "o coeficiente é confiável?" sobre "o modelo prediz bem?". Os dois objetivos não são excludentes, mas os dois ecossistemas historicamente enfatizam lados diferentes — vale ter isso em mente ao portar workflow de um pra outro.

---

9. Exercícios

Use `varejo_sintetico.csv` e `painel_regional_varejo.csv`. Carregue `dplyr` e `ggplot2` antes de começar.

Exercício 1 — Matriz de correlação e leitura crítica

Construa a matriz de correlação entre `area_m2`, `fluxo_clientes_mensal`, `ticket_medio`, `gasto_marketing` e `receita_mensal`. Identifique o par com maior correlação (fora da diagonal). Esse par tem relação causal direta, ou o valor alto é explicado por uma terceira variável em comum? Justifique usando o que você sabe da construção do dataset.

Solução:

```
vars_num <- varejo %>% select(area_m2, fluxo_clientes_mensal,
ticket_medio,
                              gasto_marketing, receita_mensal)
round(cor(vars_num), 2)
```

O par `fluxo_clientes_mensal` × `gasto_marketing` tende a apresentar a correlação mais alta entre os preditores (não entre um preditor e `receita_mensal`). Essa correlação **não é causal direta** — nada no script gerador faz `gasto_marketing` influenciar `fluxo_clientes_mensal` ou vice-versa. Ambos são gerados a partir dos parâmetros-base de `porte` (`base_fluxo` e `base_marketing`, ambos crescentes de Pequena → Grande) — `porte` é a causa comum (confounder) que produz a correlação observada entre os dois. Isso é o mesmo padrão descrito na seção 1.5, agora verificado numericamente.

Exercício 2 — Regressão simples e comparação com r²

Ajuste `receita_mensal ~ fluxo_clientes_mensal`. Reporte $\hat{\beta}_0$, $\hat{\beta}_1$ e R^2. Confirme que R^2 é igual ao quadrado do *r* de Pearson entre as duas variáveis (calculado no Exercício 1). Interprete $\hat{\beta}_1$ em português simples (uma frase, unidade explícita).

Solução:

```
modelo1 <- lm(receita_mensal ~ fluxo_clientes_mensal, data = varejo)
summary(modelo1)

r_pearson <- cor(varejo$fluxo_clientes_mensal, varejo$receita_mensal)
r_pearson^2
summary(modelo1)$r.squared   # deve bater com r_pearson^2
```

Interpretação de $\hat{\beta}_1$: "cada cliente adicional no fluxo mensal médio está associado a um aumento esperado de aproximadamente R$ [valor de β̂1] na receita mensal da loja, em média, nesta amostra." Note a linguagem: "associado a", não "causa" — a regressão simples aqui não controla por `ticket_medio` nem `gasto_marketing`, então parte desse coeficiente está absorvendo efeito de variáveis correlacionadas (Exercício 3 isola isso).

Exercício 3 — Multicolinearidade na prática

Ajuste um modelo múltiplo `receita_mensal ~ fluxo_clientes_mensal + gasto_marketing + ticket_medio + area_m2`. Calcule o VIF de cada preditor com `car::vif()`. Compare o coeficiente de `fluxo_clientes_mensal` neste modelo com o da regressão simples do Exercício 2 — ele mudou? Isso é esperado, dado o Exercício 1?

Solução:

```
library(car)

modelo3 <- lm(receita_mensal ~ fluxo_clientes_mensal + gasto_marketing +
                ticket_medio + area_m2, data = varejo)
summary(modelo3)
vif(modelo3)
```

O coeficiente de `fluxo_clientes_mensal` tende a mudar de magnitude entre os dois modelos — no modelo múltiplo, ele passa a representar o efeito de `fluxo_clientes_mensal` **controlando** por `gasto_marketing`, isolando a parte que não é compartilhada com o confundimento do Exercício 1. `fluxo_clientes_mensal` e `gasto_marketing` devem apresentar os VIFs mais altos entre os quatro preditores — consequência direta e esperada da correlação identificada no Exercício 1. Isso não invalida o modelo para fins de previsão, mas exige cautela ao interpretar o coeficiente de qualquer um dos dois preditores isoladamente.

Exercício 4 — Variável categórica e nível de referência

Ajuste `receita_mensal ~ area_m2 + porte`. Identifique qual nível de `porte` é a referência automática. Reinterprete o modelo trocando a referência para `"Grande"` com `relevel()` e confirme que os valores preditos (`fitted(modelo)`) não mudam, apenas a tabela de coeficientes.

Solução:

```
modelo4 <- lm(receita_mensal ~ area_m2 + porte, data = varejo)
summary(modelo4)   # referência: Pequena (primeiro nível do factor)

varejo_releveled <- varejo
varejo_releveled$porte <- relevel(varejo_releveled$porte, ref = "Grande")
modelo4_releveled <- lm(receita_mensal ~ area_m2 + porte, data =
varejo_releveled)
summary(modelo4_releveled)

# Confirmação: valores ajustados idênticos, independente da referência
escolhida
all.equal(fitted(modelo4), fitted(modelo4_releveled))
```

A referência original é `Pequena` (definida pela ordem de `levels()` no Módulo 1). Trocar a referência muda **quais contrastes aparecem na tabela** (agora `porteMedia` e `portePequena` são lidos contra `Grande`), mas o modelo é matematicamente o mesmo — mesmas previsões, mesmo R^2, mesmos resíduos. É só reparametrização da mesma superfície ajustada.

Exercício 5 — Regressão polinomial vs. linear

Para `taxa_conversao ~ fluxo_clientes_mensal`, ajuste um modelo linear simples e um modelo polinomial de grau 2. Compare R^2 dos dois. Rode `anova(modelo_linear, modelo_quadratico)` — o termo quadrático é justificado estatisticamente? Plote os dois ajustes sobrepostos no mesmo scatter.

Solução:

```
mod_lin  <- lm(taxa_conversao ~ fluxo_clientes_mensal, data = varejo)
mod_quad <- lm(taxa_conversao ~ poly(fluxo_clientes_mensal, degree = 2), d
ata = varejo)

summary(mod_lin)$r.squared
summary(mod_quad)$r.squared
anova(mod_lin, mod_quad)

ggplot(varejo, aes(x = fluxo_clientes_mensal, y = taxa_conversao)) +
  geom_point(alpha = 0.2) +
  geom_smooth(method = "lm", formula = y ~ x, color = "steelblue",
              linetype = "dashed", se = FALSE) +
  geom_smooth(method = "lm", formula = y ~ poly(x, 2), color =
"firebrick", se = FALSE) +
  labs(title = "Linear (azul tracejado) vs. quadrático (vermelho)",
       x = "Fluxo de clientes/mês", y = "Taxa de conversão")
```

O modelo quadrático deve apresentar R^2 visivelmente maior, e o teste `anova()` deve indicar que a melhora não é atribuível só a acaso amostral (p-valor baixo) — esperado, já que `taxa_conversao` foi construída com uma função saturante (seção 6.1), que uma parábola aproxima melhor que uma reta dentro do range observado. Vale registrar: mesmo o ajuste quadrático é uma aproximação de uma forma exponencial saturante — não é o modelo "verdadeiro", só uma aproximação polinomial local razoável dentro do range de dados observado.

Exercício 6 (desafio) — Correlação espúria por fator regional compartilhado

Usando `painel_regional_varejo.csv`, reduza o painel ao nível de cidade (`distinct(cidade_id, polo, indice_renda_media, ticket_medio_regional)`) e calcule a correlação bruta entre `indice_renda_media` e `ticket_medio_regional`. Depois, calcule a mesma correlação **dentro de cada `polo`** separadamente (`group_by(polo) %>% summarise(cor = cor(...))`). A correlação dentro de cada polo é mais fraca que a correlação bruta agregada? O que isso revela sobre a causa da correlação observada em nível de cidade?

Solução:

```
painel <- read.csv("painel_regional_varejo.csv", stringsAsFactors = FALSE)

cidades_unicas <- painel %>%
  distinct(cidade_id, polo, indice_renda_media, ticket_medio_regional)

# Correlação bruta, ignorando polo
cor(cidades_unicas$indice_renda_media, cidades_unicas$ticket_medio_regiona
l)

# Correlação dentro de cada polo
cidades_unicas %>%
  group_by(polo) %>%
  summarise(r_dentro_polo = cor(indice_renda_media,
ticket_medio_regional), n = n())
```

A correlação bruta (nível cidade, ignorando `polo`) tende a ser sensivelmente mais forte que a correlação calculada dentro de cada polo isoladamente — e isso não é coincidência do dataset, é construção deliberada: `ticket_medio_regional` no script gerador (`gerar_datasets_sinteticos.R`) é montado com peso `0.7` sobre o mesmo `polo_efeito_renda` que constrói `indice_renda_media`, mais ruído independente com peso menor. Ou seja, as duas variáveis compartilham um fator latente regional (`polo`) — cidades do mesmo polo tendem a ter tanto renda quanto ticket puxados na mesma direção pelo efeito daquele polo especificamente, **independente** de qualquer relação causal direta entre renda e ticket dentro da cidade. Ao

condicionar por `polo` (calcular a correlação dentro de cada grupo), você remove o efeito do confundidor e a correlação residual — a que sobra depois de controlar pelo fator regional — tende a se aproximar de algo mais fraco, mais próximo do ruído genuíno (`0.3 * rnorm`) que também está na fórmula. Este é exatamente o problema que o Módulo 11 formaliza como **pseudo-replicação espacial**: correlacionar variáveis por unidade geográfica sem testar/controlar autocorrelação espacial infla a aparência de relação, porque observações vizinhas (mesmo polo) não são estatisticamente independentes entre si.

Exercício 7 — Diagnóstico formal completo

Usando `modelo_multiplo` (Exercício 3): (a) rode `bptest()` e conclua se há evidência de heterocedasticidade; (b) calcule a distância de Cook e liste quantas observações ultrapassam `4/n` ; (c) rode `boxcox()` sobre o modelo e reporte o λ ótimo — isso sugere alguma transformação de `receita_mensal` ? (d) gere o intervalo de predição para uma loja hipotética com `fluxo_clientes_mensal = 6000` , `ticket_medio = 45` , `gasto_marketing = 2000` , `area_m2 = 150` e compare a largura com o intervalo de confiança equivalente.

Solução:

```
library(lmtest)
library(MASS)

modelo_multiplo <- lm(receita_mensal ~ fluxo_clientes_mensal + ticket_medi
o +
                        gasto_marketing + area_m2, data = varejo)

# (a)
bptest(modelo_multiplo)

# (b)
cook <- cooks.distance(modelo_multiplo)
limite_cook <- 4 / nrow(varejo)
sum(cook > limite_cook)

# (c)
bc <- boxcox(modelo_multiplo, plotit = FALSE)
lambda_otimo <- bc$x[which.max(bc$y)]
lambda_otimo

# (d)
novo <- data.frame(fluxo_clientes_mensal = 6000, ticket_medio = 45,
                   gasto_marketing = 2000, area_m2 = 150)
ic <- predict(modelo_multiplo, newdata = novo, interval = "confidence")
ip <- predict(modelo_multiplo, newdata = novo, interval = "prediction")
ic; ip
(ip[, "upr"] - ip[, "lwr"]) / (ic[, "upr"] - ic[, "lwr"])
# quantas vezes o IP é mais largo
```

Espera-se: (a) `bptest()` provavelmente rejeita homocedasticidade, já que `receita_mensal` é uma variável monetária que tende a ter dispersão crescente em magnitude — consistente com a leitura visual do Scale-Location já discutida na seção 3.4; (b) um pequeno número de observações (tipicamente poucos por cento do total) deve ultrapassar o limiar de Cook — vale inspecionar se são lojas atípicas legítimas (`porte = "Grande"` com valores extremos de preditor) ou candidatas a erro de dado; (c) se `bptest` confirmou heterocedasticidade, é esperado que λ ótimo fique próximo de 0 (sugerindo log) ou abaixo de 1 — reforçando de forma numérica o que o teste de heterocedasticidade já indicou qualitativamente; (d) o intervalo de predição deve ser substancialmente mais largo que o de confiança — a proporção exata depende dos dados, mas a diferença ilustra concretamente por que confundir os dois leva a subestimar a incerteza ao prever uma loja específica.

10. Entrega esperada do módulo

Script `modelagem_bivariada_varejo.R` que recebe `varejo_sintetico.csv` e produz automaticamente:

1. Matriz de correlação das variáveis numéricas principais + heatmap (`corrplot` ou `ggcorrplot`)
2. Modelo de regressão múltipla `receita_mensal ~ fluxo_clientes_mensal + ticket_medio + gasto_marketing + area_m2 + porte`, com tabela de VIF e os 4 gráficos de diagnóstico de resíduos salvos como imagem
3. Diagnóstico formal do modelo múltiplo: `bptest()`, distância de Cook (lista de observações acima do limiar), `boxcox()` com λ ótimo reportado
4. Modelo polinomial `taxa_conversao ~ poly(fluxo_clientes_mensal, 2)` com gráfico comparando ajuste linear vs. quadrático
5. Intervalo de confiança e de predição para pelo menos uma observação hipotética, com a diferença de largura explicitamente comentada

11. Critério de domínio antes de avançar ao Módulo 6

- ☐ Explica de cabeça por que $R^2 = r^2$ na regressão simples, mas não em regressão múltipla
- ☐ Consegue interpretar corretamente um coeficiente de regressão múltipla usando a linguagem “ceteris paribus” / “controlando por”
- ☐ Sabe diagnosticar multicolinearidade via VIF e explicar por que ela não prejudica a previsão, só a interpretação de coeficientes individuais
- ☐ Entende por que R^2 alto não é suficiente para aceitar um modelo — sabe citar pelo menos 2 coisas que R^2 não garante
- ☐

Sabe quando suspeitar de relação não-linear (via scatter/resíduos) e decidir entre transformar a variável, usar polinômio, ou usar `loess`/não-paramétrico

- ☐ Entende o risco específico de extrapolação em modelos polinomiais (diferente do risco em reta simples)
- ☐ Consegue explicar, com um exemplo próprio (fora do dataset sintético), uma correlação espúria causada por confundimento
- ☐ Sabe rodar e interpretar Breusch-Pagan, distância de Cook/DFBETAS e Box-Cox — e sabe o que fazer (não só constatar) diante de cada resultado
- ☐ Explica a diferença entre intervalo de confiança e intervalo de predição, e sabe qual usar em cada situação de comunicação de resultado
- ☐ Todos os 7 exercícios rodam sem erro e a interpretação escrita distingue claramente “correlação/associação” de “causalidade”

MÓDULO 05 · MODELAGEM

Regressão Logística

GLM Binomial

1. Por que regressão linear não serve para resposta binária

Tome uma variável resposta $y \in \{0, 1\}$ — por exemplo, "essa loja teve queda de receita relevante no trimestre" (1) ou não (0). Ajustar `lm(y ~ x)` diretamente tem três problemas estruturais, não incidentais:

1. **Previsões fora de [0, 1].** Uma reta não tem limite inferior nem superior — para x suficientemente extremo, o modelo prevê probabilidades negativas ou maiores que 1, o que não tem interpretação como probabilidade.
2. **Heterocedasticidade intrínseca.** A variância de uma variável Bernoulli é $p(1 - p)$ — que **depende do próprio valor previsto** p. Isso viola a premissa de homocedasticidade (Módulo 4, seção 3.4) por construção matemática, não por acaso da amostra: não existe transformação que "conserte" isso mantendo o modelo linear em y diretamente.
3. **Relação tipicamente não-linear entre x e $P(y = 1)$.** Perto de $p = 0$ ou $p = 1$, o mesmo incremento em x tem efeito marginal muito menor sobre p do que perto de $p = 0.5$ — uma reta força um efeito marginal constante que não corresponde ao fenômeno.

```
library(dplyr)
library(ggplot2)

varejo <- read.csv("varejo_sintetico.csv", stringsAsFactors = TRUE)

# Regressão linear "ingênua" sobre uma binária, só para ver o problema na
prática
modelo_ingenuo <- lm(teve_reclamacao_grave ~ fluxo_clientes_mensal, data
= varejo)
range(fitted(modelo_ingenuo))   # confirme: sai de [0, 1]?
```

A solução é trocar a função de ligação: em vez de modelar *y* diretamente, modela-se uma transformação de $P(y = 1)$ que **é** ilimitada — o log-odds — e depois transforma de volta para probabilidade. Isso é exatamente o que a regressão logística faz.

---

2. O modelo logístico

2.1 Log-odds (logit) e a função sigmoide

$$ \text{logit}(p) = \ln\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 x_1 + \dots + \beta_k x_k $$

O lado esquerdo — log-odds — pode assumir qualquer valor real, positivo ou negativo, resolvendo o problema 1 da seção anterior: o modelo é linear **nos coeficientes**, só que aplicado a uma transformação de *p*, não a *p* diretamente (mesma lógica de "linear nos parâmetros" já vista na regressão polinomial, Módulo 4, seção 6.1).

Invertendo para voltar a *p*, obtém-se a **função sigmoide (logística)**:

$$ p = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \dots + \beta_k x_k)}} $$

```
ggplot(data.frame(z = seq(-6, 6, 0.1)), aes(x = z, y = 1 / (1 + exp(-
z)))) +
  geom_line(color = "steelblue", linewidth = 1) +
  labs(title = "Função sigmoide: de log-odds (eixo x) a probabilidade
(eixo y)",
       x = "log-odds (z)", y = "P(y = 1)")
```

A curva em S garante $p \in (0, 1)$ para qualquer valor de log-odds, e o efeito marginal de x sobre p é naturalmente menor nas extremidades (perto de 0 ou 1) e maior perto de $p = 0.5$ — resolve o problema 3 da seção anterior por construção geométrica.

2.2 Estimação: máxima verossimilhança, não mínimos quadrados

Diferente de OLS (Módulo 4, seção 3.2), a regressão logística é ajustada por **máxima verossimilhança** (MLE — *maximum likelihood estimation*): não existe fórmula fechada como na regressão linear; o algoritmo busca iterativamente os $\hat{\beta}$ que tornam os dados observados mais prováveis sob o modelo. Na prática, isso é transparente no R — `glm()` resolve a otimização internamente —, mas explica por que o ajuste pode, em casos raros de separação perfeita entre classes, não convergir (aviso comum: `"algorithm did not converge"` ou `"fitted probabilities numerically 0 or 1 occurred"`).

---

3. Construção da variável resposta para este módulo

`varejo_sintetico.csv` tem colunas binárias reais (`teve_reclamacao_grave`, `participou_campanha`, `participou_teste_fidelidade`), mas todas foram geradas por sorteio **independente** de qualquer outra variável (`rbinom(n, 1, p_fixo)` — ver `gerar_datasets_sinteticos.R`) especificamente para servir de base a outros módulos (IC de proporção no Módulo 8, teste t pareado e Mann-

Whitney no Módulo 6). Usar qualquer uma delas aqui ensinaria regressão logística sobre uma relação que não existe — todos os coeficientes sairiam não-significativos, o que não ilustra nada.

Por isso, este módulo constrói uma variável resposta nova, com relação real e conhecida aos preditores, para que você possa verificar se o modelo recupera a estrutura esperada — o mesmo princípio pedagógico já usado no Módulo 2 (`area_m2` como mistura de 3 normais) e no Módulo 7 (PCA sobre estrutura conhecida):

```
set.seed(2026)

# Log-odds construído deliberadamente a partir de 3 preditores:
# lojas com mais fluxo de clientes e maior porte tendem a ter mais chance
de
# "alta performance"; ticket médio alto reduz um pouco essa chance (mais
# dependente de poucos clientes de ticket alto -> mais volátil).
logit_construido <- -4.2 +
  0.00045 * varejo$fluxo_clientes_mensal +
  1.1 * (varejo$porte == "Grande") +
  0.5 * (varejo$porte == "Media") -
  0.015 * varejo$ticket_medio

prob_construida <- plogis(logit_construido)   # plogis() = função
sigmoide, inversa de qlogis()/logit
varejo$alta_performance <- rbinom(nrow(varejo), size = 1, prob =
prob_construida)

mean(varejo$alta_performance)   # proporção geral de "1"s
```

`plogis()` e `qlogis()` são as funções base do R para a sigmoide e sua inversa (o logit) — mesmo padrão `d` / `p` / `q` / `r` do Módulo 8, aplicado à distribuição logística. Guarde `logit_construido` / `prob_construida` de lado (não use como preditor) — eles existem só para você conferir, ao final, se o modelo ajustado recupera coeficientes na direção e magnitude esperadas.

---

4. Ajustando no R

```
modelo_logit <- glm(alta_performance ~ fluxo_clientes_mensal + ticket_medi
o + porte,
                    data = varejo, family = binomial)

summary(modelo_logit)
```

`glm()` generaliza `lm()` para respostas fora do caso normal contínuo — `family = binomial` especifica a distribuição da resposta (Bernoulli/binomial) e, por padrão, `link = "logit"` (existem outras ligações — `probit`, `cloglog` — fora do escopo aqui). Repare que a sintaxe de fórmula (`y ~ x1 + x2 + fator`) é idêntica à do Módulo 4 — variáveis categóricas, interações e tudo mais funcionam do mesmo jeito.

Os coeficientes de `summary(modelo_logit)` estão em escala de **log-odds** — diretamente comparáveis em sinal (positivo aumenta a chance de $y = 1$, negativo diminui), mas não interpretáveis diretamente em magnitude sem exponenciar (seção 5).

---

5. Odds ratio: a interpretação certa dos coeficientes

$OR_j = e^{\beta_j}$

```
exp(coef(modelo_logit))
exp(confint(modelo_logit))   # intervalo de confiança do odds ratio (mais
lento -- usa perfil de verossimilhança)
```

Interpretação de odds ratio: OR_j é o fator multiplicativo sobre a **chance** (odds, não probabilidade) de $y = 1$ para cada aumento de 1 unidade em x_j, mantendo os demais preditores constantes.

- $OR = 1$: preditor sem efeito.

- $OR > 1$: aumenta a chance ($OR = 2 \rightarrow$ chance dobra a cada unidade de x_j).
- $OR < 1$: diminui a chance ($OR = 0.5 \rightarrow$ chance cai pela metade).

Cuidado de interpretação, o erro mais comum: odds ratio não é razão de probabilidades. "A chance dobra" não significa "a probabilidade dobra" — se p parte de 0.4, odds $= 0.4/0.6 = 0.667$; dobrando a odds para 1.333, a nova probabilidade é $1.333/(1 + 1.333) \approx 0.571$ — cresceu, mas não dobrou. Quanto mais perto p está de 0 ou 1, mais a diferença entre "razão de odds" e "razão de probabilidades" se acentua. Reportar odds ratio como se fosse razão de risco (probabilidade) é um erro difundido até em publicação científica — vale a pena internalizar a diferença agora.

Para `porte`, o coeficiente é relativo ao nível de referência (`Pequena`, mesma lógica do Módulo 4, seção 5.3) — `OR_{Grande}` é a razão de chance de uma loja Grande vs. uma loja Pequena, mantendo `fluxo_clientes_mensal` e `ticket_medio` fixos.

---

6. Avaliando o modelo: matriz de confusão e métricas de classificação

Diferente de regressão linear (onde R^2/R^2 ajustado avaliam ajuste), classificação binária exige comparar previsão categórica contra realidade categórica.

```
prob_prevista <- predict(modelo_logit, type = "response")   # type =
"response" retorna p, não o log-odds
classe_prevista <- ifelse(prob_prevista > 0.5, 1, 0)

matriz_confusao <- table(Observado = varejo$alta_performance, Previsto =
classe_prevista)
matriz_confusao
```

	Previsto 0	Previsto 1
Observado 0	Verdadeiro Negativo (VN)	Falso Positivo (FP)
Observado 1	Falso Negativo (FN)	Verdadeiro Positivo (VP)

```
VN <- matriz_confusao["0", "0"]; FP <- matriz_confusao["0", "1"]
FN <- matriz_confusao["1", "0"]; VP <- matriz_confusao["1", "1"]

acuracia    <- (VP + VN) / sum(matriz_confusao)
sensibilidade <- VP / (VP + FN)   # recall / taxa de verdadeiro positivo
-- "dos que são 1, quantos acertei?"
especificidade <- VN / (VN + FP) # "dos que são 0, quantos acertei?"
precisao    <- VP / (VP + FP)     # "dos que previ como 1, quantos
realmente são?"

c(acuracia = acuracia, sensibilidade = sensibilidade,
  especificidade = especificidade, precisao = precisao)
```

Por que acurácia sozinha engana: se `alta_performance` tiver proporção desbalanceada (ex.: 10% de “1”s), um modelo que sempre prevê “0” já acerta 90% — acurácia alta sem informação nenhuma. Sensibilidade e especificidade juntas contam a história real: um modelo pode ter acurácia decente e ainda assim errar sistematicamente a classe minoritária (sensibilidade baixa). Qual métrica priorizar depende do custo assimétrico do erro no domínio — em saúde, geralmente FN (deixar de detectar doença) é mais caro que FP; em detecção de fraude, depende do contexto de negócio específico.

---

7. Curva ROC e AUC

O corte de 0.5 usado na seção 6 é arbitrário — a curva ROC mostra o trade-off entre sensibilidade e especificidade **para todos os cortes possíveis** de uma vez, sem fixar um único ponto.

```
install.packages("pROC")
library(pROC)

roc_obj <- roc(varejo$alta_performance, prob_prevista)
plot(roc_obj, main = "Curva ROC")
auc(roc_obj)
```

Eixo Y: sensibilidade (taxa de verdadeiro positivo). Eixo X (invertido por convenção): especificidade. Uma curva colada na diagonal (*AUC* = 0.5) indica modelo sem poder discriminativo — equivalente a sortear a classe; uma curva colada no canto superior esquerdo (*AUC* → 1) indica separação quase perfeita entre as classes.

AUC (Area Under the Curve) resume a curva inteira num único número entre 0.5 e 1: a probabilidade de que o modelo atribua uma probabilidade prevista maior a uma observação positiva sorteada aleatoriamente do que a uma observação negativa sorteada aleatoriamente. Referência prática comum (mesma cautela de "regra de bolso, não lei" do Módulo 4 para VIF): *AUC* < 0.7 fraco, 0.7–0.8 aceitável, 0.8–0.9 bom, > 0.9 excelente — mas sempre interprete no contexto do domínio, não como corte universal.

7.1 Escolhendo o ponto de corte

```
coords(roc_obj, "best", ret = c("threshold", "sensitivity", "specificity")
)
```

`"best"` por padrão usa o índice de Youden (maximiza sensibilidade + especificidade − 1), mas o corte ótimo de negócio raramente coincide com o corte estatístico "ótimo" — se o custo de um falso negativo é muito maior que o de um falso positivo (ex.: deixar passar uma loja em risco real de fechar), vale deliberadamente escolher um corte mais baixo que 0.5, sacrificando especificidade por sensibilidade.

---

8. Diferenças-chave vindas de Python

Conceito	Python (statsmodels / sklearn)	R
Ajuste do modelo	`statsmodels.api.Logit(y, X).fit()` ou `sklearn.linear_model.LogisticRegression()`	`glm(y ~ x, fami` `binomial)`
Coeficientes em log-odds	`.params` (statsmodels)	`coef(modelo)`
Odds ratio	`np.exp(model.params)` (manual)	`exp(coef(modelo`
Probabilidade prevista	`model.predict()` (statsmodels) / `model.predict_proba()[:, 1]` (sklearn)	`predict(modelo,` `= "response")`
Matriz de confusão	`sklearn.metrics.confusion_matrix`	`table(observado` `previsto)`
Curva ROC / AUC	`sklearn.metrics.roc_curve` / `roc_auc_score`	`pROC::roc()` / `pROC::auc()`
Regularização (Lasso/Ridge)	`sklearn.linear_model.LogisticRegression(penalty=...)` — default no sklearn	`glmnet::glmnet(` `= "binomial")` — default, é pacote à

O ponto de maior atenção, mesmo padrão já visto no Módulo 4: `sklearn.LogisticRegression()` aplica regularização L2 **por padrão** (otimizado para predição), o que significa que os coeficientes não batem exatamente com `statsmodels.Logit()` ou com `glm()` do R a menos que você desative a regularização explicitamente (`penalty=None` no sklearn recente). Para fins de inferência (interpretar coeficientes, odds ratio, p-valor), `glm()` do R e `statsmodels.Logit()` são comparáveis; `sklearn` por padrão, não.

---

9. Exercícios

Exercício 1 — Confirmando o problema da regressão linear em binária

Ajuste `lm(alta_performance ~ fluxo_clientes_mensal, data = varejo)` e `glm(alta_performance ~ fluxo_clientes_mensal, data = varejo, family = binomial)`. Plote os valores ajustados dos dois modelos contra `fluxo_clientes_mensal` no mesmo gráfico. Em que faixa de `fluxo_clientes_mensal` o modelo linear produz previsões fora de $[0, 1]$?

Solução:

```
mod_lin <- lm(alta_performance ~ fluxo_clientes_mensal, data = varejo)
mod_log <- glm(alta_performance ~ fluxo_clientes_mensal, data = varejo, fa
mily = binomial)

varejo$pred_lin <- fitted(mod_lin)
varejo$pred_log <- fitted(mod_log)

ggplot(varejo, aes(x = fluxo_clientes_mensal)) +
  geom_line(aes(y = pred_lin, color = "Linear")) +
  geom_line(aes(y = pred_log, color = "Logístico")) +
  geom_hline(yintercept = c(0, 1), linetype = "dashed", color = "gray50")
+
  labs(y = "P(alta_performance = 1) prevista", color = "Modelo")

range(varejo$pred_lin)
```

O modelo linear produz valores previstos acima de 1 para `fluxo_clientes_mensal` alto (e pode produzir valores negativos no outro extremo, dependendo do range observado) — visualmente confirmado pelas linhas cruzando as guias tracejadas em 0 e 1. O modelo logístico permanece dentro de $[0, 1]$ em toda a extensão do eixo x, por construção da função sigmoide.

Exercício 2 — Interpretação de odds ratio

A partir de `modelo_logit` (seção 4), reporte o odds ratio de `porte = "Grande"` e interprete em uma frase completa, sem usar a palavra "probabilidade". Depois, calcule a probabilidade prevista para uma loja "Pequena" e uma "Grande" hipotéticas, mantendo `fluxo_clientes_mensal` e `ticket_medio` na mediana, e confirme que a diferença de probabilidade **não** é igual ao odds ratio.

Solução:

```
exp(coef(modelo_logit))["porteGrande"]

mediana_fluxo <- median(varejo$fluxo_clientes_mensal)
mediana_ticket <- median(varejo$ticket_medio)

novo <- data.frame(
  fluxo_clientes_mensal = mediana_fluxo,
  ticket_medio = mediana_ticket,
  porte = factor(c("Pequena", "Grande"), levels = levels(varejo$porte))
)

predict(modelo_logit, newdata = novo, type = "response")
```

Interpretação correta: "mantendo fluxo de clientes e ticket médio constantes, a **chance** (odds) de uma loja Grande ser classificada como alta performance é `OR` vezes a chance de uma loja Pequena." A diferença de probabilidade prevista entre as duas linhas do `predict()` não é igual ao odds ratio — confirma na prática o alerta da seção 5: odds ratio e razão de probabilidades são conceitos diferentes, e só coincidem aproximadamente quando *p* está próximo de 0 para ambos os grupos.

Exercício 3 — Matriz de confusão e o problema do corte fixo

Gere a matriz de confusão de `modelo_logit` com corte em 0.5. Repita com corte em 0.3. Compare sensibilidade e especificidade nos dois casos. O que acontece com cada métrica ao baixar o corte, e por quê?

Solução:

```
prob <- predict(modelo_logit, type = "response")

for (corte in c(0.5, 0.3)) {
  pred <- ifelse(prob > corte, 1, 0)
  m <- table(Observado = varejo$alta_performance, Previsto = pred)
  sens <- m["1", "1"] / sum(m["1", ])
  espec <- m["0", "0"] / sum(m["0", ])
  cat("Corte:", corte, "| Sensibilidade:", round(sens, 3), "|
Especificidade:", round(espec, 3), "\n")
}
```

Baixar o corte de 0.5 para 0.3 classifica mais observações como “1” — isso captura mais verdadeiros positivos (sensibilidade sobe), mas também gera mais falsos positivos (especificidade cai). É o trade-off estrutural entre as duas métricas: não existe corte que maximize as duas simultaneamente (exceto no caso irreal de separação perfeita entre classes) — a escolha do corte é uma decisão de negócio sobre qual erro custa mais, não uma pergunta puramente estatística.

Exercício 4 — Curva ROC e comparação de modelos

Ajuste dois modelos: `modelo_simples <- glm(alta_performance ~ fluxo_clientes_mensal, family = binomial)` e o `modelo_logit` completo (seção 4). Compare a AUC dos dois. O modelo com mais preditores tem AUC maior? Isso é surpreendente, dado o que o Módulo 4 (seção 4.3) ensinou sobre R^2 e número de preditores?

Solução:

```
library(pROC)

modelo_simples <- glm(alta_performance ~ fluxo_clientes_mensal, data =
varejo, family = binomial)

roc_simples <- roc(varejo$alta_performance, predict(modelo_simples, type
= "response"))
roc_completo <- roc(varejo$alta_performance, predict(modelo_logit, type =
"response"))

auc(roc_simples)
auc(roc_completo)
```

O modelo completo deve ter AUC maior ou igual ao simples — mesma lógica do R^2 na regressão linear (Módulo 4, seção 4.3): adicionar preditores nunca piora o ajuste **dentro da amostra de treino**, mesmo que o ganho seja pequeno ou o preditor adicional seja irrelevante. Não é surpreendente, é a mesma limitação já discutida: comparar modelos por métrica de ajuste dentro da amostra, sem validação fora da amostra, sempre favorece o modelo mais complexo — o motivo pelo qual comparação justa de modelos exige dados de validação/teste, fora do escopo deste plano introdutório.

Exercício 5 — Separação quase perfeita (diagnóstico de problema de convergência)

Ajuste `glm(alta_performance ~ logit_construido, data = varejo, family = binomial)` — usando a variável de log-odds que gerou a resposta, guardada na seção 3. Observe os avisos do R. Por que isso é um caso extremo (mas didático) de "separação quase perfeita", e por que isso nunca deveria acontecer com dado real (é sinal de vazamento de informação, não de modelo excelente)?

Solução:

```
modelo_vazado <- glm(alta_performance ~ logit_construido, data = varejo, f
amily = binomial)
summary(modelo_vazado)
```

O R deve emitir aviso de `"fitted probabilities numerically 0 or 1 occurred"` — porque `logit_construido` **é**, por construção, quase exatamente o log-odds verdadeiro que gerou `alta_performance` (a única diferença é o ruído do sorteio `rbinom()`). O modelo separa as classes quase perfeitamente, e os erros padrão dos coeficientes ficam artificialmente grandes/instáveis. Na prática, um resultado assim com dado real é motivo de suspeita, não comemoração — geralmente indica **vazamento de informação (data leakage)**: uma variável no conjunto de preditores que, direta ou indiretamente, já contém a resposta (por exemplo, usar `cancelou_pos_evento` como preditor de `cancelou`) — nunca um modelo genuinamente bom.

Exercício 6 (desafio) — Efeito de desbalanceamento de classe

Verifique a proporção de `alta_performance == 1` no dataset completo. Crie uma subamostra artificialmente desbalanceada (ex.: mantenha todos os "1"s, mas descarte aleatoriamente 90% dos "0"s) e reajuste o modelo completo nessa subamostra. Compare a acurácia e a AUC do modelo original com o do modelo na subamostra desbalanceada. A acurácia sozinha teria te enganado sobre a qualidade real do modelo?

Solução:

```
prop.table(table(varejo$alta_performance))

set.seed(2026)
idx_1 <- which(varejo$alta_performance == 1)
idx_0 <- which(varejo$alta_performance == 0)
idx_0_reduzido <- sample(idx_0, size = round(0.1 * length(idx_0)))

varejo_desbalanceado <- varejo[c(idx_1, idx_0_reduzido), ]
prop.table(table(varejo_desbalanceado$alta_performance))

modelo_desb <- glm(alta_performance ~ fluxo_clientes_mensal +
ticket_medio + porte,
                   data = varejo_desbalanceado, family = binomial)

prob_desb <- predict(modelo_desb, type = "response")
pred_desb <- ifelse(prob_desb > 0.5, 1, 0)
mean(pred_desb == varejo_desbalanceado$alta_performance)   # acurácia

library(pROC)
auc(roc(varejo_desbalanceado$alta_performance, prob_desb))
```

Na subamostra desbalanceada (agora com muito mais "1"s proporcionalmente), a acurácia tende a subir mecanicamente — fica mais fácil acertar quando uma classe domina —, mas a AUC (que não depende da proporção de classes, só da capacidade de ordenar corretamente positivos acima de negativos) deve permanecer relativamente estável. Isso demonstra por que acurácia é uma métrica frágil sob desbalanceamento e AUC (ou sensibilidade/especificidade reportadas separadamente) é mais informativa nesse cenário — o mesmo alerta da seção 6, agora quantificado experimentalmente.

10. Entrega esperada do módulo

Script `regressao_logistica_varejo.R` que recebe `varejo_sintetico.csv` e produz automaticamente:

1. Construção da variável `alta_performance` (seção 3) com `set.seed(2026)` fixo para reprodutibilidade
2. Modelo `glm()` ajustado com pelo menos 3 preditores, tabela de coeficientes e odds ratio com IC
3. Matriz de confusão, acurácia, sensibilidade, especificidade e precisão no corte de 0.5
4. Curva ROC plotada com AUC reportada
5. Comparação de pelo menos dois cortes de classificação diferentes, com a métrica que mudou mais destacada

---

11. Critério de domínio antes de avançar

- ☐ Explica de cabeça por que regressão linear não serve para resposta binária (os 3 motivos da seção 1, não só “porque sim”)
- ☐ Sabe a diferença entre log-odds, odds e probabilidade, e converte entre os três sem confundir
- ☐ Interpreta um odds ratio corretamente em uma frase, sem tratá-lo como razão de probabilidade
- ☐ Constrói e interpreta uma matriz de confusão, e sabe quando acurácia sozinha é enganosa
- ☐ Entende o trade-off entre sensibilidade e especificidade ao mudar o corte de classificação

- ☐ Sabe ler uma curva ROC e explicar o que AUC = 0.5 vs. AUC = 0.9 significam
- ☐ Reconhece separação quase perfeita como sinal de alerta (vazamento de informação), não de sucesso do modelo
- ☐ Todos os 6 exercícios rodam sem erro e a interpretação escrita usa a linguagem certa (odds, não probabilidade, ao descrever coeficientes)

MÓDULO 06 · MODELAGEM

Testes Não-Paramétricos

1. A lógica dos testes não-paramétricos

1.1 Por que existem

Testes paramétricos (teste t, ANOVA, Pearson) fazem inferência assumindo uma **forma específica** para a distribuição dos dados — normalmente a normal. Sob essa suposição, é possível derivar analiticamente a distribuição exata da estatística de teste, o que torna o teste poderoso (mais capaz de detectar um efeito real quando ele existe) e computacionalmente simples.

O problema: quando a distribuição real dos dados se afasta muito da suposição (forte assimetria, outliers extremos, distribuição multimodal), a validade do teste paramétrico degrada — o p-valor reportado deixa de refletir corretamente a probabilidade que ele diz medir. Testes não-paramétricos (mais precisamente, "testes de distribuição livre" — *distribution-free*) não fazem suposição sobre a forma da distribuição. Em vez de operar sobre os valores brutos, eles operam sobre a **ordem** (postos/ranks) dos dados.

1.2 Como funcionam, na essência: postos (ranks)

O procedimento comum a quase todo teste deste módulo:

1. Combine os dados (de um ou mais grupos, conforme o teste).
2. Substitua cada valor pelo seu **posto** (rank) — 1 para o menor valor, 2 para o segundo menor, e assim por diante. Empates recebem o posto médio.
3. Calcule a estatística de teste a partir dos postos, não dos valores originais.

```
x <- c(12, 15, 11, 40, 13)
rank(x)   # 3 4 2 5 1 -- observe que o "40" (outlier) vira só "o maior
posto", posto 5,
          # não importa se ele fosse 40 ou 4000
```

Essa é a fonte da robustez: um outlier extremo ainda ocupa só **uma posição** no ranking, não importa quão longe ele esteja dos demais valores. Um teste baseado em médias (como o teste t) é diretamente puxado pela magnitude do outlier; um teste baseado em postos, não.

1.3 O que se perde e o que se ganha

- **Ganho: robustez.** Válido sob premissas muito mais fracas — não exige normalidade, é pouco sensível a outliers, funciona em variáveis apenas ordinais (onde nem faz sentido calcular média).
- **Perda: poder estatístico, quando a normalidade de fato vale.** Se os dados realmente são normais, o teste paramétrico correspondente é mais eficiente — precisa de menos observações para detectar o mesmo efeito, porque usa mais informação (a magnitude exata dos valores, não só a ordem). Rodar Mann-Whitney em dados perfeitamente normais não é "errado", mas desperdiça um pouco de poder estatístico que o teste t teria aproveitado.

Esse é o trade-off central: **não-paramétrico é a escolha mais segura quando você não confia na normalidade; paramétrico é mais poderoso quando a normalidade é razoável.** A seção 6 formaliza o critério de decisão.

---

2. Teste de Mann-Whitney (Wilcoxon rank-sum) — duas amostras independentes

2.1 Hipóteses

Equivalente não-paramétrico do teste t para duas amostras independentes. Tecnicamente, Mann-Whitney testa se a distribuição de um grupo tende a ter valores sistematicamente maiores (ou menores) que a do outro — não especificamente se as médias diferem (essa é uma diferença conceitual real, não só cosmética, em relação ao teste t):

- H_0: as duas populações têm a mesma distribuição (nenhuma tende a ter valores sistematicamente maiores).
- H_1: uma das populações tende a ter valores sistematicamente maiores que a outra.

2.2 Lógica da estatística de teste

Combine as duas amostras, calcule os postos de todos os valores juntos, depois some os postos de cada grupo separadamente. Se os dois grupos vêm da mesma distribuição, a soma de postos de cada grupo deve ser proporcional ao seu tamanho — nenhum grupo deveria "monopolizar" os postos altos ou baixos. A estatística U (ou W, dependendo da implementação) formaliza o quão longe essa soma está do esperado sob H_0.

2.3 No R

```
library(dplyr)

varejo <- read.csv("varejo_sintetico.csv", stringsAsFactors = TRUE)

# Subconjunto do teste de fidelidade: só quem participou tem grupo
definido
teste_fidelidade <- varejo %>% filter(participou_teste_fidelidade == 1)

wilcox.test(vendas_incrementais_fidelidade ~ grupo_teste_fidelidade,
            data = teste_fidelidade)
```

`wilcox.test()` com uma fórmula `y ~ grupo` (grupo com exatamente 2 níveis) roda Mann-Whitney automaticamente. O output traz a estatística *W* e o p-valor — a teoria formal de interpretação de p-valor fica para o Módulo 9, mas por ora: p-valor baixo indica que é improvável que as duas distribuições sejam idênticas só por acaso amostral.

2.4 Exemplo: por que Mann-Whitney é a escolha certa aqui

O script gerador (`gerar_datasets_sinteticos.R`) constrói os dois grupos deliberadamente diferentes em forma: `Controle` é aproximadamente normal (`rnorm` , média 0), `Tratamento` é assimétrico com cauda longa (`rgamma` , deslocado) — algumas lojas “estrela” com ganho muito acima da maioria. Verifique visualmente antes de decidir o teste:

```
ggplot(teste_fidelidade, aes(x = vendas_incrementais_fidelidade, fill =
grupo_teste_fidelidade)) +
  geom_density(alpha = 0.4) +
  labs(title = "Vendas incrementais por grupo — Controle (normal) vs.
Tratamento (cauda longa)",
       x = "Vendas incrementais (R$)", y = "Densidade")
```

O grupo `Tratamento` sendo assimétrico com outliers de alto ganho é exatamente o cenário em que o teste t (que compara médias, sensível à cauda) e o Mann-Whitney (que compara distribuições via postos, insensível à magnitude da cauda) podem divergir na conclusão. Rode os dois lado a

lado para comparar diretamente — a teoria completa do teste t vem no Módulo 9, mas nada impede de já rodar `t.test()` aqui para ver a diferença prática:

```
t.test(vendas_incrementais_fidelidade ~ grupo_teste_fidelidade, data =
teste_fidelidade)
```

---

3. Teste de Wilcoxon (signed-rank) — amostras pareadas

3.1 Hipóteses e lógica

Equivalente não-paramétrico do teste t pareado — usado quando as duas medidas vêm da **mesma unidade observacional** em dois momentos ou condições (aqui: a mesma loja, antes e depois de uma campanha). A lógica:

1. Calcule a diferença $d_i = y_{i,\text{depois}} - y_{i,\text{antes}}$ para cada par.
2. Ranqueie os **valores absolutos** dessas diferenças (ignorando o sinal por enquanto).
3. Devolva o sinal original a cada posto e some os postos positivos e negativos separadamente.

- H_0: a distribuição das diferenças é simétrica em torno de zero (não há deslocamento sistemático entre antes/depois).
- H_1: há deslocamento sistemático (para cima ou para baixo).

A diferença central para Mann-Whitney: aqui a unidade de análise é o **par** (a diferença dentro da mesma loja), não a comparação entre duas amostras independentes — por isso pareamento errado (comparar linhas que não correspondem à mesma loja) invalida o teste completamente.

3.2 No R

```
campanha <- varejo %>% filter(participou_campanha == 1)

wilcox.test(campanha$receita_pos_campanha, campanha$receita_pre_campanha,
paired = TRUE)
```

`paired = TRUE` é o argumento que muda o teste de Mann-Whitney (independente) para Wilcoxon signed-rank (pareado) — o erro mais comum aqui é esquecer esse argumento e rodar o teste errado silenciosamente (o R não avisa; ele roda Mann-Whitney nas duas colunas como se fossem grupos independentes, o que estatisticamente não faz sentido para dados pareados).

3.3 Verificação visual antes do teste

```
campanha_long <- campanha %>%
  select(loja_id, receita_pre_campanha, receita_pos_campanha) %>%
  tidyr::pivot_longer(cols = starts_with("receita_"), names_to =
"momento", values_to = "receita")

ggplot(campanha_long, aes(x = momento, y = receita)) +
  geom_boxplot() +
  geom_line(aes(group = loja_id), alpha = 0.05) +   # liga os pares --
visualiza a mudança individual
  labs(title = "Receita antes vs. depois da campanha", x = "", y = "Receit
a mensal (R$)")
```

A linha ligando cada par (`geom_line(aes(group = loja_id))`) é o que diferencia visualmente um gráfico pareado de um gráfico de grupos independentes — sem isso, dois boxplots lado a lado escondem a informação mais importante: a *trajetória individual* de cada loja, que é o que o teste pareado de fato usa.

---

4. Kruskal-Wallis — três ou mais amostras independentes

4.1 Lógica

Extensão de Mann-Whitney para $k \geq 3$ grupos — exatamente como ANOVA de um fator estende o teste t de 2 para k grupos. Mesma lógica de postos: combine todas as observações de todos os grupos, ranqueie globalmente, e verifique se a soma de postos de cada grupo é compatível com o que se esperaria se todos os grupos viessem da mesma distribuição.

$$ H = \frac{12}{n(n+1)} \sum_{j=1}^{k} \frac{R_j^2}{n_j} - 3(n+1) $$

Onde R_j é a soma de postos do grupo j, n_j seu tamanho, e n o total de observações. Sob H_0 (todos os grupos vêm da mesma distribuição), H segue aproximadamente uma distribuição qui-quadrado com $k - 1$ graus de liberdade.

- H_0: todos os k grupos vêm da mesma distribuição.
- H_1: pelo menos um grupo difere dos demais.

Ponto crítico, igual à ANOVA: um Kruskal-Wallis significativo diz "existe diferença em algum lugar entre os k grupos", não diz **qual** grupo difere de qual. Para isso, precisa de post-hoc (seção 4.3).

4.2 No R

```
kruskal.test(variavel_numerica ~ variavel_grupo, data = seu_data_frame)
```

4.3 Pós-hoc: quais grupos diferem entre si

Esta é a pergunta que fica em aberto depois de um Kruskal-Wallis significativo — e a resposta direta: **sim, existe post-hoc para o caso não-paramétrico**, assim como Tukey HSD existe para depois de uma ANOVA.

Opção 1 — Teste de Dunn (o mais usado, análogo direto do Tukey):

```
install.packages("dunn.test")
library(dunn.test)

dunn.test(variavel_numerica, variavel_grupo, method = "bonferroni")
```

O teste de Dunn compara cada par de grupos usando os mesmos postos globais calculados para o Kruskal-Wallis (não recalcula ranks separadamente por par — essa é uma diferença técnica importante em relação à opção 2 abaixo), e aplica correção para comparações múltiplas. `method` aceita `"bonferroni"`, `"holm"`, `"bh"` (Benjamini-Hochberg), entre outros — a mesma lógica de correção que retorna formalmente no Módulo 9 quando o tema é comparações múltiplas.

Alternativa via pacote `FSA`, com output em formato de data frame (mais fácil de filtrar programaticamente):

```
install.packages("FSA")
FSA::dunnTest(variavel_numerica ~ variavel_grupo, data = seu_data_frame, m
ethod = "bh")
```

Opção 2 — Mann-Whitney par a par com correção (mais simples de explicar, um pouco menos formal estatisticamente porque recalcula ranks a cada par em vez de usar o ranking global):

```
pairwise.wilcox.test(variavel_numerica, variavel_grupo, p.adjust.method =
"bonferroni")
```

Por que a correção de p-valor é obrigatória aqui: com k grupos, o número de pares cresce como $\binom{k}{2}$ — 4 grupos já são 6 comparações. Rodar 6 testes com $\alpha = 0.05$ cada, sem correção, infla a chance de achar pelo menos um "falso positivo" bem acima de 5% (o mesmo problema de comparações múltiplas mencionado no Módulo 4, seção 2, e formalizado no Módulo 9). Nunca rode `wilcox.test()` par a par manualmente sem `p.adjust()` ou uma função que já corrija automaticamente.

Regra prática de leitura: só faz sentido olhar o post-hoc **depois** de um Kruskal-Wallis global significativo — rodar post-hoc diretamente, sem o teste global, reintroduz o mesmo problema de múltiplas comparações que o teste global existe para controlar.

4.4 Exemplo com dado real: temperatura por estação do ano

Usando os dados climáticos do INMET (seção 7), compare a temperatura média diária entre as 4 estações do ano em uma mesma estação meteorológica — um caso natural de $k = 4$ grupos independentes, sem premissa de normalidade garantida:

```
clima <- clima %>%
  mutate(
    mes = as.integer(format(data, "%m")),
    estacao_ano = case_when(
      mes %in% c(12, 1, 2) ~ "Verao",
      mes %in% c(3, 4, 5)  ~ "Outono",
      mes %in% c(6, 7, 8)  ~ "Inverno",
      mes %in% c(9, 10, 11) ~ "Primavera"
    )
  )

kruskal.test(temp_media_diaria ~ estacao_ano, data = clima)

dunn.test::dunn.test(clima$temp_media_diaria, clima$estacao_ano, method =
"bonferroni")
```

Verão e inverno devem aparecer como o par mais distinguível no post-hoc (maior diferença de temperatura média); outono e primavera, por serem estações de transição, tendem a se sobrepor mais entre si e com os extremos — vale conferir se o resultado do seu recorte de dados (estação/ano específicos) confirma esse padrão climatologicamente esperado.

---

5. Correlação de Spearman

5.1 Lógica

Spearman é, literalmente, **a correlação de Pearson calculada sobre os postos em vez dos valores brutos**:

r_s = Pearson(rank(x), rank(y))

Quando não há empates, existe uma fórmula simplificada equivalente:

$$ r_s = 1 - \frac{6\sum d_i^2}{n(n^2-1)} $$

onde d_i é a diferença entre os postos de x_i e y_i no par i.

5.2 O que Spearman mede — e por que isso importa

Pearson mede força de relação **linear**. Spearman mede força de relação **monotônica** — a relação pode ser uma curva, desde que seja consistentemente crescente ou consistentemente decrescente (sem inverter direção). Isso resolve diretamente a limitação de Pearson apontada no Módulo 4: uma relação fortemente não-linear, mas monotônica (como `taxa_conversao` vs. `fluxo_clientes_mensal` no dataset sintético — cresce rápido e satura, mas nunca inverte direção), pode ter r de Pearson moderado mas r_s de Spearman próximo de 1.

```
cor(varejo$fluxo_clientes_mensal, varejo$taxa_conversao, method = "pearson
")
cor(varejo$fluxo_clientes_mensal, varejo$taxa_conversao, method = "spearma
n")
```

O Spearman deve sair visivelmente mais alto que o Pearson aqui — evidência numérica direta de que a relação é monotônica forte, mesmo não sendo linear (o mesmo padrão que o gráfico `loess` vs. `lm` do Módulo 1, Exercício 3, já mostrava visualmente).

Spearman também é a escolha certa quando pelo menos uma das variáveis é **ordinal** (categorias com ordem, mas sem distância numérica real entre elas — ex.: satisfação em escala Likert 1-5) — Pearson não é definido de forma significativa para esse tipo de variável, Spearman é.

5.3 No R

```
cor.test(varejo$fluxo_clientes_mensal, varejo$taxa_conversao, method = "sp
earman")
```

`cor.test(method = "spearman")` retorna a estatística S (relacionada à soma dos d_i^2) e o p-valor — de novo, teoria formal do p-valor no Módulo 9.

5.4 Exemplo com dado real: temperatura x umidade relativa

```
cor(clima$temperatura_ar, clima$umidade_relativa, method = "pearson")
cor(clima$temperatura_ar, clima$umidade_relativa, method = "spearman")

ggplot(clima, aes(x = temperatura_ar, y = umidade_relativa)) +
  geom_point(alpha = 0.1) +
  geom_smooth(method = "loess", color = "firebrick") +
  labs(title = "Temperatura vs. umidade relativa", x = "Temperatura do ar
(°C)", y = "Umidade relativa (%)")
```

Temperatura e umidade relativa costumam ter relação negativa e aproximadamente monotônica (ar mais quente tende a reter relativamente menos umidade em % na maior parte do range observado), mas raramente perfeitamente linear — bom caso real para comparar as duas métricas de correlação lado a lado.

---

6. Critério de decisão: paramétrico ou não-paramétrico

6.1 Fluxo de decisão prático

1. **Olhe a distribuição visualmente primeiro** (Módulo 1): histograma, boxplot, densidade por grupo. Assimetria forte ou outliers extremos visíveis já são sinal de alerta.
2. **Rode um teste formal de normalidade como apoio, não como veredito único** — `shapiro.test()` é o mais comum (teoria completa e limitações no Módulo 9; adiantando o essencial: com `n` grande, `shapiro.test()` tende a rejeitar normalidade mesmo para desvios pequenos e sem relevância prática, então nunca decida só pelo p-valor dele, sempre cruze com o histograma):

```
shapiro.test(teste_fidelidade$vendas_incrementais_fidelidade[teste_fidelid
ade$grupo_teste_fidelidade == "Tratamento"])
```

1. **Considere o tamanho da amostra.** Com `n` grande (dezenas ou centenas de observações por grupo), o Teorema do Limite Central (Módulo 8) faz a distribuição amostral da média se aproximar da normal mesmo que os dados originais não sejam normais — o que torna o teste t mais robusto a desvios de normalidade do que se poderia pensar à primeira vista. Isso não é uma licença geral para ignorar assimetria (outliers extremos ainda distorcem a média mesmo em amostras grandes), mas reduz a urgência

da escolha não-paramétrica quando `n` é confortavelmente grande e a distribuição não é patologicamente distorcida.

2. **Quando em dúvida, ou quando a decisão é sensível, rode os dois** (paramétrico e não-paramétrico) e compare. Se as conclusões batem, a escolha do teste não estava sendo o fator decisivo. Se divergem — como no Exercício 1 deste módulo — isso é, por si, uma informação importante: geralmente sinaliza que a média está sendo puxada por uma cauda ou outlier que a mediana (e os testes baseados em posto) ignoram.

6.2 Tabela de correspondência (referência rápida)

Cenário	Paramétrico	Não-paramétrico
2 amostras independentes	Teste t	Mann-Whitney (`wilcox.test`)
2 amostras pareadas	Teste t pareado	Wilcoxon signed-rank (`wilcox.test(paired=TRUE)`)
3+ amostras independentes	ANOVA (`aov`)	Kruskal-Wallis (`kruskal.test`)
Post-hoc após teste global significativo	Tukey HSD	Dunn (`dunn.test`) ou pairwise Wilcoxon com correção
Correlação entre 2 variáveis contínuas	Pearson (`cor`)	Spearman (`cor` , `method="spearman"`)

7. Fonte de dados reais para este módulo: INMET

Este é o primeiro módulo do projeto a usar dado externo real, conforme decidido — o dataset sintético continua servindo como apoio (seções 2 e 3), mas Kruskal-Wallis e Spearman (seções 4 e 5) usam clima real, porque precipitação e temperatura são não-normais por fenômeno físico, não por construção didática.

7.1 Fonte primária — Portal INMET (dados históricos, estações automáticas)

Link: https://portal.inmet.gov.br/dadoshistoricos

Passo a passo:

1. Acesse o link acima. A página lista os anos disponíveis.
2. Baixe o arquivo `.zip` do ano de interesse — cada ZIP contém um `.CSV` por estação meteorológica automática em operação naquele ano, no Brasil inteiro.
3. Descompacte e localize o CSV da estação de interesse pelo nome do arquivo (o padrão de nomenclatura inclui região, UF, código da estação e nome da cidade, ex.: `INMET_S_SC_A806_[NOME_DA_CIDADE]_[DATA_INICIAL]_A_[DATA_FINAL].CSV`).
4. Alternativamente, use "Histórico de Dados Meteorológicos" no menu do portal, que permite localizar a estação num mapa interativo antes de baixar — mais direto se você já sabe a cidade/estado de interesse, mas não quer abrir o ZIP do Brasil inteiro.

Estrutura do arquivo (válida para a maioria dos anos, mas confira sempre — ver aviso abaixo):

- As **8 primeiras linhas** trazem metadados da estação (região, UF, nome, código WMO, latitude, longitude, altitude, data de fundação) — não são dados de medição, pule-as na leitura (`skip = 8`).

- A partir da linha 9: dados **horários**, aproximadamente 19 colunas, incluindo data, hora (UTC), precipitação horária (mm), pressão atmosférica, radiação global, temperatura do ar (bulbo seco), temperatura do ponto de orvalho, umidade relativa, direção e velocidade do vento.
- Valores ausentes são codificados como `-9999`, não como célula vazia — trate explicitamente na leitura.
- Separador `;` (ponto e vírgula) e decimal `,` (vírgula) — o padrão brasileiro, por isso `read.csv2()` (não `read.csv()`) é o ponto de partida certo no R.

```
clima_horario <- read.csv2(
  "INMET_S_SC_A806_NOME_DA_CIDADE_01-01-2023_A_31-12-2023.CSV",
  skip = 8,
  na.strings = "-9999",
  fileEncoding = "Latin1"   # ajuste para "UTF-8" se acentos aparecerem
corrompidos
)

names(clima_horario)   # SEMPRE confira os nomes reais antes de seguir
```

Aviso importante, e ele é o ponto pedagógico deste módulo: os nomes de coluna do INMET **não são totalmente consistentes entre estações e entre anos** — o mesmo tipo de dado real que o dataset sintético não pode simular. Não copie nomes de coluna de exemplo algum (inclusive deste documento) sem antes rodar `names(clima_horario)` e confirmar contra o arquivo que você efetivamente baixou. Depois de conferir, renomeie para algo manejável:

```
library(dplyr)

clima <- clima_horario %>%
  rename(
    data = 1,                 # ajuste os índices/nomes conforme o seu
arquivo
    hora_utc = 2,
    temperatura_ar = matches("BULBO SECO"),
    umidade_relativa = matches("UMIDADE RELATIVA"),
    precipitacao = matches("PRECIPITA")
  ) %>%
  mutate(data = as.Date(data))
```

Agregação diária (necessária para os exemplos das seções 4 e 5, que comparam por dia/estação do ano, não por hora):

```
clima <- clima %>%
  group_by(data) %>%
  summarise(
    temp_media_diaria = mean(temperatura_ar, na.rm = TRUE),
    temperatura_ar = mean(temperatura_ar, na.rm = TRUE),
    umidade_relativa = mean(umidade_relativa, na.rm = TRUE),
    precipitacao_total_dia = sum(precipitacao, na.rm = TRUE),
    .groups = "drop"
  )
```

7.2 Fontes alternativas

- **Base dos Dados** — versão já tratada e padronizada do mesmo BDMEP do INMET, consultável via SQL/BigQuery sem lidar com ZIP/CSV bruto: https://basedosdados.org/dataset/782c5607-9f69-4e12-b0d5-aa0f1a7a94e2. Requer conta Google Cloud (nível gratuito cobre uso educacional tranquilamente) e o pacote `basedosdados` (disponível para Python; para R, consulte a documentação do projeto para o cliente equivalente ou exporte via console web direto para CSV).
- **Kaggle (espelho já tratado)** — "Brazil Weather Information by INMET": https://www.kaggle.com/datasets/gregoryoliveira/brazil-weather-information-by-inmet. Mais rápido para começar (um único CSV, sem lidar com ZIP por estação), mas é um espelho de terceiros — para o exercício deste módulo, prefira a fonte oficial quando possível; use o

Kaggle como atalho se a fricção do portal oficial for um obstáculo real para você seguir em frente.

- **BDMEP (bdmep.inmet.gov.br)** — interface alternativa do próprio INMET, com dados diários já agregados por estação convencional (menos linhas que o horário automático). Exige cadastro gratuito no site antes de baixar — mencionado aqui como opção, mas não é o caminho usado nos exemplos deste módulo por causa dessa barreira extra.

---

8. Diferenças-chave vindas de Python

Conceito	Python (scipy.stats)	R
Mann-Whitney	`scipy.stats.mannwhitneyu(x, y)`	`wilcox.test(x, y)` ou `wilcox.test(y ~ grupo`
Wilcoxon pareado	`scipy.stats.wilcoxon(x, y)`	`wilcox.test(x, y, paired = TRUE)`
Kruskal-Wallis	`scipy.stats.kruskal(g1, g2, g3, ...)` — grupos como argumentos separados	`kruskal.test(y ~ grup data = df)` — sintaxe de fórmula, mais direto com data frame
Post-hoc de Dunn	`scikit-posthocs.posthoc_dunn(df, val_col=, group_col=, p_adjust=)`	`dunn.test::dunn.test(` ou `FSA::dunnTest()`
Correlação de Spearman	`scipy.stats.spearmanr(x, y)`	`cor(x, y, method = "spearman")` + `cor.test(..., method "spearman")`
Teste de normalidade	`scipy.stats.shapiro(x)`	`shapiro.test(x)`

Conceito	Python (scipy.stats)	R
Correção de p-valor múltiplo	`statsmodels.stats.multitest.multipletests(pvals, method="bonferroni")`	`p.adjust(pvals, metho = "bonferroni")` — usado internamente por `dunn.test` / `pairwise.wilcox.test`

O padrão de novo se repete: `scipy.stats` entrega as funções de teste isoladas (cada uma recebendo vetores brutos), enquanto o R aproveita a sintaxe de fórmula (`y ~ grupo`) de forma consistente entre `wilcox.test` , `kruskal.test` , `lm` , `aov` — uma vez que você aprendeu a fórmula no Módulo 4, ela se reaproveita praticamente sem mudança de sintaxe em todo teste deste módulo.

---

9. Exercícios

Exercício 1 — Mann-Whitney vs. teste t: quando divergem

Usando `varejo_sintetico.csv` , filtre `participou_teste_fidelidade == 1` . Rode `wilcox.test()` e `t.test()` para `vendas_incrementais_fidelidade` por `grupo_teste_fidelidade` . As conclusões (rejeitar/não rejeitar a hipótese de igualdade) batem? Se divergirem, qual teste você confiaria mais, dado o que você sabe da construção do grupo `Tratamento` (seção 2.4)?

Solução:

```
teste_fidelidade <- varejo %>% filter(participou_teste_fidelidade == 1)

wilcox.test(vendas_incrementais_fidelidade ~ grupo_teste_fidelidade, data
= teste_fidelidade)
t.test(vendas_incrementais_fidelidade ~ grupo_teste_fidelidade, data =
teste_fidelidade)
```

Ambos tendem a apontar diferença significativa aqui, porque o efeito construído (`rgamma` deslocado vs. `rnorm` em torno de zero) é forte o bastante para aparecer nos dois testes — mas vale conferir se a magnitude do p-valor e a robustez da conclusão diferem. O ponto conceitual, independente do resultado exato: como o grupo `Tratamento` tem cauda longa (poucas lojas "estrela" puxando a média para cima), o teste t está sujeito à influência dessas poucas observações extremas sobre a média comparada; o Mann-Whitney, baseado em postos, é a leitura mais confiável da tendência central real da diferença entre os grupos.

Exercício 2 — Wilcoxon pareado e verificação de pareamento

Usando o subconjunto `participou_campanha == 1`, rode o teste de Wilcoxon pareado entre `receita_pre_campanha` e `receita_pos_campanha`. Antes de interpretar o resultado, confirme que não há `NA` residual atrapalhando o pareamento (`sum(is.na(...))` nas duas colunas do subconjunto). Interprete o resultado.

Solução:

```
campanha <- varejo %>% filter(participou_campanha == 1)

sum(is.na(campanha$receita_pre_campanha))
sum(is.na(campanha$receita_pos_campanha))

wilcox.test(campanha$receita_pos_campanha, campanha$receita_pre_campanha,
paired = TRUE)
```

Como o script gerador constrói `efeito_campanha` com média positiva (0.06) mas desvio padrão relativamente alto (0.05) — ou seja, a maioria das lojas melhora, mas algumas pioram —, espera-se rejeitar a hipótese de que a diferença é simetricamente distribuída em torno de zero, com direção positiva (receita pós-campanha tende a ser maior). Vale complementar com `median(campanha$receita_pos_campanha - campanha$receita_pre_campanha)` para quantificar a mudança típica em R$.

Exercício 3 — Kruskal-Wallis com dado real e post-hoc de Dunn

Usando os dados do INMET (seção 7) para uma estação de sua escolha, compare `temp_media_diaria` entre as 4 estações do ano (ver `case_when` da seção 4.4). Rode `kruskal.test()`. Se significativo, rode `dunn.test::dunn.test()` com correção de Bonferroni. Quais pares de estações do ano diferem entre si? Isso é climatologicamente esperado para a sua estação escolhida?

Solução: ver código completo na seção 4.4. Interpretação: o par verão-inverno deve mostrar a diferença mais consistente e significativa (maior contraste de temperatura); outono e primavera, sendo estações de transição, tendem a se sobrepor mais tanto entre si quanto com os extremos adjacentes (outono próximo do inverno, primavera próxima do verão) — confirme isso olhando o `dunn.test()` par a par, não só o resultado global do Kruskal-Wallis.

Exercício 4 — Pearson vs. Spearman em relação não-linear (dataset sintético)

Usando `varejo_sintetico.csv`, calcule Pearson e Spearman entre `fluxo_clientes_mensal` e `taxa_conversao`. Compare os dois valores. Plote o scatter com `geom_smooth(method = "loess")`. A diferença entre Pearson e Spearman é consistente com o formato da curva observada?

Solução: ver código da seção 5.2. Espera-se Spearman visivelmente maior que Pearson, porque a relação é monotônica crescente (mais fluxo sempre associa a taxa de conversão igual ou maior, nunca inverte) mas não-linear (satura, não é reta) — exatamente o cenário em que Spearman captura mais força de associação do que Pearson.

Exercício 5 — Spearman em dado real: temperatura x umidade

Usando os dados do INMET, calcule Pearson e Spearman entre `temperatura_ar` e `umidade_relativa` (nível horário ou diário, sua escolha — documente qual usou). Visualize com `loess`. A relação parece mais próxima de linear ou de monotônica-não-linear?

Solução: ver seção 5.4. Espera-se correlação negativa nas duas métricas; a distância entre Pearson e Spearman aqui tende a ser menor que no Exercício 4 (a relação temperatura-umidade, apesar de não ser perfeitamente linear, também não é tão fortemente saturante quanto `taxa_conversao` vs. `fluxo`) — vale registrar a magnitude exata encontrada nos seus dados particulares (estação e período variam).

Exercício 6 (desafio) — Construindo o critério de decisão na prática

Escolha uma variável numérica qualquer do dataset INMET baixado (ex.: `precipitacao_total_dia`). Rode `shapiro.test()` numa amostra de até 5000 observações (limite do teste no R) e construa o histograma correspondente. A decisão indicada pelo Shapiro-Wilk bate com a leitura visual do histograma? Se você tivesse que comparar `precipitacao_total_dia` entre duas estações do ano, qual teste você escolheria e por quê?

Solução esperada (conceitual, sem gabarito numérico fixo por depender do seu recorte de dados): `precipitacao_total_dia` tende a ser fortemente assimétrica à direita — muitos dias com pouca ou nenhuma chuva, poucos dias com volume muito alto — um caso clássico onde tanto o Shapiro-Wilk (rejeitando normalidade) quanto o histograma (visualmente assimétrico, possivelmente com uma "pilha" de zeros) convergem para a mesma conclusão: teste não-paramétrico (Mann-Whitney para 2 estações, Kruskal-Wallis para as 4) é a escolha mais defensável aqui, exatamente o tipo de variável que motivou este módulo inteiro.

10. Entrega esperada do módulo

Script `nao_parametricos_varejo_clima.R` que:

1. Roda Mann-Whitney (`vendas_incrementais_fidelidade` por grupo) e Wilcoxon pareado (`receita_pre/pos_campanha`) sobre `varejo_sintetico.csv` , com gráfico de verificação visual para cada um (densidade por grupo; boxplot pareado com linhas de trajetória)
2. Lê, limpa e agrega (para nível diário) um arquivo de estação do INMET baixado conforme a seção 7
3. Roda Kruskal-Wallis de temperatura por estação do ano + post-hoc de Dunn, com tabela de quais pares diferem
4. Calcula e compara Pearson vs. Spearman para `fluxo_clientes_mensal` × `taxa_conversao` (sintético) e `temperatura_ar` × `umidade_relativa` (INMET)

11. Critério de domínio antes de avançar ao Módulo 7

- ☐ Explica de cabeça por que testes baseados em postos são robustos a outliers, usando a lógica de “um outlier ocupa só uma posição no ranking”
- ☐ Sabe escolher corretamente entre Mann-Whitney e Wilcoxon signed-rank, dado se os dados são independentes ou pareados
- ☐ Entende por que um Kruskal-Wallis significativo não identifica sozinho quais grupos diferem, e sabe nomear pelo menos uma opção de post-hoc (Dunn) e por que a correção de p-valor é obrigatória nele
- ☐ Consegue explicar a diferença conceitual entre Pearson (relação linear) e Spearman (relação monotônica) com um exemplo próprio
- ☐

Sabe montar um critério de decisão prático entre via paramétrica e não-paramétrica, sem depender cegamente de um único teste de normalidade

- ☐ Conseguiu baixar, limpar (skip de metadados, tratamento de `-9999`, conferência de nomes de coluna) e agregar pelo menos um arquivo real do INMET sem erro
- ☐ Todos os 6 exercícios rodam sem erro, e a interpretação escrita conecta o resultado do teste com a forma da distribuição observada (não só "o p-valor deu baixo")

---

12. Nota: decisões tomadas e pontos em aberto

1. **Primeira incorporação de dataset real no projeto**, conforme decidido — usei a fonte oficial do INMET (portal.inmet.gov.br/dadoshistoricos) como principal, com Base dos Dados e Kaggle como alternativas documentadas na seção 7. A partir daqui, cada módulo vai decidir dataset sintético vs. real conforme o que for mais adequado ao tema, com a mesma seção de "como obter" sempre que um dado externo for usado.

2. `shapiro.test()` e `p.adjust()` **foram adiantados operacionalmente** (seções 6.1 e 4.3) sem teoria formal de teste de hipótese — mesma lógica já aplicada a `Pr(>|t|)` no Módulo 4. Teoria completa (H0/H1, erro Tipo I/II, correção de comparações múltiplas formalizada) chega no Módulo 9.

3. **Assimetria/curtose numéricas**, pendência sinalizada desde o Módulo 2, continuam em aberto. Como este módulo já trata assimetria como motivador central (é literalmente o "porquê" dos testes não-paramétricos), pode ser um bom ponto de retomar a decisão: incluir como adendo rápido aqui, ou manter só qualitativo até o fim do plano?

4. **O dataset do INMET não é fixado a uma estação específica de propósito** — cada leitor escolhe a estação (idealmente uma perto de sua região de interesse). Isso é diferente do padrão dos Módulos 1, 2 e 4, onde todo mundo trabalha exatamente com os mesmos números. Trade-off consciente: ganha-se relevância e realismo (dado que você mesmo escolheu), perde-se comparabilidade exata de resultado entre diferentes execuções do exercício — por isso as soluções deste módulo descrevem o **padrão esperado**, não um número fixo, quando o exercício usa INMET.

MÓDULO 07 · MODELAGEM

Cluster e PCA

Estatística Multivariada

1. Pré-requisito indispensável: padronização (z-score)

Antes de qualquer distância ou covariância entre variáveis multivariadas, uma pergunta precisa ser resolvida: **as variáveis estão na mesma escala?** Se não, tanto clustering (baseado em distância) quanto PCA (baseado em variância/covariância) vão ser dominados pela variável de maior escala numérica, não pela variável de maior importância real.

No dataset deste módulo (seção 7), `Proline` varia na casa de centenas/milhares, enquanto `Hue` varia entre ~0.5 e 1.7 — em distância euclidiana bruta, `Proline` sozinho decide quase toda a distância entre dois vinhos, mesmo que `Hue` seja, quimicamente, tão informativo quanto.

A solução: padronizar cada variável para média 0 e desvio padrão 1 (z-score, já visto informalmente no Módulo 2 via CV — aqui a transformação completa, não só a razão):

$$ z = \frac{x - \bar{x}}{s} $$

```
vinho <- read.csv("wine.csv")   # ver seção 7 para como obter e nomear as
colunas

vinho_num <- vinho[, -1]           # remove a coluna "classe" (não entra
na padronização)
vinho_padronizado <- scale(vinho_num)   # scale() já centraliza e divide
pelo desvio padrão

apply(vinho_padronizado, 2, mean)   # deve dar ~0 em todas as colunas
apply(vinho_padronizado, 2, sd)     # deve dar exatamente 1 em todas as
colunas
```

Esta seção parece pequena, mas ignorá-la é o erro mais comum e mais silencioso em análise multivariada — o R não avisa se você esqueceu de padronizar, ele só entrega um resultado dominado por uma variável sem que isso apareça em nenhuma mensagem de erro.

---

2. Análise de cluster hierárquico

2.1 A ideia

Agrupar observações de forma que itens dentro do mesmo grupo sejam parecidos entre si, e itens de grupos diferentes sejam diferentes — sem que o número de grupos, nem quais observações pertencem a qual grupo, seja definido de antemão. “Parecido” é definido por uma **métrica de distância** no espaço das variáveis (padronizadas).

2.2 Métricas de distância

A mais comum é a distância **Euclidiana** — a generalização direta do teorema de Pitágoras para *p* dimensões:

$$ d(x, y) = \sqrt{\sum_{j=1}^{p}(x_j - y_j)^2} $$

Alternativas: **Manhattan** ($\sum|x_j - y_j|$, soma de diferenças absolutas — menos sensível a um único eixo muito discrepante que a Euclidiana) e **Gower** (para misturar variáveis numéricas e categóricas no mesmo cálculo de distância — fora do escopo deste dataset, que é todo numérico, mas relevante para dados reais em geral).

```
distancias <- dist(vinho_padronizado, method = "euclidean")
```

`dist()` retorna a matriz de distâncias par a par entre todas as 178 × 177/2 combinações de vinhos — a matéria-prima para qualquer método de agrupamento hierárquico.

2.3 Métodos de ligação (linkage)

Uma vez com as distâncias par a par, o algoritmo hierárquico aglomerativo começa com cada observação como seu próprio cluster e vai fundindo os dois clusters mais "próximos" repetidamente, até sobrar um só. A pergunta que o método de ligação responde: **quando dois clusters (não dois pontos) são "próximos"?**

- **Single linkage** (ligação simples): distância entre os dois pontos mais próximos dos dois clusters. Tende a formar clusters alongados/encadeados ("chaining") — sensível a um único par de pontos próximos, mesmo que os clusters como um todo sejam bem diferentes.
- **Complete linkage** (ligação completa): distância entre os dois pontos mais distantes dos dois clusters. Tende a formar clusters compactos e de tamanho parecido.
- **Average linkage** (ligação média): média de todas as distâncias par a par entre os dois clusters. Meio-termo entre single e complete.
- **Ward (`ward.D2`)**: não usa distância entre pontos diretamente — funde os dois clusters cuja fusão **minimiza o aumento na variância interna total** (soma de quadrados dentro dos clusters). Na prática, é o método mais usado por padrão quando não há razão específica para outro — produz

clusters compactos e de tamanho relativamente equilibrado, e tende a ser o mais robusto para dados reais.

```
cluster_hier <- hclust(distancias, method = "ward.D2")
```

2.4 Dendrograma — leitura e corte

```
plot(cluster_hier, labels = FALSE, main = "Dendrograma — Wine dataset
(Ward)",
     xlab = "Observações", ylab = "Distância de fusão")
```

O dendrograma é uma árvore: cada fusão é representada por um "U" cuja **altura** é a distância na qual os dois clusters foram unidos. Ler um dendrograma:

- Quanto mais baixo o ponto de fusão, mais parecidos eram os dois grupos fundidos ali.
- Um "salto" grande na altura entre uma fusão e a próxima é o sinal visual de que, abaixo desse salto, você tinha grupos genuinamente separados — e acima dele, já está forçando a união de coisas diferentes.

Cortando a árvore em *k* grupos:

```
grupos <- cutree(cluster_hier, k = 3)   # força 3 clusters
table(grupos)
```

`cutree()` corta o dendrograma horizontalmente numa altura correspondente a exatamente *k* ramos — é o que converte a árvore hierárquica numa atribuição de cluster discreta por observação, do mesmo jeito que você usaria `kmeans()` (seção 2.6).

2.5 Validação: os clusters fazem sentido?

O Wine dataset tem uma vantagem pedagógica rara: o rótulo real de cultivar (`classe`, 1/2/3) está disponível, mesmo que o clustering não o use. Isso permite validar se o agrupamento não-supervisionado "redescobriu" a estrutura real:

```
table(Cluster = grupos, Cultivar_real = vinho$classe)
```

Aviso importante: ter o rótulo verdadeiro disponível para conferência é uma situação de **aprendizado**, não a realidade da maioria dos problemas de clustering aplicados — normalmente você faz cluster exatamente porque não sabe os grupos reais (segmentação de clientes, por exemplo, não vem com um rótulo "segmento verdadeiro" pré-existente). Aqui a tabela de contingência serve para validar se o método está funcionando corretamente antes de confiar nele num problema sem essa rede de segurança.

2.6 K-means (extensão — complemento natural ao hierárquico)

Não estava no escopo original deste módulo, mas é impossível falar de clustering aplicado sem mencionar o método mais usado na prática ao lado do hierárquico: **k-means**. Diferença central: k-means exige que você defina *k* **antes** de rodar (não corta uma árvore depois — ele particiona diretamente), e usa um algoritmo iterativo (não hierárquico) que minimiza a soma de distâncias quadráticas de cada ponto ao centro (centroide) do seu cluster.

```
set.seed(42)   # k-means depende de inicialização aleatória -- fixe a
semente para reprodutibilidade
km <- kmeans(vinho_padronizado, centers = 3, nstart = 25)

table(Cluster = km$cluster, Cultivar_real = vinho$classe)
```

`nstart = 25` roda o algoritmo 25 vezes com pontos de partida aleatórios diferentes e mantém o melhor resultado — k-means pode convergir para um mínimo local ruim dependendo da inicialização, então rodar só uma vez é arriscado.

Quando usar hierárquico vs. k-means: hierárquico não exige decidir *k* de antemão (você decide olhando o dendrograma) e produz uma estrutura de sub-grupos aninhados (útil quando faz sentido pensar em hierarquia real, como taxonomia biológica); k-means é mais rápido e escala melhor para *n* grande (não precisa computar a matriz de distância completa $n \times n$), mas exige *k* definido a priori. Na prática, é comum rodar os dois e comparar.

Escolhendo *k* sem rótulo disponível — método do cotovelo (*elbow method*), olhando a soma de quadrados dentro dos clusters (`km$tot.withinss`) para vários valores de *k*:

```
wss <- sapply(1:10, function(k) kmeans(vinho_padronizado, centers = k, nst
art = 25)$tot.withinss)

plot(1:10, wss, type = "b", xlab = "Número de clusters (k)", ylab = "Soma
de quadrados intra-cluster",
     main = "Método do cotovelo")
```

O “cotovelo” — o ponto onde adicionar mais um cluster deixa de reduzir a soma de quadrados de forma expressiva — é a heurística visual para escolher *k*. Não é um critério estatístico formal (não há teste de hipótese aqui), é uma convenção prática.

---

3. PCA — Análise de Componentes Principais

3.1 O que é e para que serve

PCA responde: dado um conjunto de *p* variáveis correlacionadas entre si, existe um número menor de eixos (**componentes principais**) que resume a maior parte da variação total dos dados? Cada componente principal é uma **combinação linear** das variáveis originais, construída de forma que:

- O primeiro componente (PC1) captura a maior variância possível dos dados.
- O segundo componente (PC2) captura a maior variância possível **entre o que sobrou**, sob a restrição de ser ortogonal (não-correlacionado) ao PC1.
- E assim por diante — cada componente subsequente captura o máximo de variância restante, sempre ortogonal a todos os anteriores.

O resultado: se as variáveis originais são fortemente correlacionadas entre si (muita informação redundante), poucos componentes já capturam a maior parte da variância total — é aí que está o ganho de "reduzir dimensionalidade sem perder muita informação".

Quando PCA NÃO ajuda: se as *p* variáveis originais já são pouco correlacionadas entre si, não há redundância a comprimir — os componentes principais acabam sendo próximos das próprias variáveis originais, e a "redução de dimensionalidade" não economiza quase nada. PCA é uma ferramenta para explorar **redundância**, não mágica de compressão universal.

3.2 A matemática por dentro

Partindo dos dados padronizados Z (seção 1 — obrigatório fazer antes), o PCA opera sobre a **matriz de covariância** de Z (que, por os dados já estarem padronizados, é numericamente idêntica à matriz de correlação dos dados originais):

$$ \Sigma = \frac{1}{n-1} Z^T Z $$

O PCA resolve o problema de **autovalores e autovetores** dessa matriz:

$$\Sigma v_j = \lambda_j v_j$$

- Cada **autovetor** v_j define a direção do componente principal j no espaço original de p variáveis — os pesos (**cargas fatoriais**, *loadings*) que multiplicam cada variável original para formar aquele componente.
- Cada **autovalor** λ_j é a variância capturada por aquele componente. Como os dados estão padronizados (cada variável com variância 1), a soma de todos os autovalores é igual a p (o número de variáveis) — a variância total do sistema não muda, o PCA só a redistribui entre novos eixos ordenados por importância.

Os **escores** (scores) — a posição de cada observação no novo sistema de eixos — se calculam projetando os dados padronizados sobre os autovetores:

$$\text{Escore}_j = Zv_j$$

Essa é literalmente a "rotação" do espaço original para um novo sistema de coordenadas, escolhido especificamente para alinhar o primeiro eixo com a direção de maior variância dos dados. Nenhuma informação é criada ou

descartada nesse passo — é uma transformação linear reversível quando se mantêm todos os p componentes; a "perda" só acontece quando você decide manter apenas os primeiros $k < p$ componentes (seção 3.4).

3.3 Calculando no R

Base R — `prcomp()`:

```
pca_vinho <- prcomp(vinho_num, scale. = TRUE)
```

`scale. = TRUE` é o argumento que padroniza as variáveis automaticamente antes do cálculo — equivalente a passar `vinho_padronizado` (já centralizado e escalado) em vez de `vinho_num` bruto. **Nunca rode PCA sem isso** (ou sem padronizar manualmente antes), pelo mesmo motivo da seção 1.

```
summary(pca_vinho)
```

O `summary()` retorna, por componente: o desvio padrão (raiz do autovalor), a **proporção de variância explicada** e a **proporção acumulada** — a tabela central para decidir quantos componentes manter (seção 3.4).

```
pca_vinho$rotation   # matriz de cargas fatoriais (loadings) -- os
autovetores
pca_vinho$x           # escores de cada observação em cada componente
```

Pacote `FactoMineR` — alternativa mais orientada a visualização e interpretação direta, popular em análise exploratória multivariada acadêmica no Brasil e na França:

```
install.packages(c("FactoMineR", "factoextra"))
library(FactoMineR)
library(factoextra)

pca_fm <- PCA(vinho_num, scale.unit = TRUE, graph = FALSE)
```

`FactoMineR::PCA()` já calcula automaticamente contribuições de cada variável a cada componente e qualidade de representação ($\cos^2$) — métricas que `prcomp()` não devolve prontas, exigindo cálculo manual a partir dos loadings.

3.4 Quantos componentes manter

Scree plot — gráfico da variância explicada por componente, em ordem decrescente:

```
plot(pca_vinho, type = "l", main = "Scree plot -- Wine dataset")
# ou, com factoextra, mais legível:
fviz_eig(pca_fm, addlabels = TRUE)
```

Critérios de decisão (nenhum é uma regra matemática rígida — são convenções práticas, use mais de um em conjunto):

1. **Variância acumulada** — manter componentes suficientes para explicar uma fração alvo da variância total (convenção comum: 70-90%, dependendo de quanto de perda de informação é aceitável para o objetivo do estudo).
2. **Critério de Kaiser** — manter apenas componentes com autovalor > 1 (ou seja, que capturam mais variância do que uma única variável padronizada capturaria sozinha — um componente com autovalor < 1 está, em certo sentido, resumindo menos informação do que uma variável original isolada já teria).
3. **"Cotovelo" no scree plot** — mesma lógica visual do método do cotovelo em k-means (seção 2.6): procurar o ponto onde a curva de variância explicada para de cair de forma expressiva e vira quase reta.

3.5 Interpretando os componentes: cargas fatoriais e círculo de correlação

Um componente principal, sozinho, é só um número — a interpretação vem de olhar **quais variáveis originais pesam mais** nele (maiores cargas fatoriais em valor absoluto):

```
pca_vinho$rotation[, 1:2]   # cargas do PC1 e PC2
```

No Wine dataset, é comum PC1 carregar fortemente variáveis relacionadas a fenóis/flavonoides (`Flavanoids` , `Total_phenols` , `Proanthocyanins` — todas quimicamente correlacionadas entre si) e PC2 capturar um eixo relacionado a cor/intensidade (`Color_intensity` , `Hue`) — a interpretação exata depende dos dados baixados, então confira sempre os `loadings` reais em vez de assumir esse padrão de antemão.

Círculo de correlação — visualização de quanto cada variável original se correlaciona com os dois primeiros componentes:

```
fviz_pca_var(pca_fm, col.var = "contrib", gradient.cols = c("steelblue", "
firebrick"))
```

Vetores próximos entre si (ângulo pequeno) indicam variáveis originais fortemente correlacionadas entre si; vetores próximos a 90° indicam quase nenhuma correlação; vetores em direções opostas (próximos a 180°) indicam correlação forte e negativa. O comprimento do vetor indica o quão bem aquela variável é representada pelos dois componentes plotados ($\cos^2$).

3.6 Visualização de indivíduos e biplot

```
fviz_pca_ind(pca_fm, col.ind = as.factor(vinho$classe), addEllipses =
TRUE,
             legend.title = "Cultivar")

fviz_pca_biplot(pca_fm, col.ind = as.factor(vinho$classe), col.var = "blac
k",
                 legend.title = "Cultivar")
```

O **biplot** combina os dois gráficos anteriores num só: pontos são as observações (vinhos) projetadas em PC1×PC2, setas são as variáveis originais — permite ler, num único gráfico, tanto quais vinhos se parecem entre si quanto quais variáveis originais estão "puxando" cada agrupamento de vinhos naquela direção. Colorir os pontos por `classe` real (mesmo raciocínio da seção 2.5) é uma checagem visual direta de que os dois primeiros componentes já separam bem os três cultivares — se a separação visual for nítida com só 2 de 13 dimensões, isso é evidência forte de que a maior parte da variância relevante do dataset está concentrada nesses dois eixos.

---

4. Combinando PCA e cluster

Uma prática comum e útil: rodar clustering **sobre os escores do PCA** (em vez de sobre as 13 variáveis padronizadas originais), mantendo só os primeiros componentes que já cobrem a maior parte da variância:

```
escores_pca <- pca_vinho$x[, 1:3]   # primeiros 3 componentes, por exemplo

cluster_sobre_pca <- hclust(dist(escores_pca), method = "ward.D2")
grupos_pca <- cutree(cluster_sobre_pca, k = 3)

table(Cluster_via_PCA = grupos_pca, Cultivar_real = vinho$classe)
```

Por que isso pode ajudar: reduzir de 13 para 3 dimensões antes de calcular distâncias remove ruído de variáveis pouco informativas (que só adicionam variância irrelevante à distância euclidiana) e reduz o custo computacional em datasets muito maiores que este. **Risco a ter em mente:** você está clusterizando sobre uma versão comprimida dos dados — se informação relevante para a separação dos grupos estivesse concentrada nos componentes descartados (o que não costuma acontecer se os primeiros componentes já cobrem 80%+ da variância, mas é uma suposição, não uma garantia), o cluster sobre PCA perderia esse sinal.

Mesmo quando você não usa o PCA reduzido para clusterizar de fato, ele continua sendo a ferramenta padrão para **visualizar** um clustering feito sobre as 13 dimensões originais (impossível de plotar diretamente) — projetando os clusters do espaço original sobre o plano PC1×PC2 só para fins de visualização:

```
fviz_cluster(list(data = vinho_padronizado, cluster = grupos), geom = "poi
nt",
             ellipse.type = "norm", main = "Cluster hierárquico (Ward),
visualizado via PCA")
```

---

5. Diferenças-chave vindas de Python

Conceito	Python (sklearn / scipy)	R
Padronização	`sklearn.preprocessing.StandardScaler().fit_transform(X)`	`scale(X)`
Distância par a par	`scipy.spatial.distance.pdist(X)`	`dist(X)`
Cluster hierárquico	`scipy.cluster.hierarchy.linkage(X, method='ward')`	`hclust(dist(X` `"ward.D2")`
Dendrograma	`scipy.cluster.hierarchy.dendrogram(...)`	`plot(hclust_`

Conceito	Python (sklearn / scipy)	R
Cortar em k clusters	`scipy.cluster.hierarchy.fcluster(Z, k, criterion='maxclust')`	`cutree(hclust`
K-means	`sklearn.cluster.KMeans(n_clusters=k, n_init=25)`	`kmeans(X, cen = 25)`
PCA	`sklearn.decomposition.PCA(n_components=k)`	`prcomp(X, sca FactoMineR::P`
Variância explicada	`pca.explained_variance_ratio_`	`summary(prcon $importance`
Cargas fatoriais (loadings)	`pca.components_`	`prcomp_obj$ro`
Escores (scores)	`pca.transform(X)`	`prcomp_obj$x`
Biplot	Sem função nativa — `matplotlib` manual ou pacote `pca` de terceiros	`factoextra::f` pronto

O padrão mais visível aqui: `sklearn.decomposition.PCA` e `sklearn.cluster` entregam os números (variância explicada, cargas, escores) e deixam a visualização inteiramente a cargo de `matplotlib` — o R, com `factoextra`, já entrega gráficos de PCA/cluster prontos e razoavelmente bem formatados (scree plot, círculo de correlação, biplot, cluster projetado) com uma função cada, o que reduz bastante código boilerplate de visualização para análise exploratória multivariada.

---

6. Fonte de dados: Wine dataset (UCI)

Por que este dataset: é real (análise química de 178 vinhos italianos de 3 cultivares diferentes, sem nenhum dado sintético envolvido), pequeno o bastante para rodar sem fricção, sem valores ausentes, e tem uma vantagem pedagógica rara para um módulo não-supervisionado: o rótulo verdadeiro de cultivar existe e pode ser usado para validar se cluster e PCA "recuperam" a estrutura real (seções 2.5 e 3.6) — coisa que a maioria dos problemas reais de clustering não oferece.

Como obter

Fonte primária — UCI Machine Learning Repository: https://archive.ics.uci.edu/dataset/109/wine

Clique em "Download" na página — o pacote inclui `wine.data` (os dados, sem cabeçalho, separados por vírgula) e `wine.names` (documentação das colunas). Licença CC BY 4.0 (uso livre com atribuição). Citação oficial: Aeberhard, S. & Forina, M. (1992). Wine [Dataset]. UCI Machine Learning Repository.

O arquivo `wine.data` não tem cabeçalho — a primeira coluna é a classe (cultivar, 1/2/3) e as 13 seguintes são, nesta ordem: `Alcohol`, `Malic_acid`, `Ash`, `Alcalinity_of_ash`, `Magnesium`, `Total_phenols`, `Flavanoids`, `Nonflavanoid_phenols`, `Proanthocyanins`, `Color_intensity`, `Hue`, `OD280_OD315`, `Proline`.

```
vinho <- read.csv(
  "wine.data",
  header = FALSE,
  col.names = c("classe", "Alcohol", "Malic_acid", "Ash", "Alcalinity_of_a
sh",
                "Magnesium", "Total_phenols", "Flavanoids", "Nonflavanoid_
phenols",
                "Proanthocyanins", "Color_intensity", "Hue",
"OD280_OD315", "Proline")
)

vinho$classe <- factor(vinho$classe, labels = c("Cultivar_1",
"Cultivar_2", "Cultivar_3"))
str(vinho)   # confirme 178 linhas, 14 colunas, sem NA -- se algo
divergir, confira o download
```

Alternativa — Kaggle (mesmo dataset, CSV já com cabeçalho): https://www.kaggle.com/datasets/aarontanjaya/uci-wine-dataset

Mais conveniente se você não quiser nomear as colunas manualmente — mas confira `names()` ao carregar, pelo mesmo princípio do Módulo 6: nunca assuma nomes de coluna de um espelho de terceiros sem conferir contra o arquivo que você de fato baixou.

---

7. Exercícios

Exercício 1 — Efeito da padronização

Rode `hclust()` com `ward.D2` sobre os dados **sem** padronizar (`vinho_num` bruto) e compare com o resultado sobre `vinho_padronizado`. Use `table()` cruzando os clusters de cada versão com `classe` real. A versão sem padronização recupera os 3 cultivares tão bem quanto a padronizada? Qual variável você suspeita estar dominando a distância na versão não padronizada?

Solução:

```
cluster_bruto <- hclust(dist(vinho_num), method = "ward.D2")
grupos_bruto <- cutree(cluster_bruto, k = 3)
table(grupos_bruto, vinho$classe)

cluster_padrao <- hclust(dist(vinho_padronizado), method = "ward.D2")
grupos_padrao <- cutree(cluster_padrao, k = 3)
table(grupos_padrao, vinho$classe)

apply(vinho_num, 2, var)
# compare a variância bruta de cada coluna -- a maior aqui é a suspeita
```

Espera-se que a versão padronizada recupere os 3 cultivares de forma bem mais nítida — `Proline`, com variância bruta ordens de magnitude maior que as demais colunas, tende a dominar sozinho a distância euclidiana não padronizada, distorcendo o agrupamento na direção dessa única variável.

Exercício 2 — Comparando métodos de ligação

Rode `hclust()` sobre `vinho_padronizado` com `method = "single"`, `"complete"` e `"ward.D2"`. Para cada um, corte em 3 clusters e cruze com `classe` real. Qual método recupera melhor os 3 cultivares? Isso é consistente com o que a seção 2.3 diz sobre as tendências de cada método?

Solução:

```
for (metodo in c("single", "complete", "ward.D2")) {
  h <- hclust(dist(vinho_padronizado), method = metodo)
  g <- cutree(h, k = 3)
  cat("\n---", metodo, "---\n")
  print(table(g, vinho$classe))
}
```

`single` tende a performar visivelmente pior aqui — sua propensão a "encadeamento" (seção 2.3) costuma produzir um cluster gigante e dois quase vazios em dados reais com grupos moderadamente sobrepostos. `ward.D2` tende a ser o mais equilibrado e mais próximo dos 3 cultivares reais, consistente com sua tendência a produzir clusters compactos e de tamanho comparável.

Exercício 3 — Scree plot e critério de Kaiser

Rode `prcomp()` sobre os dados padronizados. Construa o scree plot. Quantos componentes têm autovalor > 1 (critério de Kaiser)? Quantos componentes são necessários para acumular 80% da variância? Os dois critérios concordam no número de componentes a manter?

Solução:

```
pca_vinho <- prcomp(vinho_num, scale. = TRUE)

autovalores <- pca_vinho$sdev^2
sum(autovalores > 1)                          # critério de Kaiser

variancia_acumulada <- cumsum(autovalores) / sum(autovalores)
which(variancia_acumulada >= 0.80)[1]          # nº de componentes p/ 80%
da variância

plot(pca_vinho, type = "l", main = "Scree plot")
```

Com 13 variáveis moderadamente correlacionadas (não extremamente redundantes), é comum que o critério de Kaiser aponte um número de componentes (tipicamente 3-4) menor que os 13 originais, mas os dois critérios raramente coincidem exatamente — se divergirem por 1-2 componentes, prefira o critério de variância acumulada quando o objetivo é reter informação para modelagem posterior, e o critério de Kaiser quando o objetivo é interpretação (componentes com autovalor abaixo de 1 tendem a ser difíceis de interpretar de forma substantiva).

Exercício 4 — Interpretando PC1 e PC2

Extraia `pca_vinho$rotation[, 1:2]`. Para cada um dos dois primeiros componentes, identifique as 3 variáveis com maior carga em valor absoluto. Dê um nome descritivo a cada componente com base nessas variáveis (ex.: “eixo de intensidade fenólica”). Gere o círculo de correlação (`fviz_pca_var`) e confirme visualmente sua interpretação.

Solução: ver seção 3.5 para o código-base. A interpretação exata depende dos valores reais obtidos no seu ambiente — o objetivo do exercício é praticar a leitura de `rotation`, não decorar um resultado fixo. Como checagem: as 3 variáveis com maior carga em PC1 devem aparecer no círculo de correlação como vetores longos e próximos entre si em ângulo (alta correlação mútua) — se os vetores de maior carga numérica não baterem com os vetores visualmente mais alinhados no gráfico, há um erro na leitura da matriz de `rotation`.

Exercício 5 — Cluster sobre PCA vs. cluster sobre dados originais

Rode cluster hierárquico (`ward.D2`) sobre os 3 primeiros componentes principais (seção 4) e compare, via `table()`, com o cluster rodado sobre as 13 variáveis padronizadas originais (Exercício 1, versão padronizada). Os dois agrupamentos coincidem na maioria das observações? Quanto de variância os 3 componentes usados cobrem, ao todo (`summary(pca_vinho)`)?

Solução: ver seção 4 para o código-base do cluster sobre PCA;

```
table(Cluster_original = grupos_padrao, Cluster_via_PCA = grupos_pca)
summary(pca_vinho)$importance[3, 3]   # variância acumulada até PC3
```

Se os 3 primeiros componentes cobrirem uma fração alta da variância total (comum neste dataset, dado o grau de correlação química entre as variáveis), os dois agrupamentos devem coincidir na grande maioria das observações — divergências pontuais tendem a se concentrar em vinhos “de fronteira”, quimicamente intermediários entre dois cultivares.

Exercício 6 (desafio) — Método do cotovelo sem usar o rótulo real

Repita o método do cotovelo (seção 2.6) usando **apenas** `vinho_padronizado`, sem consultar `classe` em nenhum momento — simule a situação real de clustering, onde você não teria esse rótulo disponível. O cotovelo do gráfico

aponta para $k = 3$ (o número real de cultivares) de forma clara, ou o resultado é ambíguo? O que isso ensina sobre a confiabilidade do método do cotovelo como critério único?

Solução: ver código da seção 2.6. Na prática, o método do cotovelo raramente produz um "cotovelo" tão nítido e inequívoco quanto os exemplos didáticos sugerem — é comum haver mais de uma dobra plausível no gráfico. A lição pretendida: método do cotovelo é uma heurística de apoio, não uma resposta definitiva — na ausência de rótulo real (o caso comum), a decisão final de k deveria combinar o cotovelo com conhecimento de domínio sobre quantos grupos fariam sentido substantivo para o problema, não só o formato da curva.

---

8. Entrega esperada do módulo

Script `multivariada_wine.R` que:

1. Carrega e padroniza o Wine dataset
2. Roda cluster hierárquico com pelo menos 2 métodos de ligação, com dendrograma e validação contra `classe` real
3. Roda PCA completo (`prcomp` + `FactoMineR`), com scree plot, tabela de variância explicada, círculo de correlação e biplot colorido por cultivar
4. Roda cluster sobre os componentes principais e compara com o cluster sobre os dados originais

---

9. Critério de domínio antes de avançar ao Módulo 8

- ☐ Explica de cabeça por que padronizar é obrigatório antes de PCA e de clustering baseado em distância, com um exemplo concreto de variável que dominaria a distância sem isso
- ☐

Sabe diferenciar os métodos de ligação (single, complete, average, Ward) e prever qual tende a produzir clusters mais compactos

- ☐ Consegue explicar PCA em termos de autovalores/autovetores da matriz de covariância, não só "o botão que reduz dimensão"
- ☐ Sabe justificar quantos componentes manter usando pelo menos dois critérios (Kaiser, variância acumulada, scree plot) e reconhece que nenhum é uma regra rígida
- ☐ Consegue interpretar substantivamente um componente principal a partir das cargas fatoriais (`rotation`), não só reportar o número de variância explicada
- ☐ Entende a diferença prática entre cluster hierárquico e k-means, e sabe quando cada um é mais adequado
- ☐ Todos os 6 exercícios rodam sem erro, com interpretação escrita conectando o resultado numérico ao que ele significa sobre a estrutura dos dados

---

10. Nota: extensões e decisões deste módulo

1. **K-means (seção 2.6) e método do cotovelo** foram incluídos como extensão — o plano original citava só cluster hierárquico, mas k-means é o par mais comum na prática e ajuda a demonstrar o trade-off entre os dois paradigmas de clustering (hierárquico sem *k* pré-definido vs. particional com *k* pré-definido).
2. **Dataset real desde o primeiro momento**, sem versão sintética envolvida — o Wine dataset já é exatamente o tipo de exemplo que o plano original sugeria ("características de vinhos"), então não houve necessidade de escolher entre sintético e real: era real por natureza.

3. **Silhouette score**, outra métrica comum de validação de clustering (complementar ao método do cotovelo, mede o quão bem cada ponto se encaixa no seu cluster vs. no cluster vizinho mais próximo), não entrou neste módulo para não sobrecarregar — fica como candidato a menção rápida no Módulo 9, se fizer sentido no contexto de validação estatística mais formal.

PARTE 3

Inferência

A transição de descrever a amostra para inferir sobre a população: distribuições, intervalos de confiança, poder estatístico e o núcleo dos testes de hipótese.

MÓDULO 08 · INFERÊNCIA

Distribuições e Intervalos de Confiança

Estatística Inferencial I

1. Amostragem: como a amostra chega até você

Antes de qualquer fórmula de erro amostral ou intervalo de confiança, uma pergunta anterior e mais fundamental: **como** essa amostra foi produzida? Toda a teoria deste módulo (seções 6 em diante) assume implicitamente que a amostra foi obtida por um mecanismo conhecido e bem comportado. Se essa suposição falhar, a matemática continua rodando — R não se recusa a calcular um IC sobre uma amostra viesada — mas o resultado é **precisamente errado**: um intervalo de confiança estreito em torno de um número que não representa a população de interesse.

1.1 Tipos de amostragem probabilística

- **Amostragem aleatória simples (AAS):** cada elemento da população tem probabilidade igual e conhecida de ser selecionado, independente dos demais. É o caso que toda fórmula deste módulo assume por padrão (inclusive `sample()` do R, seção 6.3).
- **Amostragem estratificada:** a população é dividida em estratos (grupos internamente mais homogêneos — ex.: por região, por porte de propriedade) e a amostragem aleatória simples é aplicada **dentro** de cada estrato, com tamanho de amostra proporcional (ou não) ao tamanho do estrato. Reduz a variância da estimativa quando os estratos são de fato mais homogêneos internamente que a população como um todo — mesma lógica de "estatística por grupo revela mais que estatística global" já vista no Módulo 2 (seção 5, `area_m2` por `porte`).
- **Amostragem por conglomerados (cluster sampling):** em vez de sortear elementos individuais, sorteiam-se grupos inteiros já existentes (ex.:

sorteiam-se municípios inteiros, depois entrevistam-se todas as propriedades daquele município) — mais barato operacionalmente (não precisa de lista completa de todos os elementos, só dos conglomerados), mas tipicamente produz erro padrão **maior** que AAS de mesmo *n*, porque elementos dentro do mesmo conglomerado tendem a ser mais parecidos entre si do que dois elementos aleatórios da população (o mesmo problema de dependência espacial que o Módulo 11 formaliza como autocorrelação).

- **Amostragem sistemática:** seleciona-se todo *k*-ésimo elemento de uma lista ordenada, a partir de um ponto de partida aleatório (ex.: todo décimo produtor cadastrado). Aproxima AAS na prática quando a lista não tem periodicidade oculta relacionada a *k* — um risco real e específico desse método: se a lista de cadastro tiver alguma ciclicidade (ex.: agrupada por região a cada 10 registros), a amostra sistemática pode replicar viés estrutural.

```
library(dplyr)

pecuaria <- read.csv("pecuaria_municipal.csv", stringsAsFactors = TRUE)

# AAS
amostra_aas <- pecuaria %>% slice_sample(n = 100)

# Estratificada por região, proporcional ao tamanho de cada estrato
amostra_estratificada <- pecuaria %>%
  group_by(regiao) %>%
  slice_sample(prop = 100 / nrow(pecuaria)) %>%
  ungroup()

table(amostra_estratificada$regiao)   # proporção de cada região
preservada, por construção
```

1.2 Viés de seleção e representatividade

Viés de seleção ocorre quando o mecanismo de amostragem torna certos elementos da população sistematicamente mais (ou menos) prováveis de aparecer na amostra, de um jeito relacionado à própria variável de interesse — não é erro amostral (seção 3), é erro **não amostral**, e não desaparece com *n* maior. Exemplo concreto no seu domínio: uma pesquisa sobre

produtividade de rebanho que só coleta dados de propriedades cadastradas em um sistema de gestão digital tende a sobrerrepresentar propriedades maiores/mais tecnificadas — a proporção de propriedades pequenas/tradicionais na amostra fica sistematicamente menor que na população real, e nenhum aumento de *n* dentro desse mesmo mecanismo de coleta corrige isso.

Pergunta prática antes de confiar em qualquer dataset aberto (IBGE, INMET, DATASUS, ou qualquer base usada neste plano de estudos): quem decidiu que unidades entram na base, e esse critério está relacionado à variável que você está medindo? Um censo declarado (como a PPM/IBGE usada neste módulo, seção 12) elimina a pergunta por definição — cobre a população inteira, não uma amostra dela — mas a maioria dos datasets abertos reais **não** é censo, é amostra ou registro administrativo com sua própria regra de inclusão, nem sempre documentada com clareza.

1.3 Ponderação amostral (survey weights)

Quando a amostragem não é AAS pura (por exemplo, estratificada com proporções diferentes das da população, ou por conglomerados), cada observação não representa "a mesma fração" da população — calcular média/proporção simples sobre a amostra, ignorando essa estrutura, produz uma estimativa enviesada mesmo que a coleta em si não tenha erro. A correção é dar a cada observação um **peso amostral** (survey weight), tipicamente o inverso da probabilidade de seleção daquele elemento — pesquisas como a PNAD/IBGE usam pesos extensivamente por esse motivo.

```
install.packages("survey")
library(survey)

# Exemplo conceitual: se cada município tivesse um peso pré-calculado
(pnad-style)
# desenho_amostral <- svydesign(ids = ~1, weights = ~peso, data =
amostra_estratificada)
# svymean(~rebanho_bovino, desenho_amostral)
```

Este módulo usa a PPM/IBGE, que é censo — não exige ponderação (seção 12 explica por quê). A menção acima é conceitual, para quando você encontrar pesquisa amostral real com pesos declarados (PNAD é o caso mais comum em dado público brasileiro) — usar `mean()` / `prop.table()` direto sobre uma base com peso amostral declarado, ignorando o peso, é um erro metodológico comum e silencioso.

---

2. Probabilidade condicional e Teorema de Bayes

Uma base conceitual curta, mas que sustenta por que testes de hipótese (Módulo 9) fazem o que fazem — sem isso, "p < 0.05 rejeita H0" vira uma regra decorada, não uma conclusão entendida.

2.1 Probabilidade condicional

$$ P(A \mid B) = \frac{P(A \cap B)}{P(B)}, \qquad P(B) > 0 $$

$P(A \mid B)$ — "probabilidade de A dado que B aconteceu" — não é, em geral, igual a $P(A)$. Quando $P(A \mid B) = P(A)$, os eventos são **independentes** (saber que B aconteceu não muda nada sobre a chance de A) — a mesma independência que a Binomial (seção 4.1) exige entre tentativas, e que a autocorrelação espacial (Módulo 11) viola.

Exemplo no seu domínio: P(prenhez | vaca de determinado escore corporal) tipicamente difere de P(prenhez) marginal — condicionar em escore corporal muda a probabilidade porque as duas variáveis não são independentes.

```
# Tabela de contingência hipotética: escore corporal x resultado de IATF
tabela <- matrix(c(120, 80, 60, 140), nrow = 2, byrow = TRUE,
                 dimnames = list(Escore = c("Baixo", "Adequado"), Resulta
do = c("Vazia", "Prenha")))
tabela

prop.table(tabela, margin = 1)   # proporção de prenhez DENTRO de cada
escore -- P(prenha | escore)
```

`margin = 1` normaliza por linha — cada linha (escore) soma 100%, dando diretamente *P*(resultado | escore). Comparar essas duas linhas é comparar *P*(prenha | Baixo) com *P*(prenha | Adequado) — o mesmo tipo de tabela de contingência que o Módulo 9 formaliza com teste de qui-quadrado.

2.2 Teorema de Bayes

$$ P(A \mid B) = \frac{P(B \mid A) \, P(A)}{P(B)} $$

Bayes inverte a condicional: a partir de *P*(*B* | *A*) (mais fácil de conhecer, geralmente) e *P*(*A*) (a probabilidade a priori), calcula *P*(*A* | *B*) (o que você realmente quer saber). É essencial para quem vai lidar com dado de diagnóstico/saúde — exatamente o caso do seu mestrado.

Exemplo clássico, com números realistas para deixar a intuição concreta: um teste diagnóstico para uma doença rara tem sensibilidade de 95% (*P*(teste+ | doente) = 0.95) e especificidade de 90% (*P*(teste− | sadio) = 0.90, logo *P*(teste+ | sadio) = 0.10). A prevalência da doença na população é 1% (*P*(doente) = 0.01). Dado um resultado positivo, qual a probabilidade real de a pessoa estar doente?

```
p_doente <- 0.01
p_teste_pos_dado_doente <- 0.95
p_teste_pos_dado_sadio <- 0.10

p_teste_pos <- p_teste_pos_dado_doente * p_doente +
p_teste_pos_dado_sadio * (1 - p_doente)
p_doente_dado_teste_pos <- (p_teste_pos_dado_doente * p_doente) / p_teste_
pos
p_doente_dado_teste_pos
```

O resultado sai bem abaixo de 95% — tipicamente na casa de 8-9%, apesar de o teste "parecer" excelente (95% de sensibilidade). A razão: quando a doença é rara, o número absoluto de falsos positivos entre os muitos sadios testados supera o número de verdadeiros positivos entre os poucos doentes testados, mesmo com especificidade alta. Essa é a **falácia da taxa base** (base rate fallacy) — confundir $P(\text{teste+} \mid \text{doente})$ com $P(\text{doente} \mid \text{teste+})$ é um dos erros de interpretação mais comuns e mais consequentes em diagnóstico, e a mesma estrutura lógica do erro de interpretação do p-valor que o Módulo 9 (seção 3) vai formalizar: p-valor é $P(\text{dados} \mid H_0)$, não $P(H_0 \mid \text{dados})$ — a mesma inversão de condicional, no mesmo lugar em que a intuição costuma falhar.

2.3 Variável aleatória discreta vs. contínua

Só uma amarração de vocabulário antes da seção 4: **variável aleatória discreta** assume valores contáveis (número de vacas prenhas, número de casos de distocia — seção 4) e sua distribuição é descrita por uma **função de massa** ($P(X = k)$, um número por valor possível); **variável aleatória contínua** assume qualquer valor num intervalo (rebanho médio municipal, temperatura) e é descrita por uma **função de densidade** (seção 5) — $P(X = x)$ para um valor exato é sempre zero numa contínua; só intervalos ($P(a < X < b)$) têm probabilidade positiva. A confusão entre "altura da curva de densidade" e "probabilidade" é comum: densidade pode ultrapassar 1 (não é limitada a $[0, 1]$ como probabilidade), porque o que integra a 1 é a **área**, não a altura pontual.

Duas fontes de “a amostra não bate exatamente com a população” — distinguir as duas é o primeiro passo antes de qualquer fórmula:

- **Erro amostral (sampling error):** a diferença entre $\bar{x}$ e μ que existe só porque você observou uma parte, não o todo — mesmo com coleta perfeita, sem nenhum viés, o erro amostral existe simplesmente porque uma amostra de $n = 100$ municípios não é o Brasil inteiro. É quantificável e diminui (de forma previsível, seção 5) conforme n aumenta. Toda a teoria deste módulo lida com **este** tipo de erro.
- **Erro não amostral (non-sampling error):** vem de problemas no processo de medição ou coleta — viés de seleção (amostra não representativa da população-alvo), erro de medição, não-resposta, erro de digitação/processamento. **Não diminui** aumentando n — coletar mais dados de forma sistematicamente enviesada só produz uma estimativa errada com mais confiança (menor intervalo de confiança em torno do número errado). Nenhuma fórmula de intervalo de confiança deste módulo corrige erro não amostral; ela assume que ele não existe.

Esse segundo ponto merece ênfase: **um intervalo de confiança estreito não é evidência de que a estimativa está correta** — ele só quantifica o erro amostral. Se a coleta tiver viés sistemático, um IC estreito só significa “estou muito confiante de um número errado”.

---

4. Distribuições de probabilidade discretas

4.1 Distribuição Binomial

Modela o número de “sucessos” em n tentativas **independentes**, cada uma com a mesma probabilidade p de sucesso — o caso clássico de “sim/não” repetido.

$$ P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}, \qquad E[X] = np, \qquad \text{Var}(X) = np(1-p) $$

Quando se aplica: tentativas independentes entre si, cada uma com exatamente dois resultados possíveis, e *p* constante em todas as tentativas.

Exemplo de aplicação direta ao seu domínio: taxa de concepção num protocolo de IATF (Inseminação Artificial em Tempo Fixo). Se a taxa histórica de concepção é 45% e você insemina um lote de 50 vacas, o número de vacas prenhas segue (aproximadamente — assumindo independência entre vacas, que na prática pode ser violada por fatores de manejo/nutrição compartilhados dentro do lote, uma ressalva real, não só acadêmica) uma Binomial(n=50, p=0.45):

```
n_vacas <- 50
p_concepcao <- 0.45

# Probabilidade de exatamente 25 vacas prenhas
dbinom(25, size = n_vacas, prob = p_concepcao)

# Probabilidade de 20 OU MENOS vacas prenhas (cauda inferior -- lote com
desempenho ruim)
pbinom(20, size = n_vacas, prob = p_concepcao)

# Simulação: 10.000 lotes hipotéticos de 50 vacas, distribuição do número
de prenhas
lotes_simulados <- rbinom(10000, size = n_vacas, prob = p_concepcao)
hist(lotes_simulados, main = "Distribuição simulada -- vacas prenhas por
lote de 50",
     xlab = "Número de vacas prenhas")
```

`dbinom()` dá a probabilidade pontual, `pbinom()` a probabilidade acumulada (útil para "qual a chance do resultado ser pior que X"), `rbinom()` gera amostras aleatórias da distribuição — o mesmo padrão `d` / `p` / `q` / `r` se repete para toda distribuição no R (já usado informalmente com `rnorm` / `rgamma` no dataset sintético dos módulos anteriores).

4.2 Distribuição de Poisson

Modela o número de ocorrências de um evento **raro** num intervalo fixo (de tempo, área ou outra unidade), quando os eventos acontecem de forma independente a uma taxa média conhecida λ:

$$ P(X = k) = \frac{\lambda^k e^{-\lambda}}{k!}, \qquad E[X] = \text{Var}(X) = \lambda $$

Quando se aplica: contagem de eventos raros e independentes num intervalo — diferente da Binomial, não há um "número de tentativas" fixo nem um "fracasso" bem definido (não faz sentido perguntar "quantas vezes o parto NÃO foi difícil" da mesma forma que se pergunta "quantas moedas NÃO deram cara").

Exemplo: se uma propriedade registra, em média, $\lambda = 2.3$ casos de distocia (parto difícil) por mês:

```
lambda_distocia <- 2.3

# Probabilidade de exatamente 0 casos no mês (mês "limpo")
dpois(0, lambda = lambda_distocia)

# Probabilidade de mais de 5 casos no mês (mês atípico, investigar causa)
1 - ppois(5, lambda = lambda_distocia)
```

A média igual à variância ($E[X] = \text{Var}(X) = \lambda$) é uma assinatura da Poisson — na prática, contagens reais frequentemente têm variância **maior** que a média (sobredispersão/*overdispersion*, por exemplo quando casos de distocia se agrupam em surtos ligados a um touro ou lote específico, violando a independência assumida) — quando isso acontece, a Poisson deixa de ser o modelo adequado e entra a Binomial Negativa, fora do escopo deste módulo introdutório mas importante registrar que a Poisson não é a resposta universal para toda contagem.

---

5. Distribuição Normal e Normal padrão

Já usada informalmente desde o Módulo 1 (curva de sino, `rnorm()` no dataset sintético) — aqui, a formalização:

$$ f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} $$

Normal padrão ($Z \sim N(0, 1)$): o caso especial com $\mu = 0$, $\sigma = 1$. Qualquer normal pode ser convertida à normal padrão via padronização (mesma transformação z-score do Módulo 7, seção 1):

$$ z = \frac{x - \mu}{\sigma} $$

```
pnorm(1.96)          # área acumulada até z = 1.96 -- deve dar ~0.975
qnorm(0.975)         # o z que corresponde a 97.5% de área acumulada --
deve dar ~1.96
```

`qnorm()` é o inverso de `pnorm()` — dado uma probabilidade, devolve o valor de z. Essa dupla ($z = 1.96$ para 97.5% acumulado, equivalente a 95% de área central) é a constante que vai reaparecer sem parar na seção 8 (intervalo de confiança de 95%) — vale internalizar de onde ela vem, não só decorar o número.

---

6. Distribuição amostral

6.1 O que é

A **distribuição amostral de uma estatística** (aqui, focando na média $\bar{x}$) é a distribuição de probabilidade que você obteria se repetisse o processo "tirar uma amostra de tamanho n, calcular $\bar{x}$" infinitas vezes. Não é a distribuição dos dados originais — é a distribuição de um **resumo** calculado repetidamente sobre amostras diferentes.

6.2 Por que ela tem menos variância que a população

$$ \sigma_{\bar{x}} = \frac{\sigma}{\sqrt{n}} $$

Esse é o **erro padrão da média** (*standard error*) — o desvio padrão da distribuição amostral da média, não o desvio padrão dos dados originais (confundir os dois é um dos erros mais comuns em relatórios aplicados). Ele diminui com $\sqrt{n}$: quadruplicar o tamanho da amostra reduz o erro padrão pela metade, não por 4× — retornos decrescentes que já valem para a decisão prática de "vale a pena dobrar minha amostra?" (seção 9 formaliza isso via cálculo de tamanho de amostra).

6.3 Demonstrando com dado real — simulação

Aqui é onde o censo completo do IBGE vira ferramenta didática. Suponha `pecuaria` já carregado (seção 10) com uma linha por município e a coluna `rebanho_bovino`:

```
mu_real <- mean(pecuaria$rebanho_bovino)     # a média POPULACIONAL real
-- você tem o censo inteiro
sigma_real <- sd(pecuaria$rebanho_bovino)

cat("Média populacional real:", round(mu_real), "\n")
cat("Desvio padrão populacional real:", round(sigma_real), "\n")

# Simule 2000 amostras de tamanho n = 30, calcule a média de cada uma
set.seed(42)
medias_amostrais_n30 <- replicate(2000, mean(sample(pecuaria$rebanho_bovin
o, size = 30)))

hist(medias_amostrais_n30, breaks = 40,
     main = "Distribuição amostral da média (n = 30, 2000 repetições)",
     xlab = "Média amostral do rebanho bovino")
abline(v = mu_real, col = "firebrick", lwd = 2)   # onde está a média
populacional real

sd(medias_amostrais_n30)          # erro padrão EMPÍRICO (observado na
simulação)
sigma_real / sqrt(30)             # erro padrão TEÓRICO (fórmula da seção
6.2)
```

Os dois últimos números devem ficar bem próximos — a simulação confirma empiricamente a fórmula teórica, não é só uma coincidência: é a fórmula sendo verificada na prática com dados reais, algo que a maioria dos cursos de estatística só mostra algebricamente.

---

7. Teorema do Limite Central (TLC)

7.1 Enunciado

Para amostras aleatórias suficientemente grandes, retiradas de **qualquer** distribuição populacional com média μ e variância σ^2 finitas, a distribuição amostral da média se aproxima de uma distribuição Normal:

$$ \bar{X} \xrightarrow{aprox.} N\left(\mu, \frac{\sigma^2}{n}\right) $$

O ponto que faz do TLC a base de praticamente todo teste estatístico clássico (Módulo 9 inteiro depende disso): **isso vale mesmo que a distribuição original dos dados não seja normal.** A forma da distribuição original só afeta o quão rápido (com que n) a aproximação normal fica boa — não afeta o fato de que ela eventualmente fica boa.

7.2 Demonstração visual com o dataset real

`rebanho_bovino` é um caso quase ideal para demonstrar o TLC de forma convincente: a distribuição original é **extremamente assimétrica à direita** (a maioria dos municípios tem rebanhos modestos; poucos municípios — como o líder nacional, com rebanho da ordem de milhões de cabeças, uma fração expressiva do total do país — puxam a cauda para valores muito altos). Se o TLC funcionar mesmo aqui, o argumento fica visualmente inegável:

```
library(ggplot2)
library(patchwork)   # para combinar gráficos lado a lado

hist_original <- ggplot(pecuaria, aes(x = rebanho_bovino)) +
  geom_histogram(bins = 50) +
  labs(title = "Distribuição ORIGINAL (população) -- fortemente
assimétrica")

gerar_hist_medias <- function(n_amostra, titulo) {
  medias <- replicate(2000, mean(sample(pecuaria$rebanho_bovino, size =
n_amostra)))
  ggplot(data.frame(medias = medias), aes(x = medias)) +
    geom_histogram(bins = 40) +
    labs(title = titulo, x = "Média amostral")
}

hist_n5   <- gerar_hist_medias(5,   "Distribuição amostral, n = 5")
hist_n30  <- gerar_hist_medias(30,  "Distribuição amostral, n = 30")
hist_n200 <- gerar_hist_medias(200, "Distribuição amostral, n = 200")

hist_original / (hist_n5 | hist_n30 | hist_n200)
```

O padrão esperado: com $n = 5$, a distribuição amostral da média ainda carrega bastante da assimetria original (o TLC ainda não "convergiu"). Com $n = 30$, já fica visivelmente mais próxima de uma curva normal simétrica — a convenção de "$n \geq 30$ como regra prática" que aparece em muitos livros-

texto vem exatamente desse tipo de padrão empírico, não de uma prova matemática de que 30 é mágico (para distribuições originais mais extremas que esta, $n = 30$ pode não bastar; para distribuições já bem comportadas, $n = 30$ é generoso). Com $n = 200$, a aproximação normal deve estar excelente.

---

8. Distribuição amostral da proporção

Mesma lógica da seção 4, aplicada a uma proporção $\hat{p}$ em vez de uma média — por exemplo, a proporção de municípios cujo rebanho bovino excede um limiar (digamos, 100 mil cabeças):

$$ \sigma_{\hat{p}} = \sqrt{\frac{p(1-p)}{n}} $$

```
p_real <- mean(pecuaria$rebanho_bovino > 100000)   # proporção
POPULACIONAL real
cat("Proporção real de municípios com rebanho > 100 mil cabeças:", round(p
_real, 3), "\n")

proporcoes_amostrais <- replicate(2000, mean(sample(pecuaria$rebanho_bovin
o, size = 100) > 100000))

sd(proporcoes_amostrais)                    # erro padrão empírico
sqrt(p_real * (1 - p_real) / 100)           # erro padrão teórico
```

Mesma verificação de sempre: os dois números devem bater de perto, confirmando a fórmula com simulação sobre dado real.

---

9. Intervalos de confiança

9.1 A lógica

Um intervalo de confiança de 95% **não significa** "há 95% de probabilidade de que μ esteja neste intervalo específico" — μ é um número fixo (seção 0), ou está dentro do intervalo ou não está, sem meio-termo probabilístico. A interpretação correta, frequentista: **se você repetisse o processo de amostragem e construção do intervalo muitas vezes, aproximadamente 95% dos intervalos construídos desse jeito conteriam o verdadeiro μ.** É uma propriedade do **procedimento**, não do intervalo particular que você calculou uma vez. A seção 9.5 demonstra isso literalmente por simulação — de novo, aproveitando o fato de você ter o censo completo e saber a resposta certa.

9.2 IC para a média — σ conhecido

$$ \bar{x} \pm z_{\alpha/2} \cdot \frac{\sigma}{\sqrt{n}} $$

Caso raro na prática (você quase nunca conhece σ populacional de verdade sem ter o censo completo) — mas útil como ponto de partida teórico, e um caso em que este módulo pode genuinamente usá-lo, já que temos o censo:

```
n <- 30
amostra <- sample(pecuaria$rebanho_bovino, size = n)
x_barra <- mean(amostra)

z_critico <- qnorm(0.975)   # para 95% de confiança
margem_erro <- z_critico * (sigma_real / sqrt(n))   # usando sigma REAL,
só porque temos o censo

c(x_barra - margem_erro, x_barra + margem_erro)
```

9.3 IC para a média — σ desconhecido (caso comum)

Quando σ é desconhecido (a imensa maioria dos casos reais), substitui-se por s (desvio padrão amostral) — mas essa substituição introduz incerteza extra, e a distribuição Normal deixa de ser exatamente correta. A correção: usar a **distribuição t de Student**, que tem caudas mais pesadas que a normal (reconhecendo a incerteza extra de estimar σ a partir da própria amostra) e converge para a Normal conforme n cresce:

$$ \bar{x} \pm t_{\alpha/2, \, n-1} \cdot \frac{s}{\sqrt{n}} $$

```
t.test(amostra, conf.level = 0.95)$conf.int
```

`t.test()` já devolve o intervalo de confiança pronto — não é preciso montar a fórmula manualmente (embora valha fazer isso uma vez à mão, no Exercício 2, para não tratar a função como caixa-preta). A teoria completa da distribuição t (graus de liberdade, forma, por que caudas mais pesadas) é formalizada no Módulo 9 junto com o teste t — aqui, o uso é só como base do intervalo de confiança.

9.4 IC para uma proporção

$$ \hat{p} \pm z_{\alpha/2} \cdot \sqrt{\frac{\hat{p}(1-\hat{p})}{n}} $$

```
amostra_binaria <- sample(pecuaria$rebanho_bovino, size = 100) > 100000
p_chapeu <- mean(amostra_binaria)

prop.test(x = sum(amostra_binaria), n = 100, conf.level = 0.95)$conf.int
```

`prop.test()` por padrão aplica uma correção de continuidade (correção de Yates) que ajusta o intervalo para o fato de a proporção ser, na origem, uma variável discreta sendo aproximada por uma distribuição contínua — resulta num intervalo ligeiramente diferente da fórmula "crua" acima; para desativar e comparar diretamente com a fórmula manual, use `correct = FALSE`.

9.5 Verificação empírica da interpretação do IC (a demonstração mais importante do módulo)

```
n <- 30
cobriu_mu <- replicate(1000, {
  amostra <- sample(pecuaria$rebanho_bovino, size = n)
  ic <- t.test(amostra)$conf.int
  ic[1] <= mu_real && mu_real <= ic[2]
# este IC específico capturou a média real?
})

mean(cobriu_mu)   # deve dar próximo de 0.95
```

Essa é a demonstração central da seção 9.1: rodando o procedimento de construção de IC 1000 vezes sobre amostras diferentes do mesmo censo, a fração de intervalos que efetivamente contém μ_{real} deve ficar perto de 95% — não porque cada intervalo individual "tem 95% de chance", mas porque **o procedimento**, repetido, acerta aproximadamente essa fração das vezes. Essa distinção sutil é onde a maioria das interpretações erradas de IC (inclusive em relatórios profissionais) tropeça.

9.6 IC para diferença entre duas médias

Compara duas médias populacionais (ex.: rebanho médio de municípios da Região Sul vs. Região Norte) via a diferença de médias amostrais e seu erro padrão combinado (fórmula de Welch, que não assume variâncias iguais entre os grupos — mais robusta que a versão clássica que assume variâncias iguais):

$$ (\bar{x}_1 - \bar{x}_2) \pm t_{\alpha/2} \cdot \sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}} $$

```
t.test(rebanho_bovino ~ regiao, data = subset(pecuaria, regiao %in% c("Sul
", "Norte")))$conf.int
```

Repare que esta é a mesma função `t.test()` da seção 9.3 — só muda a forma de chamada (fórmula `y ~ grupo` em vez de vetor único), o mesmo padrão de sintaxe reaproveitado desde o Módulo 6.

---

10. Determinação de tamanho de amostra

Pergunta inversa às seções anteriores: em vez de "dado *n*, qual o erro/margem?", pergunta-se "que *n* eu preciso para garantir uma margem de erro *E* específica?".

10.1 Para a média

Isolando *n* na fórmula da margem de erro (seção 9.2):

$$ n = \left(\frac{z_{\alpha/2} \cdot \sigma}{E}\right)^2 $$

```
z_critico <- qnorm(0.975)
E_desejado <- 5000   # margem de erro desejada, em cabeças de gado

n_necessario <- (z_critico * sigma_real / E_desejado)^2
ceiling(n_necessario)   # arredonda para cima -- nunca "quase" o
suficiente
```

Problema prático: essa fórmula pede σ, exatamente o parâmetro que você normalmente não conhece antes de amostrar (o paradoxo de precisar de informação da população para planejar como vai coletar dados dela).

Solução usual: usar um σ estimado de um estudo piloto, de literatura anterior, ou (menos preciso) uma regra prática como "amplitude estimada / 4" como aproximação grosseira de σ.

10.2 Para uma proporção

$$ n = \frac{z_{\alpha/2}^2 \cdot p(1-p)}{E^2} $$

Quando p é totalmente desconhecido, usa-se $p = 0.5$ — porque $p(1 - p)$ é maximizado em $p = 0.5$, então essa escolha garante o **tamanho de amostra conservador** (o maior n necessário considerando qualquer valor real de p), nunca subestimando a amostra necessária:

```
E_desejado_prop <- 0.05   # margem de erro de 5 pontos percentuais
p_estimado <- 0.5          # cenário conservador, sem informação prévia

n_necessario_prop <- (z_critico^2 * p_estimado * (1 - p_estimado)) / E_des
ejado_prop^2
ceiling(n_necessario_prop)
```

10.3 Power analysis: tamanho de amostra a partir do poder desejado

As seções 10.1 e 10.2 calculam n a partir de uma margem de erro desejada — pergunta adequada para **estimação** (intervalo de confiança). Para **teste de hipótese** (Módulo 9), a pergunta muda de forma sutil mas importante: "que n eu preciso para ter uma chance razoável de **detectar** uma diferença real, se ela existir, de um tamanho que me importa?"

Poder estatístico $(1 - \beta)$ é a probabilidade de rejeitar H_0 corretamente, dado que H_1 é verdadeira — o complemento do Erro Tipo II (β: falhar em detectar um efeito real). Um estudo com poder baixo (comum quando n é pequeno) tem alta chance de concluir "sem diferença significativa" mesmo quando existe uma diferença real relevante — não porque a diferença não existe, mas porque o estudo não tinha n suficiente para enxergá-la.

Quatro quantidades estão amarradas entre si em qualquer cálculo de poder: **tamanho de efeito** (o quão grande é a diferença que importa detectar, seção 9.6 conceitualmente relacionado — formalizado com Cohen's d no Módulo 9), ***n***, **nível de significância** (α) e **poder** ($1 - \beta$). Fixando três, calcula-se a quarta — o uso mais comum é fixar effect size, α e poder desejado (convenção: 80%) para descobrir o *n* necessário:

```
install.packages("pwr")
library(pwr)

# Quanto n preciso para detectar uma diferença de médias de tamanho de
efeito
# d = 0.5 (Cohen's d "médio" -- ver Módulo 9 para a definição formal),
# com alfa = 0.05 e poder = 0.80 (convenção-padrão)?
pwr.t.test(d = 0.5, sig.level = 0.05, power = 0.80, type = "two.sample")

# Pergunta inversa: com o n que eu já tenho, qual poder eu realmente
atingi
# para detectar esse mesmo effect size?
pwr.t.test(n = 40, d = 0.5, sig.level = 0.05, type = "two.sample")
```

Por que isso importa na prática, não só academicamente: planejar coleta de dados sem calcular poder é comum e caro — ou se coleta *n* insuficiente (estudo que nunca teria chance real de detectar o efeito de interesse, esforço de coleta desperdiçado) ou *n* excessivo (recurso gasto além do necessário). No Módulo 9, seção 9 (exercício sobre subamostragem de ANOVA) você já vai ver esse efeito na prática — poder caindo conforme *n* diminui, mesmo com a diferença real permanecendo constante; esta seção formaliza o cálculo que antecipa isso **antes** de coletar, em vez de só constatar depois.

---

11. Diferenças-chave vindas de Python

Conceito	Python (scipy.stats)	R
Binomial (pmf/cdf/ amostra)	`scipy.stats.binom.pmf/cdf/rvs`	`dbinom` / `pbinom` / `rbinom`
Poisson	`scipy.stats.poisson.pmf/cdf/rvs`	`dpois` / `ppois` / `rpois`
Normal	`scipy.stats.norm.pdf/cdf/ppf/rvs`	`dnorm` / `pnorm` / `qnorm` / `r`
IC para média (t)	`scipy.stats.t.interval(...)` ou `statsmodels`	`t.test(x)$conf.int` (ma direto, já entrega tudo em conjunto com o teste)
IC para proporção	`statsmodels.stats.proportion.proportion_confint`	`prop.test(x, n)$conf.i`
Simulação de distribuição amostral	`numpy` + loop/ `vectorize` manual	`replicate()` — sintaxe m compacta para "repita isso vezes e junte os resultados
Amostra com pesos amostrais	`pandas.DataFrame.sample(weights=...)` para sorteio; `statsmodels` para análise ponderada	`sample()` / `slice_sample(weight_by` pacote `survey` para análi ponderada completa
Power analysis	`statsmodels.stats.power.tt_ind_solve_power`	`pwr::pwr.t.test()` — me família de funções (`pwr.anova.test` , `pwr.chisq.test` , etc.) par outros testes

O padrão `d` / `p` / `q` / `r` do R (densidade, acumulada, quantil, amostra aleatória — o mesmo prefixo para qualquer distribuição: `norm` , `binom` , `pois` , `t` , `chisq` ...) não tem equivalente direto tão sistemático no `scipy.stats` ,

onde cada distribuição é um objeto com métodos (`.pdf` , `.cdf` , `.ppf` , `.rvs`) — funcionalmente equivalente, mas a convenção de nomenclatura do R, uma vez memorizada, generaliza para qualquer distribuição nova sem precisar consultar documentação de novo.

---

12. Fonte de dados: Pesquisa da Pecuária Municipal (IBGE)

Por que este dataset: é um censo completo (todos os municípios brasileiros, não uma amostra) do rebanho bovino — o que viabiliza literalmente toda a pedagogia de simulação deste módulo (seções 4, 5, 6, 7.5) — e é estatística pública do IBGE, sem ser dado proprietário de nenhuma empresa.

Como obter

Opção 1 — Portal SIDRA (interativo, recomendado para uma tabela pontual): https://sidra.ibge.gov.br/pesquisa/ppm

1. Acesse o link. Escolha a tabela de **"Efetivo dos rebanhos"** (bovino).
2. Selecione o nível geográfico **"Município"** (todos), a variável **"Efetivo dos rebanhos"** e o **último ano disponível**.
3. Exporte em formato `.csv` ou `.xlsx` .
4. A tabela exportada normalmente vem em formato "largo" com linhas de cabeçalho extras do SIDRA — confira `str()` / `head()` após importar e limpe conforme necessário (mesmo cuidado já estabelecido no Módulo 6 com dado real: **nunca assuma a estrutura exata sem conferir**).

```
library(dplyr)

pecuaria_bruto <- read.csv("tabela_ppm_municipios.csv", skip = 4, fileEnco
ding = "Latin1")
# ajuste 'skip' conforme o número real de linhas de cabeçalho do seu
export

names(pecuaria_bruto)   # confira antes de renomear

pecuaria <- pecuaria_bruto %>%
  rename(municipio = 1, rebanho_bovino = matches("Efetivo|Bovino")) %>%
  mutate(rebanho_bovino = as.numeric(rebanho_bovino)) %>%
  filter(!is.na(rebanho_bovino))
```

Opção 2 — Base dos Dados (versão já tratada, consultável via SQL): https://basedosdados.org/dataset/f7df4160-7a6f-4658-a287-3a73d412ed10

Mais conveniente se você já tem familiaridade com BigQuery/SQL do projeto — evita lidar com a formatação de exportação do SIDRA diretamente.

Opção 3 — Pacote `sidrar` (acesso programático à API do SIDRA):

```
install.packages("sidrar")
library(sidrar)

# O código exato da tabela/variável PPM deve ser confirmado no portal
SIDRA
# (sujeito a mudar entre atualizações do sistema) antes de usar em
produção
info_sidra("3939")   # explore a tabela antes de baixar -- confirme que é
a tabela certa
```

Mencionado como atalho, mas trate como algo a **verificar no portal antes de confiar** — códigos de tabela do SIDRA podem mudar ao longo do tempo, e um código errado retorna dado silenciosamente incorreto (não gera erro), o mesmo tipo de risco já discutido para nomes de coluna no Módulo 6.

---

13. Exercícios

Exercício 1 — Binomial aplicada: risco de um lote de IATF abaixo da meta

Com taxa histórica de concepção $p = 0.45$ e lote de $n = 60$ vacas, calcule: (a) probabilidade de exatamente 30 vacas prenhas; (b) probabilidade de 20 ou menos vacas prenhas (resultado "abaixo da meta"); (c) simule 5000 lotes e confirme que a proporção simulada de lotes com 20 ou menos prenhas bate com o valor calculado em (b).

Solução:

```
dbinom(30, size = 60, prob = 0.45)
pbinom(20, size = 60, prob = 0.45)

lotes <- rbinom(5000, size = 60, prob = 0.45)
mean(lotes <= 20)   # deve bater de perto com pbinom(20, 60, 0.45)
```

Exercício 2 — IC para a média, construído manualmente

Tire uma amostra de $n = 40$ municípios de `pecuaria$rebanho_bovino`. Calcule o IC de 95% para a média (a) manualmente, usando a fórmula da seção 9.3 com `qt()` no lugar de `qnorm()`; (b) via `t.test()`. Confirme que os dois batem. O IC captura a média populacional real (`mu_real`)?

Solução:

```
set.seed(1)
amostra <- sample(pecuaria$rebanho_bovino, size = 40)

x_barra <- mean(amostra)
s <- sd(amostra)
n <- 40
t_critico <- qt(0.975, df = n - 1)   # qt(), não qnorm() -- distribuição
t, sigma desconhecido

ic_manual <- c(x_barra - t_critico * s / sqrt(n), x_barra + t_critico *
s / sqrt(n))
ic_manual

t.test(amostra)$conf.int   # deve bater com ic_manual

ic_manual[1] <= mu_real && mu_real <= ic_manual[2]
```

Exercício 3 — TLC na prática

Repita a demonstração da seção 7.2 usando `rebanho_bovino`, mas desta vez para $n = 2$, $n = 50$ e $n = 500$. Em qual desses n a distribuição amostral já parece razoavelmente normal a olho nu? Isso é consistente com o grau de assimetria da distribuição original ser bem mais extremo que o de um dataset "bem comportado"?

Solução: ver função `gerar_hist_medias()` da seção 7.2, reaplicada aos três valores de n pedidos. Como `rebanho_bovino` é extremamente assimétrico (poucos municípios concentram uma fração desproporcional do rebanho nacional), espera-se que $n = 2$ ainda mostre distorção clara, $n = 50$ já esteja consideravelmente mais próximo do normal, e $n = 500$ esteja com aparência quase perfeitamente normal — mas o ponto exato de "já ficou normal o bastante" é subjetivo por natureza, não uma fronteira numérica fixa.

Exercício 4 — Verificação de cobertura do IC (replicando a seção 9.5) para diferentes níveis de confiança

Repita a simulação da seção 9.5 (1000 réplicas, $n = 30$) para IC de 90%, 95% e 99%. A fração de intervalos que captura `mu_real` sobe conforme o nível de confiança pedido sobe? Isso é o comportamento esperado — por quê?

Solução:

```
verificar_cobertura <- function(nivel_confianca, n = 30, replicas = 1000)
{
  cobriu <- replicate(replicas, {
    amostra <- sample(pecuaria$rebanho_bovino, size = n)
    ic <- t.test(amostra, conf.level = nivel_confianca)$conf.int
    ic[1] <= mu_real && mu_real <= ic[2]
  })
  mean(cobriu)
}

sapply(c(0.90, 0.95, 0.99), verificar_cobertura)
```

Espera-se cobertura empírica próxima de 90%, 95% e 99% respectivamente, crescente com o nível de confiança — porque um nível de confiança maior exige um intervalo mais largo (a fórmula da seção 9.3 usa um $t_{\alpha/2}$ maior conforme α diminui), e um intervalo mais largo tem mais chance de conter μ_{real} por pura geometria, ao custo de ser uma estimativa menos precisa (mais larga) — o trade-off central entre confiança e precisão que toda escolha de nível de confiança carrega.

Exercício 5 — Tamanho de amostra necessário

Você quer estimar o rebanho médio municipal com margem de erro de 5.000 cabeças e 95% de confiança, usando `sigma_real` como estimativa de σ. Calcule n necessário pela fórmula da seção 10.1. Tire uma amostra desse tamanho e confirme, via `t.test()`, que a margem de erro observada na prática fica de fato próxima (não necessariamente idêntica, por variação amostral) da meta de 5.000.

Solução:

```
z_critico <- qnorm(0.975)
n_necessario <- ceiling((z_critico * sigma_real / 5000)^2)
n_necessario

set.seed(7)
amostra_teste <- sample(pecuaria$rebanho_bovino, size = n_necessario)
ic_teste <- t.test(amostra_teste)$conf.int
(ic_teste[2] - ic_teste[1]) / 2   # margem de erro observada -- compare
com a meta de 5000
```

A margem de erro observada deve ficar na vizinhança de 5.000, mas raramente exatamente igual — a fórmula usa σ populacional verdadeiro (situação privilegiada só porque temos o censo completo), enquanto `t.test()` usa s estimado da amostra específica, que varia amostra a amostra.

Exercício 6 (desafio) — Diferença regional de rebanho médio

Filtre `pecuaria` para duas regiões de sua escolha (ex.: Sul vs. Centro-Oeste — historicamente a região com maior concentração de rebanho bovino no Brasil). Calcule o IC para a diferença de médias de `rebanho_bovino` entre as duas regiões via `t.test()`. O intervalo inclui zero? O que isso implica sobre a evidência de diferença real entre as duas regiões (antecipando a lógica de teste de hipótese do Módulo 9, sem ainda formalizá-la)?

Solução:

```
duas_regioes <- pecuaria %>% filter(regiao %in% c("Sul", "Centro-Oeste"))
t.test(rebanho_bovino ~ regiao, data = duas_regioes)$conf.int
```

Dado que o Centro-Oeste concentra historicamente uma fração desproporcional do rebanho bovino nacional (fronteira agropecuária, grandes propriedades), espera-se que o IC para a diferença de médias **não inclua zero**, e com folga — sinal de que a diferença observada entre as duas regiões dificilmente é só ruído amostral. Um IC que não inclui zero para

uma diferença é, informalmente, a mesma lógica por trás de "rejeitar a hipótese de que as médias são iguais" — conexão direta com o teste de hipótese que o Módulo 9 formaliza.

Exercício 7 — AAS vs. estratificada: comparando variância da estimativa

Usando `pecuaria`, tire 500 vezes uma amostra de $n = 50$ municípios por AAS pura (`slice_sample(n = 50)`) e calcule $\bar{x}$ de `rebanho_bovino` em cada réplica. Repita 500 vezes usando amostragem estratificada por `regiao`, com $n = 50$ distribuído proporcionalmente ao tamanho de cada região. Compare o desvio padrão das 500 médias nos dois cenários — a estratificada produz estimativas com menor variância?

Solução:

```
replicar_aas <- function() {
  amostra <- pecuaria %>% slice_sample(n = 50)
  mean(amostra$rebanho_bovino)
}

replicar_estratificada <- function() {
  amostra <- pecuaria %>%
    group_by(regiao) %>%
    slice_sample(prop = 50 / nrow(pecuaria)) %>%
    ungroup()
  mean(amostra$rebanho_bovino)
}

medias_aas <- replicate(500, replicar_aas())
medias_estrat <- replicate(500, replicar_estratificada())

sd(medias_aas)
sd(medias_estrat)
```

Se `regiao` for um estrato genuinamente mais homogêneo internamente do que a população total (municípios da mesma região tendem a ter rebanho mais parecido entre si do que municípios de regiões diferentes — plausível dado o padrão de concentração regional de pecuária no Brasil), a amostragem estratificada deve produzir `sd(medias_estrat)` menor que `sd(medias_aas)` — a mesma estimativa em expectativa, mas mais precisa

(menos variável) réplica a réplica. Essa é a demonstração empírica direta do ganho de precisão que justifica estratificar em pesquisas reais, ao custo de exigir que a variável de estratificação seja conhecida de antemão para toda a população (aqui, `regiao` de cada município).

Exercício 8 — Bayes na prática + power analysis para o próprio estudo

1. Refaça o cálculo de Bayes da seção 2.2, mas para uma prevalência de doença de 20% em vez de 1% (mantendo sensibilidade 95% e especificidade 90%). *P*(doente | teste+) sobe ou desce em relação ao cenário de prevalência 1%? Isso é consistente com a lógica da falácia da taxa base? (b) Você planeja comparar `rebanho_bovino` médio entre duas regiões usando uma amostra (não o censo completo). Usando `pwr.t.test()`, calcule o *n* por grupo necessário para detectar um tamanho de efeito Cohen's d = 0.3 (diferença pequena-a-moderada) com poder de 80% e $\alpha = 0.05$.

Solução:

```
# (a)
p_doente_20 <- 0.20
p_teste_pos_20 <- 0.95 * p_doente_20 + 0.10 * (1 - p_doente_20)
(0.95 * p_doente_20) / p_teste_pos_20

# (b)
library(pwr)
pwr.t.test(d = 0.3, sig.level = 0.05, power = 0.80, type = "two.sample")
```

1. *P*(doente | teste+) sobe substancialmente em relação ao cenário de prevalência 1% (deve ficar bem acima de 50% agora) — quanto maior a prevalência de base, menos a diferença entre "sensibilidade do teste" e "probabilidade real de doença dado teste positivo" se distorce, porque o número de falsos positivos deixa de dominar sobre o de verdadeiros positivos. Confirma a lógica da seção 2.2: a falácia da taxa base é mais severa exatamente quando a condição é rara. (b) O *n* por grupo deve sair bem maior que os $n = 50$ usados nos exemplos anteriores deste módulo

— effect sizes pequenos (d=0.3) exigem amostras consideravelmente maiores que effect sizes grandes para o mesmo poder, o ponto central de por que "estatisticamente não significativo" com *n* pequeno não é evidência de ausência de efeito (seção 10.3) — pode ser só um estudo subdimensionado para o tamanho de efeito real.

---

14. Entrega esperada do módulo

Script `inferencial_pecuaria.R` que:

1. Carrega o censo completo da PPM/IBGE e calcula μ, σ e p (proporção acima de um limiar) populacionais reais
2. Roda a simulação de distribuição amostral da média para pelo menos 3 tamanhos de amostra diferentes, com histogramas comparativos demonstrando o TLC
3. Constrói IC de 95% para a média (manual + `t.test()`) e para uma proporção (`prop.test()`) a partir de amostras simuladas
4. Roda a verificação empírica de cobertura do IC (seção 9.5) para pelo menos 2 níveis de confiança diferentes
5. Calcula tamanho de amostra necessário para uma margem de erro alvo definida por você
6. Compara AAS vs. amostragem estratificada por `regiao` (variância da estimativa entre as duas)
7. Calcula, via `pwr.t.test()`, o *n* necessário para detectar um tamanho de efeito à sua escolha, com poder de 80%

---

15. Critério de domínio antes de avançar ao Módulo 9

- [ ]

Explica a diferença entre parâmetro populacional e estatística amostral sem confundir os símbolos (μ vs. $\bar{x}$, σ vs. s)

- ☐ Sabe distinguir AAS, estratificada, conglomerados e sistemática, e explicar quando cada uma reduz variância ou introduz risco de viés
- ☐ Entende por que viés de seleção não desaparece com n maior, e sabe formular a pergunta certa antes de confiar em qualquer dataset aberto
- ☐ Aplica o Teorema de Bayes corretamente e explica a falácia da taxa base sem recorrer a jargão — consegue montar o exemplo do zero, não só repetir o resultado
- ☐ Distingue erro amostral de erro não amostral e explica por que só o primeiro diminui com n maior
- ☐ Sabe quando usar Binomial vs. Poisson, com um exemplo próprio de cada (fora do dataset deste módulo)
- ☐ Consegue explicar o Teorema do Limite Central sem citar a palavra "normal" referindo-se aos dados originais — o TLC é sobre a distribuição amostral da média, não sobre os dados brutos
- ☐ Interpreta corretamente um intervalo de confiança de 95% (propriedade do procedimento repetido, não probabilidade do intervalo específico) e consegue explicar por que a interpretação errada mais comum está errada
- ☐ Sabe escolher entre `qnorm` / `z` (σ conhecido) e `qt` /t de Student (σ desconhecido) para IC de média
- ☐ Sabe calcular tamanho de amostra necessário para uma margem de erro definida, para média e para proporção
- ☐ Sabe calcular tamanho de amostra a partir de effect size + poder desejado (`pwr`), e explica por que essa é uma pergunta diferente de "margem de erro desejada"

- ☐ Todos os 8 exercícios rodam sem erro, com as simulações de cobertura/TLC produzindo resultados consistentes com o esperado teoricamente

---

16. Nota: decisões deste módulo

1. **Distribuições discretas (Binomial/Poisson) usaram exemplos ilustrativos do domínio zootécnico** (taxa de concepção em IATF, casos de distocia) em vez do censo da PPM diretamente — esses exemplos não precisam de dado baixado (são só a mecânica da distribuição aplicada a um cenário realista), enquanto o censo da PPM foi reservado para a parte que de fato exige população completa: distribuição amostral, TLC e verificação de cobertura de IC. Fez mais sentido usar a ferramenta certa para cada parte do conteúdo do que forçar um único dataset em tudo.
2. `sigma_real` **conhecido é um privilégio pedagógico deste módulo**, não a situação real mais comum — na prática você quase nunca vai ter o censo completo para calcular σ de antemão. A seção 9.3 (IC com σ desconhecido, via t de Student) é o caso que você vai usar quase sempre fora deste ambiente de aprendizado; a seção 9.2 (σ conhecido) existe aqui só porque a simulação torna possível demonstrá-la com honestidade.
3. **Amostragem (seção 1), probabilidade condicional/Bayes (seção 2) e power analysis (seção 10.3)** foram incorporados nesta revisão, motivados por retorno externo de revisão de conteúdo — todas as seções e subseções do módulo foram renumeradas de forma consistente para acomodar as duas novas seções no início. Sem pendência.
4. **Seção de amostragem ponderada (survey weights, seção 1.3)** ficou em nível conceitual — a PPM/IBGE usada no restante do módulo é censo, não exige peso amostral, então não há dado deste módulo para exercitar `survey::svydesign()` de ponta a ponta. Fica registrado como candidato a exercício prático se um módulo futuro usar uma pesquisa amostral real com pesos declarados (PNAD é o caso mais provável).

5. **Assimetria/curtose numérica e silhouette score** — ambos já resolvidos no Módulo 9 (seções 5.1 e 10, respectivamente). Referência antiga removida por estar desatualizada.

MÓDULO 09 · INFERÊNCIA

Testes de Hipótese

Estatística Inferencial II

1. O que é um teste de hipótese

Um teste de hipótese é, na essência, uma **prova por contradição estatística**: você assume que não há efeito/diferença (hipótese nula, H_0), e verifica se os dados observados seriam muito improváveis sob essa suposição. Se forem improváveis o bastante, você rejeita H_0 em favor da hipótese alternativa (H_1).

- **H_0 (hipótese nula):** a afirmação "padrão", geralmente de ausência de efeito, diferença ou relação — o que você assumiria por padrão sem evidência em contrário (ex.: "a média de rendimento é igual entre as duas regiões").
- **H_1 (hipótese alternativa):** o que você suspeita ou quer demonstrar — o que os dados precisam evidenciar fortemente para você abandonar H_0.

Ponto crítico, e a fonte do erro de interpretação mais comum: um teste de hipótese nunca **prova** H_0 verdadeira. "Não rejeitar H_0" significa "os dados não deram evidência suficiente contra H_0", não "H_0 está confirmada" — a ausência de evidência de diferença não é evidência de ausência de diferença. Essa assimetria (é possível rejeitar H_0 com confiança, nunca "aceitar" H_0 com o mesmo rigor) está embutida na própria lógica de prova por contradição.

2. Erro Tipo I, Erro Tipo II e poder do teste

Toda decisão de um teste de hipótese pode dar errado de duas formas distintas:

	H_0 verdadeira (na realidade)	H_0 falsa (na realidade)
Rejeitar H_0	Erro Tipo I (falso positivo) — probabilidade α	Decisão correta — probabilidade $1 - \beta$ (**poder do teste**)
Não rejeitar H_0	Decisão correta — probabilidade $1 - \alpha$	Erro Tipo II (falso negativo) — probabilidade β

- **Erro Tipo I** (α): concluir que há uma diferença/efeito quando, na realidade, não há. É a taxa de erro que você **controla diretamente** ao escolher o nível de significância (convencionalmente $\alpha = 0.05$) — é literalmente a definição de α: a probabilidade de rejeitar H_0 incorretamente, se H_0 for de fato verdadeira.
- **Erro Tipo II** (β): deixar de detectar uma diferença/efeito que realmente existe. Menos falado, mas igualmente custoso na prática — um teste com pouco poder (amostra pequena, efeito sutil) pode "não encontrar diferença" simplesmente porque não tinha sensibilidade para detectá-la, não porque a diferença não existe.
- **Poder do teste** ($1 - \beta$): a probabilidade de detectar corretamente um efeito real, dado que ele existe. Cresce com n maior, efeito real maior, e variância menor — a mesma lógica de tamanho de amostra do Módulo 8 (seção 8) se aplica aqui: um teste de hipótese mal dimensionado (amostra pequena demais para o efeito que se quer detectar) tem alto risco de Erro Tipo II, mesmo com um efeito real presente.

Trade-off entre α e β: para n fixo, reduzir α (exigir mais evidência para rejeitar H_0, ex.: usar $\alpha = 0.01$ em vez de 0.05) aumenta β — fica mais difícil também detectar efeitos reais. A única forma de reduzir os dois simultaneamente é aumentar n.

---

3. O que o p-valor realmente significa

Definição correta: o p-valor é a probabilidade de observar um resultado tão extremo ou mais extremo que o observado, **assumindo que H_0 é verdadeira**.

p-valor = P(dado tão ou mais extremo que o observado | H_0 verdadeira)

O que o p-valor NÃO significa — os erros de interpretação mais comuns, inclusive em relatórios técnicos profissionais:

- **"O p-valor é a probabilidade de H_0 ser verdadeira."** Errado — o p-valor é calculado **assumindo** H_0 verdadeira; ele não pode, ao mesmo tempo, medir a probabilidade dessa suposição. A probabilidade de H_0 ser verdadeira dado os dados é uma pergunta bayesiana diferente, que exige informação adicional (probabilidade a priori) que o p-valor sozinho não fornece.
- **"p < 0.05 significa que o efeito é grande/importante."** Errado — p-valor mistura magnitude do efeito com tamanho da amostra. Com n grande o suficiente, até uma diferença trivial e irrelevante na prática produz p-valor baixo (já visto no Módulo 4, seção 1.4, com correlação). Significância estatística ≠ relevância prática — sempre reporte a magnitude do efeito (diferença de médias, tamanho de efeito) ao lado do p-valor, nunca o p-valor sozinho.
- **"p = 0.06 e p = 0.04 são conclusões qualitativamente diferentes."** Tecnicamente, um cruza o corte convencional de 0.05 e outro não —

mas os dois representam níveis de evidência quase idênticos contra H_0. Tratar $\alpha = 0.05$ como uma fronteira mágica binária (em vez de uma convenção contínua) é um vício de leitura comum.

- **"p-valor não significativo prova que não há efeito."** Já coberto na seção 1 — ausência de evidência não é evidência de ausência.

Critério de decisão: rejeita-se H_0 quando p-valor $< \alpha$ (nível de significância escolhido **antes** de olhar os dados — escolher α depois de ver o p-valor, ajustando o corte para "dar certo", é uma prática estatisticamente inválida conhecida informalmente como *p-hacking*).

---

4. Teste t

4.1 Para uma média (comparação com um valor de referência)

$H_0 : \mu = \mu_0$ (a média populacional é igual a um valor de referência específico). Estatística de teste:

$$ t = \frac{\bar{x} - \mu_0}{s/\sqrt{n}} $$

Compare com a distribuição t de Student com $n - 1$ graus de liberdade (a mesma distribuição usada para o intervalo de confiança no Módulo 8, seção 7.3 — IC e teste de hipótese são, matematicamente, duas faces da mesma moeda: um IC de 95% para μ que não contém μ_0 equivale a rejeitar $H_0 : \mu = \mu_0$ com $\alpha = 0.05$).

```
t.test(amostra, mu = valor_de_referencia)
```

4.2 Para diferença de duas médias

$H_0 : \mu_1 = \mu_2$ (equivalentemente, $\mu_1 - \mu_2 = 0$).

- **Variâncias desconhecidas e assumidas diferentes (Welch, o padrão em R):**

```
t.test(variavel ~ grupo, data = df)   # var.equal = FALSE é o padrão
```

- **Variâncias desconhecidas e assumidas iguais (teste t clássico "pooled"):**

```
t.test(variavel ~ grupo, data = df, var.equal = TRUE)
```

Qual usar: o teste de Welch é mais robusto (não exige a premissa de variâncias iguais, que muitas vezes é irrealista) e é o padrão do R por boa razão — use `var.equal = TRUE` só quando você já testou formalmente que as variâncias são homogêneas (seção 6) e quer o teste ligeiramente mais poderoso que essa premissa extra permite.

4.3 Para amostras pareadas

Já visto informalmente no Módulo 6 em contraste com o Wilcoxon signed-rank — a versão paramétrica correspondente:

```
t.test(depois, antes, paired = TRUE)
```

Premissa adicional em relação ao teste de duas amostras independentes: a normalidade que importa é a das **diferenças** $d_i = y_i - x_i$, não a de cada variável isoladamente — os dois grupos originais podem até não ser normais, desde que a distribuição das diferenças pareadas seja aproximadamente normal.

---

5. Testes de normalidade — pré-requisito para saber se o teste t é válido

Toda a seção 4 assume normalidade (dos dados, ou da distribuição amostral da média via TLC se *n* for grande o suficiente — Módulo 8, seção 5). Esta seção fecha o ciclo aberto desde o Módulo 6: **como decidir, com rigor, se essa premissa se sustenta ou se você deveria estar na via não-paramétrica.**

5.1 Diagnóstico numérico complementar: assimetria e curtose

Esta é a pendência registrada desde o Módulo 2 — o momento certo de fechá-la, porque assimetria e curtose são sintomas numéricos que **antecipam** o que um teste formal de normalidade provavelmente vai concluir.

```
install.packages("e1071")
library(e1071)

skewness(dados)
# assimetria: 0 = simétrica; > 0 = cauda à direita; < 0 = cauda à esquerda
kurtosis(dados)     # curtose (excesso): 0 = "cauda normal"; > 0 = caudas
mais pesadas que a normal
```

- **Assimetria (skewness):** mede o desequilíbrio da distribuição em torno da média. Um valor de referência prático (não uma lei estatística rígida): |*skewness*| < 0.5 é geralmente tratado como aproximadamente simétrico; |*skewness*| > 1 já indica assimetria substancial o bastante para preocupar a validade do teste t.
- **Curtose (kurtosis, forma "excesso" — a convenção do `e1071`, que já subtrai 3 para que a normal tenha curtose 0):** mede o peso das caudas relativo à normal. Curtose positiva alta (*leptocúrtica*) indica caudas pesadas e pico mais alto que a normal — mais propensão a outliers extremos do que uma normal com o mesmo desvio padrão sugeriria.

Esses dois números não são um teste de hipótese (não têm p-valor associado nesta forma simples) — são diagnóstico descritivo que orienta a leitura visual e formal que seguem. Um `skewness` de 2.3, por exemplo, já é forte indício de que Shapiro-Wilk vai rejeitar normalidade antes mesmo de rodar o teste.

5.2 Avaliação visual: histograma e Q-Q plot

```
hist(dados, main = "Histograma")

qqnorm(dados, main = "Q-Q plot")
qqline(dados, col = "firebrick")   # linha de referência -- pontos
alinhados a ela = normal
```

O Q-Q plot (*quantile-quantile*) compara os quantis observados dos dados com os quantis teóricos que uma normal perfeita teria — pontos que se afastam da linha de referência **nas pontas** (mas seguem a linha no meio) são o sinal clássico de caudas mais pesadas ou mais leves que a normal; um afastamento sistemático em forma de "S" ou curva ao longo de todo o gráfico é sinal de assimetria.

5.3 Shapiro-Wilk

O teste de normalidade mais usado na prática para amostras pequenas/médias:

H_0 : os dados vêm de uma distribuição normal

```
shapiro.test(dados)
```

Limitação importante, já adiantada informalmente desde o Módulo 6: com *n* grande (algumas centenas ou mais), Shapiro-Wilk tende a rejeitar normalidade mesmo para desvios pequenos e sem relevância prática para a validade do teste t — o mesmo problema geral do p-valor com *n* grande (seção 3). Limite prático de aplicação no R: `shapiro.test()` só aceita entre 3

e 5000 observações; para bases maiores, tire uma amostra aleatória de até 5000 pontos para o teste, ou confie mais na leitura visual (seção 5.2) e no TLC (Módulo 8) do que num único teste formal.

5.4 Kolmogorov-Smirnov

Compara a função de distribuição acumulada (CDF) empírica dos dados com a CDF teórica de uma normal específica:

```
ks.test(dados, "pnorm", mean = mean(dados), sd = sd(dados))
```

Cuidado técnico: ao estimar `mean` e `sd` a partir dos próprios dados (como no código acima) em vez de especificá-los de antemão, o teste perde parte de sua validade teórica original (o KS clássico assume que os parâmetros da distribuição de referência são conhecidos, não estimados da amostra) — na prática, isso tende a tornar o teste **conservador demais** (menos propenso a rejeitar do que deveria). Por essa razão, Shapiro-Wilk é geralmente preferido a KS especificamente para testar normalidade; KS é mais apropriado quando se compara contra uma distribuição totalmente especificada de antemão, ou para comparar duas amostras entre si (`ks.test(amostra1, amostra2)`, sem envolver normalidade).

5.5 Anderson-Darling

Variante do KS que dá peso extra às caudas da distribuição — mais sensível a desvios de normalidade justamente na região (valores extremos) que mais importa para a validade de testes que dependem de caudas bem comportadas:

```
install.packages("nortest")
library(nortest)

ad.test(dados)
```

5.6 Quando usar qual, e o que fazer se a normalidade falhar

Recomendação prática: Shapiro-Wilk como teste formal principal (para *n* moderado), sempre acompanhado da leitura visual (histograma + Q-Q plot) e do diagnóstico de assimetria/curtose — nunca decida só pelo p-valor de um teste de normalidade isolado, pela mesma razão da seção 3 (p-valor isolado não conta a história toda, especialmente com *n* grande). Anderson-Darling como alternativa quando o comportamento nas caudas é a preocupação central (ex.: dados financeiros, eventos extremos).

Se a normalidade falhar de forma consistente entre os critérios: volte ao Módulo 6 — Mann-Whitney no lugar do teste t de duas amostras, Wilcoxon signed-rank no lugar do teste t pareado, Kruskal-Wallis no lugar da ANOVA (seção 7). O critério de decisão completo (visual + formal + tamanho de amostra) já foi montado no Módulo 6, seção 6 — este módulo só adiciona o instrumental formal (testes numéricos) que faltava naquele momento.

---

6. Teste de homogeneidade de variâncias — pré-requisito para ANOVA

Assim como o teste t clássico assume variâncias iguais entre grupos (seção 4.2), a ANOVA de um fator (seção 7) também assume **homocedasticidade**: a variância dentro de cada grupo deve ser aproximadamente a mesma entre os grupos comparados.

Teste de Levene (mais robusto a desvios de normalidade — geralmente preferido):

```
install.packages("car")
library(car)

leveneTest(rendimento ~ regiao, data = pam)
```

Teste de Bartlett (mais poderoso quando a normalidade de fato se sustenta em cada grupo, mas mais sensível a violá-la — usar só depois de já ter verificado normalidade dentro de cada grupo via seção 5):

```
bartlett.test(rendimento ~ regiao, data = pam)
```

H_0 em ambos: as variâncias são iguais entre os grupos. Se rejeitar H_0 (variâncias diferentes), a ANOVA clássica (seção 7) fica menos confiável — alternativas incluem `oneway.test(..., var.equal = FALSE)` (versão Welch da ANOVA, análoga ao Welch do teste t) ou, se a normalidade também falhar, Kruskal-Wallis (Módulo 6).

---

7. ANOVA de um fator

7.1 A lógica: decompondo variância

ANOVA (*Analysis of Variance*) testa se **3 ou mais** grupos têm a mesma média populacional — a extensão natural do teste t de duas amostras (paralelo direto ao Kruskal-Wallis do Módulo 6, que é a versão não-paramétrica exatamente desta técnica).

$H_0 : \mu_1 = \mu_2 = \ldots = \mu_k$ $\quad$ H_1 : pelo menos uma média difere das demais

A ideia central: decompor a variabilidade total dos dados em duas partes — variabilidade **entre grupos** (o quanto as médias dos grupos diferem entre si) e variabilidade **dentro dos grupos** (o quanto os pontos variam em torno da própria média do grupo, ruído que nenhum modelo elimina):

$$SQ_{total} = SQ_{entre} + SQ_{dentro}$$

Se a variabilidade entre grupos for grande **em relação** à variabilidade dentro dos grupos, isso é evidência de que os grupos realmente diferem — não é só ruído. Essa razão, ajustada pelos graus de liberdade apropriados, é a estatística F:

$$ F = \frac{MQ_{entre}}{MQ_{dentro}} = \frac{SQ_{entre}/(k-1)}{SQ_{dentro}/(n-k)} $$

Sob H_0, F segue uma **distribuição F** com $(k-1, n-k)$ graus de liberdade. Um F grande (bem acima de 1) é evidência contra H_0 — indica que a variação entre grupos é desproporcionalmente maior que a variação natural dentro de cada grupo.

7.2 No R

```
modelo_anova <- aov(rendimento ~ regiao, data = pam)
summary(modelo_anova)
```

O output traz a tabela ANOVA clássica: graus de liberdade, soma de quadrados, quadrado médio, estatística F e p-valor — exatamente os componentes da fórmula acima, calculados automaticamente.

7.3 Exemplo com dado real: rendimento de soja por região

```
library(dplyr)

soja <- pam %>% filter(cultura == "Soja")

boxplot(rendimento ~ regiao, data = soja,
        main = "Rendimento médio de soja por região (kg/ha)",
        xlab = "Região", ylab = "Rendimento (kg/ha)")

# Premissas antes de confiar no resultado -- nunca pule esta etapa
shapiro.test(residuals(aov(rendimento ~ regiao, data = soja)))   #
normalidade dos RESÍDUOS, não da variável bruta
leveneTest(rendimento ~ regiao, data = soja)

modelo_soja <- aov(rendimento ~ regiao, data = soja)
summary(modelo_soja)
```

Ponto técnico importante: a premissa de normalidade da ANOVA é sobre os **resíduos** do modelo (`residuals(modelo)`), não sobre a variável `rendimento` bruta ignorando os grupos — testar `shapiro.test(soja$rendimento)` diretamente, sem separar por grupo, mistura a variabilidade real entre regiões (parte do sinal, não do ruído) com a variabilidade dentro dos grupos, o que pode indicar "não-normalidade" mesmo quando cada grupo individualmente é bem comportado.

Dado que rendimento agrícola varia estruturalmente entre regiões brasileiras (diferenças de solo, clima, tecnologia de produção, escala das propriedades — Centro-Oeste tende a concentrar rendimentos mais altos de soja que outras regiões, por fatores de mecanização e escala), espera-se *F* grande e p-valor baixo — evidência de que a região explica parte real da variação do rendimento, não só ruído.

---

8. Teste de Tukey (post-hoc)

Igual ao Kruskal-Wallis do Módulo 6, uma ANOVA significativa diz "existe diferença em algum lugar entre os *k* grupos", não diz **quais** pares diferem. Tukey HSD (*Honestly Significant Difference*) é o post-hoc padrão para depois de uma ANOVA — o análogo direto e paramétrico do teste de Dunn (Módulo 6, seção 4.3):

```
TukeyHSD(modelo_soja)

plot(TukeyHSD(modelo_soja))   # visualização: intervalos que não cruzam o
zero = diferença significativa
```

Tukey já aplica correção para comparações múltiplas internamente (como Dunn faz para Kruskal-Wallis) — controla a taxa de erro Tipo I **conjunta** sobre todas as comparações par a par simultaneamente, não just cada comparação isoladamente a $\alpha = 0.05$ (o mesmo problema de comparações múltiplas apontado desde o Módulo 4, seção 2, e formalizado agora). Mesma regra do Módulo 6: só interprete o post-hoc depois de confirmar que o teste global (ANOVA) foi significativo.

---

9. ANOVA de dois fatores

9.1 Quando há mais de uma variável categórica explicativa

Estende a lógica de decomposição de variância para **duas** variáveis categóricas simultaneamente, permitindo testar três hipóteses de uma vez:

- **Efeito principal do Fator A** (ex.: região importa, independente da cultura?)
- **Efeito principal do Fator B** (ex.: cultura importa, independente da região?)

- **Interação A×B** (ex.: o efeito da região sobre o rendimento é diferente dependendo da cultura? — a pergunta que nem o teste separado de A nem o de B respondem sozinhos)

$$y_{ijk} = \mu + \alpha_i + \beta_j + (\alpha\beta)_{ij} + \varepsilon_{ijk}$$

9.2 No R

```
modelo_2fatores <- aov(rendimento ~ regiao * cultura, data = pam)
summary(modelo_2fatores)
```

A sintaxe `regiao * cultura` expande automaticamente para `regiao + cultura + regiao:cultura` (mesma convenção de fórmula já usada para termos de interação em regressão, Módulo 4, seção 5.5) — três linhas na tabela ANOVA, uma para cada efeito.

9.3 Interpretando a interação primeiro

Regra prática: sempre leia a linha de interação **antes** dos efeitos principais. Se a interação for significativa, os efeitos principais isolados passam a ser difíceis de interpretar de forma direta — "o efeito da região" deixa de ser um número único, porque ele muda dependendo da cultura. Nesse caso, a leitura correta é via gráfico de interação:

```
interaction.plot(pam$regiao, pam$cultura, pam$rendimento,
                 xlab = "Região", ylab = "Rendimento médio (kg/ha)", trac
e.label = "Cultura")
```

Linhas paralelas no gráfico de interação indicam ausência de interação real (o efeito da região é consistente entre culturas); linhas que se cruzam ou divergem em inclinação indicam interação — o ranking de regiões por rendimento muda dependendo da cultura analisada.

Post-hoc em desenho de dois fatores: `TukeyHSD(modelo_2fatores)` funciona da mesma forma, mas retorna comparações para cada fator separadamente e, se solicitado, para a combinação completa fator A × fator B:

```
TukeyHSD(modelo_2fatores, which = "regiao")          # só comparações de
região
TukeyHSD(modelo_2fatores, which = "regiao:cultura")   # comparações de
cada combinação região×cultura
```

---

10. Nota conceitual: silhouette (Módulo 7) vs. Tukey (aqui) — dois mundos diferentes de "grupo"

A observação registrada no fechamento do Módulo 7 merece esta ponte, mesmo sem entrar em código: **silhouette e Tukey respondem perguntas fundamentalmente diferentes sobre "grupo".**

- **Silhouette** (não-supervisionado, Módulo 7): você **não sabe** os grupos de antemão — o cluster os descobre, e silhouette avalia se essa descoberta faz sentido geometricamente (cada ponto está mais perto do seu próprio cluster do que do cluster vizinho?). Não há H_0, não há p-valor — é uma métrica de qualidade de agrupamento, não um teste de hipótese.
- **ANOVA + Tukey** (supervisionado, este módulo): os grupos **já são conhecidos e definidos de antemão** por uma variável categórica real (região, cultura, tratamento) — a pergunta não é "esses grupos fazem sentido geometricamente", é "a diferença de média observada entre grupos já definidos é grande o bastante para não ser só ruído amostral".

Não existe uma técnica que substitua a outra — elas resolvem problemas de naturezas diferentes (descobrir estrutura vs. testar diferença entre estrutura já conhecida). A única ponte real entre os dois mundos: depois de rodar um cluster (Módulo 7) e nomear os clusters resultantes como uma nova variável categórica, nada impede de usar ANOVA/Tukey **sobre os clusters descobertos** para testar se outras variáveis (não usadas no cluster original)

diferem significativamente entre os clusters encontrados — nesse momento, e só nesse momento, os dois mundos se encontram tecnicamente.

---

11. Qui-quadrado e Teste Exato de Fisher: associação entre categóricas

Todo teste até aqui compara **médias** entre grupos (t, ANOVA). Quando as duas variáveis de interesse são **ambas categóricas** — não "rendimento por região" mas "produtividade alta/baixa por região", ou em dado de saúde/censo, "desfecho por categoria de exposição" —, a pergunta muda de "as médias diferem" para "a distribuição de uma variável categórica depende da outra". O parceiro natural do teste t para esse cenário é o **qui-quadrado de independência**.

11.1 Tabela de contingência e a lógica do teste

H_0: as duas variáveis categóricas são **independentes** (a distribuição de uma não muda conforme o nível da outra). A estatística compara frequências **observadas** contra frequências **esperadas sob independência**:

$$ \chi^2 = \sum \frac{(O_i - E_i)^2}{E_i} $$

Onde E_i (frequência esperada de cada célula, sob H_0) é calculada a partir das margens da tabela: $E_{ij} = \dfrac{(\text{total da linha } i)(\text{total da coluna } j)}{n}$. Quanto mais as frequências observadas se afastam do que a independência preveria, maior χ^2, mais evidência contra H_0.

```
library(dplyr)

pam <- read.csv("pam_municipal.csv", stringsAsFactors = TRUE)   # já com
regiao mapeada, seção 15

soja <- pam %>% filter(cultura == "Soja")

# Categoriza rendimento em alta/baixa produtividade (acima/abaixo da
mediana nacional de soja)
soja$produtividade <- ifelse(soja$rendimento > median(soja$rendimento), "A
lta", "Baixa")

tabela_contingencia <- table(soja$regiao, soja$produtividade)
tabela_contingencia
```

11.2 Qui-quadrado de Pearson (tabelas RxC)

```
teste_qui <- chisq.test(tabela_contingencia)
teste_qui

teste_qui$expected   # frequências esperadas -- confira se alguma célula
tem E < 5 (seção 11.4)
```

H_0 rejeitada (p-valor baixo) indica que produtividade alta/baixa **não** está distribuída igualmente entre as regiões — há associação entre região e faixa de produtividade. Isso não diz *qual* região puxa o resultado (seção 11.3 resolve isso), só que a tabela como um todo se afasta de independência.

11.3 Resíduos padronizados: onde está a associação

O χ^2 global é análogo ao teste F da ANOVA (seção 7) — significativo diz "existe efeito em algum lugar da tabela", não onde. Resíduos padronizados de Pearson (`$stdres`) apontam a célula específica:

```
teste_qui$stdres
```

Célula com resíduo padronizado $|z| > 2$ (aproximadamente, mesma lógica de z-crítico do Módulo 8) contribui desproporcionalmente para o χ^2 — uma região com resíduo positivo grande na coluna "Alta" tem **mais** municípios de alta produtividade do que a independência prediria; resíduo negativo grande indica o oposto.

11.4 Teste Exato de Fisher: quando a amostra é pequena demais

O qui-quadrado de Pearson é uma **aproximação** que depende de frequências esperadas não muito pequenas — regra prática comum: nenhuma célula com $E_i < 5$. Quando essa condição falha (tabelas 2×2 com contagens baixas, comum em subgrupos raros ou amostras pequenas), o qui-quadrado fica impreciso e o Teste Exato de Fisher — que calcula a probabilidade exata da tabela observada (e tabelas mais extremas) via distribuição hipergeométrica, sem aproximação — é a alternativa correta:

```
# Exemplo com subamostra pequena, forçando E < 5 em alguma célula
sub_pequena <- soja %>% filter(regiao %in% c("Norte", "Nordeste")) %>% sli
ce_sample(n = 15)
tabela_pequena <- table(sub_pequena$regiao, sub_pequena$produtividade)

chisq.test(tabela_pequena)$expected   # confira células com E < 5
fisher.test(tabela_pequena)           # alternativa exata, sem essa
restrição
```

Correção de Yates (aplicada por padrão em `chisq.test()` para tabelas 2×2, `correct = TRUE`) é uma correção de continuidade — reduz o χ^2 ligeiramente para compensar o fato de a distribuição discreta estar sendo aproximada por uma distribuição contínua; ela reduz o risco de Erro Tipo I (rejeitar H_0 com facilidade demais) em tabelas 2×2 pequenas, mas ainda assim Fisher é preferível quando $E_i < 5$ em qualquer célula, não só nas 2×2.

---

12. Tamanho de efeito (effect size): a peça que o p-valor sozinho não dá

Já visto informalmente desde o Módulo 4 (R^2 não mede tamanho de efeito) e mencionado como convenção de leitura de *r* (Módulo 4, seção 1.1) — aqui a formalização completa, uma vez por família de teste, para fechar o alerta repetido ao longo do plano: **significância estatística não é magnitude prática**. Um p-valor baixo com *n* grande não diz se a diferença importa; effect size diz.

12.1 Cohen's d — diferença de médias (teste t)

$$ d = \frac{\bar{x}_1 - \bar{x}_2}{s_{\text{pooled}}} $$

Onde s_{pooled} é o desvio padrão combinado dos dois grupos. *d* expressa a diferença de médias em **unidades de desvio padrão** — comparável entre variáveis de escalas diferentes, o mesmo princípio de normalização já visto no CV (Módulo 2) e no *z*-score (Módulo 7).

```
install.packages("effsize")
library(effsize)

cohen.d(rendimento ~ regiao, data = soja %>% filter(regiao %in% c("Sul", "
Centro-Oeste")))
```

Convenção de Cohen (1988), mesma cautela já dada para |*r*| no Módulo 4 — referência de linguagem, não lei: $d \approx 0.2$ pequeno, $d \approx 0.5$ médio, $d \approx 0.8$ grande.

12.2 Eta-squared e Omega-squared — ANOVA

$$ \eta^2 = \frac{SQ_{\text{entre}}}{SQ_{\text{total}}} $$

Proporção da variância total explicada pelo fator categórico — leitura direta análoga ao R^2 da regressão (Módulo 4, seção 4.1), aplicada à decomposição de variância da ANOVA (seção 7.1).

```
install.packages("effectsize")
library(effectsize)

eta_squared(modelo_soja, partial = FALSE)
omega_squared(modelo_soja)
```

η^2 tem o mesmo viés que R^2 comum (seção do Módulo 4 sobre R^2 ajustado): tende a superestimar a variância explicada na população, especialmente com *n* pequeno ou muitos grupos. **Omega-squared** (ω^2) corrige esse viés — analogamente ao papel do R^2 ajustado — e é geralmente preferido para relatar magnitude de efeito em ANOVA de forma mais conservadora.

12.3 Odds ratio — qui-quadrado e regressão logística

Para tabelas 2×2 especificamente, odds ratio (já formalizado no Módulo 5, seção 5) é o tamanho de efeito natural — quantifica a força da associação em vez de só confirmar que ela existe:

```
# Reaproveitando tabela_pequena (seção 11.4) se for 2x2, ou construindo
uma tabela 2x2 direta
tabela_2x2 <- table(soja$regiao %in% c("Centro-Oeste"), soja$produtividade
)
fisher.test(tabela_2x2)   # já retorna odds ratio estimado e seu IC, para
tabelas 2x2
```

12.4 Por que reportar sempre os dois números juntos

Um resultado como “p < 0.001, d = 0.15” e outro “p = 0.04, d = 0.9” contam histórias opostas do que o p-valor sozinho sugere: o primeiro é “estatisticamente muito significativo, mas o efeito é pequeno” (comum com *n* grande — o mesmo alerta de correlação trivial mas significativa do Módulo 4, seção 1.4); o segundo é “efeito grande, evidência estatística mais

modesta" (comum com *n* pequeno). Reportar só o p-valor esconde essa distinção — a prática correta, em qualquer teste deste módulo, é sempre reportar p-valor **e** tamanho de efeito lado a lado, nunca um sem o outro.

---

13. Bootstrap e permutação: alternativa quando as premissas falham

Nota curta — não um mergulho profundo, mas o registro de uma ferramenta que preenche a lacuna entre "via paramétrica" (este módulo) e "via não-paramétrica clássica" (Módulo 6, testes de posto): reamostragem computacional.

13.1 Bootstrap: IC sem depender de TLC

Quando a distribuição amostral de uma estatística é difícil de derivar analiticamente (ou o TLC do Módulo 8 é questionável para o *n* disponível), bootstrap constrói um IC empírico **reamostrando os próprios dados observados, com reposição**, milhares de vezes:

```
install.packages("boot")
library(boot)

estatistica_media <- function(dados, indices) mean(dados[indices])

resultado_boot <- boot(soja$rendimento, statistic = estatistica_media, R
= 2000)
boot.ci(resultado_boot, type = "perc")   # IC percentil bootstrap
```

A lógica: cada reamostragem com reposição do mesmo tamanho *n* simula "e se eu tivesse coletado uma amostra ligeiramente diferente" — a variabilidade entre as 2000 médias recalculadas aproxima empiricamente a variabilidade que a fórmula analítica do Módulo 8 (seção 9) tentaria capturar via TLC, sem precisar assumir normalidade da distribuição amostral.

13.2 Teste de permutação: alternativa ao teste t quando premissas falham

Em vez de comparar contra uma distribuição teórica (t de Student), o teste de permutação constrói a distribuição de H_0 **embaralhando os rótulos de grupo** milhares de vezes (sob H_0, a atribuição de grupo é arbitrária) e vendo onde a diferença observada cai nessa distribuição simulada:

```
diferenca_observada <- diff(tapply(soja$rendimento[soja$regiao %in% c("Sul
", "Centro-Oeste")],
                                   droplevels(soja$regiao[soja$regiao %i
n% c("Sul", "Centro-Oeste")]),
                                   mean))

set.seed(2026)
sub <- soja %>% filter(regiao %in% c("Sul", "Centro-Oeste"))
diferencas_permutadas <- replicate(2000, {
  regiao_embaralhada <- sample(sub$regiao)
  diff(tapply(sub$rendimento, regiao_embaralhada, mean))
})

p_valor_permutacao <- mean(abs(diferencas_permutadas) >= abs(diferenca_obs
ervada))
p_valor_permutacao
```

Sem assumir normalidade nem variâncias iguais — a única suposição é que, sob H_0, trocar os rótulos de grupo não deveria mudar sistematicamente a diferença observada. Útil especificamente quando o Exercício 1 (assimetria/curtose) e a seção 6 (homogeneidade de variância) sinalizam violação simultânea de premissas que nem Welch nem Kruskal-Wallis resolvem completamente.

---

14. Diferenças-chave vindas de Python

Conceito	Python (scipy.stats / statsmodels)	R
Teste t (uma amostra)	`scipy.stats.ttest_1samp(x, mu0)`	`t.test(x, mu = mu`
Teste t (duas amostras)	`scipy.stats.ttest_ind(x, y, equal_var=False)`	`t.test(x, y)` (We padrão)
Teste t pareado	`scipy.stats.ttest_rel(x, y)`	`t.test(x, y, pair TRUE)`
Shapiro-Wilk	`scipy.stats.shapiro(x)`	`shapiro.test(x)`
Kolmogorov-Smirnov	`scipy.stats.kstest(x, 'norm', args=(mean, std))`	`ks.test(x, "pnorm sd)`
Anderson-Darling	`scipy.stats.anderson(x, dist='norm')`	`nortest::ad.test(`
Levene	`scipy.stats.levene(*grupos)`	`car::leveneTest(y`
Bartlett	`scipy.stats.bartlett(*grupos)`	`bartlett.test(y ~`
ANOVA de um fator	`scipy.stats.f_oneway(*grupos)` (só a estatística F/p) ou `statsmodels.formula.api.ols(...).fit()` + `anova_lm` (tabela completa)	`aov(y ~ grupo)` + — sempre a tabela c
Tukey HSD	`statsmodels.stats.multicomp.pairwise_tukeyhsd(...)`	`TukeyHSD(modelo_a`
ANOVA de dois fatores	`statsmodels`: `ols('y ~ C(A) * C(B)', data=df).fit()` + `anova_lm(..., typ=2)`	`aov(y ~ A * B, da`
Assimetria/ curtose	`scipy.stats.skew(x)` / `scipy.stats.kurtosis(x)`	`e1071::skewness(x` `e1071::kurtosis(x`

Conceito	Python (scipy.stats / statsmodels)	R
Qui-quadrado de independência	`scipy.stats.chi2_contingency(tabela)`	`chisq.test(tabela`
Teste Exato de Fisher	`scipy.stats.fisher_exact(tabela)` (só 2×2)	`fisher.test(tabel` RxC)
Cohen's d	Não tem função nativa — cálculo manual comum	`effsize::cohen.d(`
Eta-squared / Omega-squared	`pingouin.anova(..., effsize='n2')` (pacote `pingouin`, fora da tríade scipy/statsmodels/sklearn)	`effectsize::eta_s` `omega_squared()`
Bootstrap	`scipy.stats.bootstrap`	`boot::boot()` + `b`
Teste de permutação	`scipy.stats.permutation_test`	Sem pacote único d implementação man `sample()` / `replica` como na seção 13.2

Padrão recorrente desde os primeiros módulos: `scipy.stats.f_oneway` entrega só *F* e p-valor — para a tabela ANOVA completa (somas de quadrados, graus de liberdade) em Python, é preciso migrar para `statsmodels` com sintaxe de fórmula (`ols` + `anova_lm`), que é, não por acaso, a sintaxe mais parecida com o `aov(y ~ x)` do R. É um bom sinal de que, quanto mais uma análise em Python se aproxima de inferência formal (em vez de só um número resumo), mais o código acaba convergindo para a mesma filosofia de fórmula estatística que o R usa nativamente desde sempre.

---

15. Fonte de dados: Produção Agrícola Municipal (IBGE)

Por que este dataset: cobre rendimento médio (kg/ha) de lavouras por município, com região e cultura como variáveis categóricas naturais e reais — exatamente o tipo de estrutura que o plano original pedia para este módulo ("safras por região"), e permite tanto ANOVA de um fator (região) quanto de dois fatores (região × cultura) sem precisar de uma segunda fonte de dados.

Como obter

Opção 1 — Portal SIDRA: https://sidra.ibge.gov.br/pesquisa/pam/tabelas

1. Acesse a tabela **1612** (lavouras temporárias — cobre soja e milho).
2. Selecione a variável **"Rendimento médio da produção"** (kg/ha), o produto (**Soja em grão**, depois repita para **Milho em grão** se for montar o exercício de dois fatores), nível geográfico **Município** (todos) e o ano mais recente disponível.
3. Exporte em `.csv`.

```
library(dplyr)

pam_bruto <- read.csv("tabela_1612_pam.csv", skip = 4, fileEncoding = "Lat
in1")
names(pam_bruto)   # confira sempre -- mesma ressalva de todo dado real
deste projeto

# Mapeamento UF -> Região (a exportação do SIDRA normalmente traz UF, não
Região direto)
mapa_regiao <- c(
  AC="Norte", AP="Norte", AM="Norte", PA="Norte", RO="Norte", RR="Norte",
TO="Norte",
  AL="Nordeste", BA="Nordeste", CE="Nordeste", MA="Nordeste", PB="Nordeste
",
  PE="Nordeste", PI="Nordeste", RN="Nordeste", SE="Nordeste",
  DF="Centro-Oeste", GO="Centro-Oeste", MT="Centro-Oeste", MS="Centro-
Oeste",
  ES="Sudeste", MG="Sudeste", RJ="Sudeste", SP="Sudeste",
  PR="Sul", RS="Sul", SC="Sul"
)

pam <- pam_bruto %>%
  rename(municipio = 1, uf = matches("UF|Unidade"), rendimento = matches("
Rendimento")) %>%
  mutate(
    rendimento = as.numeric(rendimento),
    regiao = mapa_regiao[uf]
  ) %>%
  filter(!is.na(rendimento), rendimento > 0)
```

Opção 2 — Base dos Dados (já tratada, com região já disponível como coluna): https://basedosdados.org/dataset/fc403b40-a7e1-40e7-9efe-910847b45a69

Evita o mapeamento manual UF→região acima — vantagem real de conveniência se você já usa BigQuery/SQL para os outros datasets do projeto.

---

16. Exercícios

Exercício 1 — Diagnóstico completo de normalidade

Para `rendimento` de soja (todos os municípios, ignorando região por enquanto), calcule assimetria e curtose (`e1071`), gere histograma e Q-Q plot, e rode Shapiro-Wilk (em amostra de até 5000 pontos, se a base for maior) e Anderson-Darling. Os diagnósticos concordam entre si? O resultado é esperado dado o tipo de variável (rendimento agrícola, positivo, tipicamente assimétrico)?

Solução:

```
library(e1071)
library(nortest)

skewness(soja$rendimento)
kurtosis(soja$rendimento)

hist(soja$rendimento, main = "Rendimento de soja (kg/ha)")
qqnorm(soja$rendimento); qqline(soja$rendimento, col = "firebrick")

amostra_shapiro <- if (nrow(soja) > 5000) sample(soja$rendimento, 5000) el
se soja$rendimento
shapiro.test(amostra_shapiro)
ad.test(soja$rendimento)
```

Rendimento agrícola costuma ter assimetria positiva moderada (limitado inferiormente por zero, sem limite superior rígido, com alguns municípios de rendimento excepcionalmente alto puxando a cauda) — espera-se `skewness > 0` , rejeição de normalidade nos testes formais, consistente com a leitura visual do histograma e do Q-Q plot desviando nas caudas.

Exercício 2 — Homogeneidade de variância entre regiões

Rode Levene e Bartlett para `rendimento ~ regiao` em `soja` . Os dois concordam? Se as variâncias forem heterogêneas, o que isso implica para a confiabilidade da ANOVA clássica da seção 7.3?

Solução:

```
library(car)

leveneTest(rendimento ~ regiao, data = soja)
bartlett.test(rendimento ~ regiao, data = soja)
```

Regiões com pacotes tecnológicos e escala de produção muito diferentes (Centro-Oeste com grandes propriedades mecanizadas vs. regiões com predominância de agricultura familiar) tendem a mostrar variância de rendimento bem diferente entre si — se Levene/Bartlett rejeitarem homogeneidade, a ANOVA clássica (que assume variâncias iguais) fica menos confiável; considere `oneway.test(rendimento ~ regiao, data = soja, var.equal = FALSE)` (Welch-ANOVA) como alternativa mais robusta.

Exercício 3 — ANOVA e Tukey completos

Rode a ANOVA de um fator (rendimento de soja por região) e, se significativa, o Tukey HSD. Quais pares de região diferem significativamente? O resultado bate com o que se esperaria da geografia real da produção de soja no Brasil?

Solução: ver seções 7.3 e 8 para o código-base. Espera-se que o Centro-Oeste apareça com rendimento médio significativamente maior que a maioria das outras regiões nos pares do Tukey — consistente com o papel histórico do Centro-Oeste como fronteira agrícola de alta produtividade em soja no Brasil.

Exercício 4 — ANOVA de dois fatores com interação

Combine dados de soja e milho num único data frame (`cultura` como segunda variável categórica) e rode `aov(rendimento ~ regiao * cultura, data = pam)` . A interação região×cultura é significativa? Gere o `interaction.plot()` e descreva, em uma frase, o padrão visual encontrado.

Solução: ver seção 9.2-9.3 para o código-base. Se a interação for significativa, o gráfico deve mostrar linhas não-paralelas entre as duas culturas ao longo das regiões — indicando que a vantagem relativa de uma região sobre outra não é a mesma para soja e para milho (ex.: uma região pode liderar em rendimento de soja mas não se destacar da mesma forma em milho).

Exercício 5 — Comparando com o Módulo 6: ANOVA vs. Kruskal-Wallis

Rode `kruskal.test(rendimento ~ regiao, data = soja)` (Módulo 6) ao lado da ANOVA da seção 7.3. As duas conclusões (rejeitar/não rejeitar) coincidem? Dado o resultado do diagnóstico de normalidade do Exercício 1, qual dos dois testes você reportaria com mais confiança num relatório real?

Solução:

```
kruskal.test(rendimento ~ regiao, data = soja)
summary(aov(rendimento ~ regiao, data = soja))
```

Com *n* grande por grupo (muitos municípios por região), o TLC (Módulo 8) tende a tornar a ANOVA razoavelmente robusta mesmo com alguma não-normalidade nos dados brutos — mas se o Exercício 1 mostrou assimetria/curtose acentuadas E os grupos tiverem tamanhos desbalanceados entre regiões, Kruskal-Wallis é a escolha mais defensável para reportar como resultado principal, com a ANOVA como checagem complementar.

Exercício 6 (desafio) — Erro Tipo II na prática: poder do teste com subamostras

Tire subamostras cada vez menores de `soja` (ex.: 10%, 30%, 60%, 100% dos municípios de cada região, mantendo todas as regiões representadas) e rode a ANOVA em cada subamostra. O p-valor se mantém significativo em todos os tamanhos, ou a significância "desaparece" nas subamostras

menores mesmo a diferença real entre regiões (presumivelmente) continuando a existir? O que isso demonstra sobre a relação entre tamanho de amostra e Erro Tipo II?

Solução:

```
testar_com_fracao <- function(fracao) {
  sub <- soja %>% group_by(regiao) %>% slice_sample(prop = fracao) %>% ung
roup()
  summary(aov(rendimento ~ regiao, data = sub))[[1]][["Pr(>F)"]][1]
}

sapply(c(0.10, 0.30, 0.60, 1.00), testar_com_fracao)
```

Espera-se que o p-valor suba (fique menos significativo, podendo até deixar de rejeitar H_0) nas frações menores — não porque a diferença real entre regiões deixou de existir, mas porque a amostra reduzida perdeu poder estatístico para detectá-la. Essa é a demonstração prática do Erro Tipo II da seção 2: um teste "não significativo" numa amostra pequena não é evidência de que não há efeito, pode ser só falta de poder para enxergá-lo.

Exercício 7 — Qui-quadrado e resíduos padronizados

Categorize `rendimento` de `soja` em "Alta"/"Baixa" produtividade pela mediana (seção 11.1), rode `chisq.test()` para `regiao × produtividade`, e identifique, via resíduos padronizados, qual região mais contribui para a associação e em qual direção (mais ou menos produtividade alta do que a independência preveria).

Solução:

```
soja$produtividade <- ifelse(soja$rendimento > median(soja$rendimento), "A
lta", "Baixa")
tabela <- table(soja$regiao, soja$produtividade)

teste_qui <- chisq.test(tabela)
teste_qui
teste_qui$stdres
```

Dado o padrão de concentração de produtividade de soja já discutido nos exercícios anteriores (Centro-Oeste com vantagem histórica), espera-se resíduo padronizado positivo e destacado na célula Centro-Oeste × Alta (mais municípios de alta produtividade do que a independência prediria) e, espelhado, resíduo negativo em Centro-Oeste × Baixa. O χ^2 global confirma que existe associação; os resíduos localizam onde.

Exercício 8 — Effect size ao lado do p-valor, sempre

Para a comparação Sul vs. Centro-Oeste (mesmo par do Exercício 6 do Módulo 8 e da seção 12.1), reporte lado a lado: p-valor do `t.test()`, Cohen's d, e a diferença de médias em unidade original (kg/ha). Escreva uma frase de interpretação que use os três números — não só o p-valor.

Solução:

```
library(effsize)

duas_regioes <- soja %>% filter(regiao %in% c("Sul", "Centro-Oeste"))

t.test(rendimento ~ regiao, data = duas_regioes)$p.value
cohen.d(rendimento ~ regiao, data = duas_regioes)
diff(tapply(duas_regioes$rendimento, duas_regioes$regiao, mean))
```

Frase-modelo esperada: "a diferença de rendimento entre Centro-Oeste e Sul é estatisticamente significativa (p < ...) e de magnitude [pequena/média/grande, conforme o d observado], correspondendo a uma diferença média de [X] kg/ha." Reportar só o p-valor deixaria de fora se a diferença, apesar de real, é grande o bastante para importar numa decisão prática (ex.: viabilidade de expansão de área em uma região vs. outra).

Exercício 9 — Permutação como checagem cruzada do teste t

Repita o teste de permutação da seção 13.2 para o mesmo par Sul vs. Centro-Oeste. O p-valor de permutação fica próximo do p-valor do `t.test()` clássico? Em que situação (pensando nas premissas do teste t) você esperaria que os dois divergissem mais?

Solução: ver código-base da seção 13.2, substituindo `sub` pelos dados de Sul/Centro-Oeste. Com *n* grande por grupo e sem violação severa de normalidade (mesmo diagnóstico do Exercício 1 aplicado a essas duas regiões), os dois p-valores devem ficar próximos — o teste t já é razoavelmente robusto nessas condições (TLC, Módulo 8). A divergência tende a crescer quando a normalidade é violada de forma severa (assimetria/curtose extremas) e/ou *n* é pequeno — exatamente o cenário em que a via não-paramétrica (Kruskal-Wallis, Módulo 6) ou a permutação passam a ser a escolha mais defensável, e o teste t clássico começa a reportar um p-valor que não reflete corretamente a incerteza real.

---

17. Entrega esperada do módulo

Script `hipoteses_pam.R` que:

1. Carrega e limpa o dataset PAM (soja e milho, com região mapeada)
2. Roda diagnóstico completo de normalidade (assimetria/curtose, visual, Shapiro-Wilk, Anderson-Darling) sobre o rendimento por grupo
3. Roda teste de homogeneidade de variância (Levene) entre regiões
4. Roda ANOVA de um fator (região) com Tukey HSD, e ANOVA de dois fatores (região × cultura) com gráfico de interação
5. Compara o resultado da ANOVA com Kruskal-Wallis (Módulo 6) sobre os mesmos dados
6. Categoriza rendimento em alta/baixa produtividade e roda qui-quadrado + resíduos padronizados para região × produtividade; roda Fisher em pelo menos uma subtabela pequena
7. Reporta Cohen's d (ou eta²/omega² para a ANOVA) ao lado de cada p-valor, nunca isoladamente
8. Roda pelo menos um IC bootstrap e um teste de permutação, comparando com o resultado paramétrico equivalente

---

18. Critério de domínio antes de avançar ao Módulo 10

- ☐ Explica a lógica de prova por contradição de um teste de hipótese, e por que "não rejeitar H_0" não é o mesmo que "provar H_0"
- ☐ Distingue Erro Tipo I de Erro Tipo II sem confundir qual cada um controla, e explica poder do teste em termos de β
- ☐ Interpreta corretamente um p-valor (probabilidade do dado dado H_0, não o inverso) e cita pelo menos dois erros comuns de interpretação
- ☐ Sabe escolher entre teste t de Welch e teste t clássico (pooled), e por que Welch é o padrão mais seguro
- ☐ Consegue rodar e interpretar Shapiro-Wilk, sabendo especificamente por que ele fica pouco confiável com n muito grande
- ☐ Entende por que a premissa de normalidade da ANOVA é sobre os resíduos, não sobre a variável bruta ignorando os grupos
- ☐ Sabe explicar a lógica de decomposição de variância por trás da estatística F (entre vs. dentro dos grupos)
- ☐ Sabe quando e por que rodar Tukey só depois de uma ANOVA global significativa
- ☐ Consegue interpretar uma interação significativa numa ANOVA de dois fatores, e sabe que isso muda a leitura dos efeitos principais isolados
- ☐ Sabe rodar e interpretar qui-quadrado de independência, ler resíduos padronizados para localizar a associação, e sabe quando trocar por Fisher exato
- ☐

Reporta tamanho de efeito (Cohen's d, eta²/omega², odds ratio) ao lado de todo p-valor, e explica por que os dois números juntos contam uma história que nenhum sozinho conta

- ☐ Entende a lógica de bootstrap (reamostragem com reposição) e de teste de permutação (embaralhamento de rótulos), e sabe quando cada um é preferível à via paramétrica clássica
- ☐ Todos os 9 exercícios rodam sem erro, com interpretação escrita conectando cada resultado ao conceito teórico correspondente (não só "o p-valor deu baixo")

---

19. Nota: pendências fechadas e decisões deste módulo

1. **Assimetria e curtose numéricas, pendentes desde o Módulo 2, fecham aqui** (seção 5.1) — como diagnóstico complementar aos testes formais de normalidade, no lugar certo: junto com Shapiro-Wilk, KS e Anderson-Darling, não isolado como um adendo solto.
2. **Silhouette score (Módulo 7) não entrou tecnicamente neste módulo** — não havia lugar natural para a técnica em si (ela pertence ao mundo não-supervisionado, este módulo é todo supervisionado/grupos conhecidos). A seção 10 fecha essa pendência conceitualmente, sem forçar código onde ele não pertence.
3. **Qui-quadrado/Fisher (seção 11), effect size formal (seção 12) e bootstrap/permutação (seção 13)** foram incorporados nesta revisão, motivados por retorno externo de revisão de conteúdo — todas as seções a partir da 11 foram renumeradas (a partir do antigo "Diferenças-chave vindas de Python") para acomodar o conteúdo novo. Sem pendência.
4. **Bootstrap e permutação (seção 13) ficaram deliberadamente em nível de nota, não de seção completa com exercícios próprios em**

profundidade — a decisão de escopo aprovada foi essa (prioridade baixa/opcional no plano de aprimoramento). O Exercício 9 dá prática mínima; um mergulho mais fundo (bootstrap para outras estatísticas além da média, permutação para ANOVA multi-grupo) fica fora deste ciclo.

PARTE 4

Aplicações

Duas famílias de dado com estrutura própria — séries temporais e dados espaciais — tratadas com o ferramental estatístico maduro do ecossistema R.

MÓDULO 10 · APLICAÇÕES

Séries Temporais

1. Estrutura de dados de série temporal em R

1.1 `ts` — a estrutura clássica (e suas limitações)

```
serie_ts <- ts(valores, start = c(2019, 1), frequency = 12)  # mensal,
começa jan/2019
```

`ts` guarda só um vetor numérico + metadado de frequência/início — não guarda datas de verdade, só a posição. Funciona bem para uma série única e regular, mas quebra em dois cenários comuns: (1) painel — múltiplas séries (uma por cidade, por exemplo) exige uma lista de objetos `ts` ou um loop manual; (2) datas irregulares ou com furos (dias úteis, feriados) não têm representação nativa boa.

1.2 `tsibble` — o padrão tidyverts

```
library(tsibble)
library(dplyr)

painel <- read.csv("painel_regional_varejo.csv") %>%
  mutate(mes_ano = yearmonth(mes_ano))   # tsibble usa tipos de data/mês
nativos, não string

painel_tsb <- as_tsibble(painel, index = mes_ano, key = cidade_id)
painel_tsb
```

Dois conceitos centrais que não existem em `ts`:

- **`index`**: a coluna de tempo (aceita `Date`, `yearmonth`, `yearquarter`, entre outros — tipos que carregam a granularidade certa, diferente de guardar tudo como `Date` genérico).

- `key`: a(s) coluna(s) que identificam séries distintas dentro da mesma tabela. Com `key = cidade_id`, o `tsibble` sabe que existem 50 séries independentes, uma por cidade — equivalente conceitual de um `MultiIndex` (tempo, entidade) em pandas, mas com a garantia estrutural de que cada combinação chave×tempo aparece no máximo uma vez.

`tsibble` valida essa unicidade na criação — se houver duplicata (mesma cidade, mesmo mês, duas linhas), `as_tsibble()` lança erro em vez de aceitar silenciosamente, o mesmo espírito de "falhar ruidosamente" já visto no Módulo 6 com nomes de coluna.

```
has_gaps(painel_tsb)          # verifica furos na série (meses faltando)
por cidade
scan_gaps(painel_tsb)         # lista exatamente onde estão, se houver
```

Furos importam porque ACF, decomposição e ARIMA assumem espaçamento regular — um mês faltando silenciosamente desloca todos os lags seguintes sem gerar erro, um bug silencioso clássico.

---

2. Decomposição

2.1 Aditiva vs. multiplicativa

Aditiva: $Y_t = T_t + S_t + R_t$ Multiplicativa: $Y_t = T_t \times S_t \times R_t$

Onde T_t é tendência, S_t é sazonalidade, R_t é resíduo (ruído). Critério de escolha, sempre visual antes de qualquer regra: se a **amplitude** da sazonalidade cresce proporcionalmente ao nível da série (picos de dezembro cada vez maiores em valor absoluto conforme a série cresce), é multiplicativa; se a amplitude sazonal se mantém constante em valor absoluto independente do nível, é aditiva. Uma multiplicativa pode sempre

ser tratada como aditiva em log: $\log(Y_t) = \log(T_t) + \log(S_t) + \log(R_t)$ — truque padrão quando a série é estritamente positiva (preço de commodity, por exemplo, nunca é zero ou negativo).

2.2 Decomposição clássica vs. STL

```
# Clássica (base R, precisa de objeto ts)
serie_ts <- ts(vendas_cidade, frequency = 12)
plot(decompose(serie_ts, type = "additive"))

# STL (Seasonal-Trend decomposition using Loess) -- tidyverts, mais
robusto
library(feasts)
library(fabletools)

painel_tsb %>%
  filter(cidade_id == "CID01") %>%
  model(STL(indice_vendas ~ trend(window = 13) + season(window = "periodic
")) %>%
  components() %>%
  autoplot()
```

`decompose()` clássico usa média móvel simples para extrair tendência e assume sazonalidade **constante** ano a ano (a mesma forma sazonal se repete sem variar) — rígido e sensível a outliers na ponta da série (as primeiras/últimas observações perdem metade da janela da média móvel). `STL` usa suavização Loess, permite que a sazonalidade **mude lentamente ao longo do tempo** (`season(window = "periodic")` força constante, um número finito como `season(window = 13)` permite variação), e é robusto a outliers por construção. Para qualquer série real, prefira STL; `decompose()` serve mais como introdução conceitual.

2.3 Aplicação no painel sintético — verificando contra o gabarito conhecido

O painel foi construído com sazonalidade anual multiplicativa (pico nov/dez, padrão de varejo) e tendência de crescimento mensal de aproximadamente 0.15% — ver `gerar_datasets_sinteticos.R`. Isso dá o mesmo tipo de

vantagem pedagógica do censo da PPM no Módulo 8: você sabe a resposta certa antes de decompor, então pode verificar se a decomposição está recuperando a estrutura real.

```
componentes <- painel_tsb %>%
  filter(cidade_id == "CID01") %>%
  model(STL(indice_vendas ~ trend(window = 13) + season(window = "periodic
"))) %>%
  components()

# a tendência extraída deve ser crescente e suave
componentes %>% autoplot(trend)

# o componente sazonal deve mostrar pico por volta de nov/dez -- confira a
fase
componentes %>% autoplot(season_year)
```

Se o pico sazonal extraído não cair perto de novembro/dezembro, ou se a tendência mostrar serrilhado forte em vez de suavidade, é sinal de janela de suavização mal ajustada (`window` pequeno demais captura ruído como se fosse sinal) — outra checagem que só é possível por conhecer o gabarito.

---

3. Estacionariedade

3.1 Por que ARIMA exige isso

Um processo é **estacionário (fracamente)** quando média, variância e autocovariância não dependem de t — a série "parece a mesma" estatisticamente em qualquer janela de tempo que você olhar. ARIMA (seção 5) modela a estrutura de autocorrelação assumindo esse comportamento constante; rodar ARIMA numa série não estacionária (tendência crescente, variância explodindo) produz coeficientes que não têm interpretação estável e previsão que diverge rapidamente da realidade — o motivo do "I" (integrado) existir no nome do modelo: ele modela explicitamente a **diferenciação necessária** para alcançar estacionariedade antes de ajustar a parte ARMA.

3.2 Teste de Dickey-Fuller Aumentado (ADF)

- H_0: a série tem raiz unitária (não é estacionária — comportamento tipo passeio aleatório).
- H_1: a série é estacionária.

```
library(tseries)

adf.test(componentes$indice_vendas)   # aplicado à série bruta (não aos componentes STL)
```

Ponto de atenção, mesma lógica do Shapiro-Wilk no Módulo 9: ADF tem baixo poder estatístico em amostras curtas — pode falhar em rejeitar H_0 mesmo quando a série é, na prática, razoavelmente estacionária. Sempre cruze com inspeção visual (a série tem tendência óbvia? a variância muda claramente ao longo do tempo?) antes de decidir só pelo p-valor.

Complemento — teste KPSS, com hipóteses **invertidas** em relação ao ADF (mesma lógica de "dois testes que se complementam" do Módulo 9 com Shapiro-Wilk + Anderson-Darling, mas aqui o ganho é maior porque as hipóteses nulas são opostas):

- H_0 do KPSS: a série **é** estacionária.

```
library(feasts)

painel_tsb %>%
  filter(cidade_id == "CID01") %>%
  features(indice_vendas, unitroot_kpss)
```

Rodar os dois lado a lado evita o erro de "não rejeitei ADF, logo a série é estacionária" — se ADF não rejeita (sugerindo não-estacionária) **e** KPSS rejeita (também sugerindo não-estacionária), a evidência é consistente. Se os dois divergem, a série está numa zona ambígua e diferenciação leve costuma ser a escolha mais segura.

3.3 Diferenciação

$Y_t' = Y_t - Y_{t-1}$ (diferenciação de ordem 1 – o "d=1" do ARIMA)

```
# quantas diferenciações são necessárias para alcançar estacionariedade --
automatizado
painel_tsb %>%
  filter(cidade_id == "CID01") %>%
  features(indice_vendas, unitroot_ndiffs)

# aplicar manualmente, para inspecionar o efeito
componentes$diff_1 <- c(NA, diff(componentes$indice_vendas))
```

`unitroot_ndiffs()` roda a bateria de testes internamente e devolve direto o número de diferenciações recomendado — usado depois como ponto de partida para `ARIMA()` (seção 5), mas vale sempre plotar a série diferenciada e conferir visualmente se ela de fato parece estacionária (oscilando em torno de zero, sem tendência residual) antes de aceitar o número sugerido cegamente.

---

4. Autocorrelação (ACF) e autocorrelação parcial (PACF)

4.1 O que cada uma mede

- **ACF** no lag *k*: correlação entre Y_t e Y_{t-k}, **sem controlar** pelos lags intermediários — inclui efeito indireto (se Y_t correlaciona com Y_{t-1} e Y_{t-1} correlaciona com Y_{t-2}, parte da correlação ACF em lag 2 é só "eco" do lag 1).
- **PACF** no lag *k*: correlação entre Y_t e Y_{t-k} **controlando** pelos lags intermediários ($Y_{t-1}, \ldots, Y_{t-k+1}$) — o efeito "direto" e isolado daquele lag específico. Conceitualmente, é o mesmo tipo de raciocínio de

coeficiente parcial em regressão múltipla (Módulo 4): mede o efeito de uma variável (aqui, um lag) segurando as demais constantes.

```
componentes %>% ACF(indice_vendas, lag_max = 24) %>% autoplot()
componentes %>% PACF(indice_vendas, lag_max = 24) %>% autoplot()

# os dois de uma vez
painel_tsb %>% filter(cidade_id == "CID01") %>% gg_tsdisplay(indice_vendas
, plot_type = "partial")
```

4.2 Leitura prática para identificar ordem do modelo (regra de Box-Jenkins, ponto de partida — não substitui `ARIMA()` automático, mas ajuda a interpretar o resultado dele)

Padrão observado	Sugere
ACF corta abruptamente após lag *q*, PACF decai gradualmente	MA(*q*)
PACF corta abruptamente após lag *p*, ACF decai gradualmente	AR(*p*)
Ambos decaem gradualmente (sem corte claro)	ARMA misto — ordem exata via `ARIMA()` automático
ACF com picos regulares a cada *s* lags (ex.: a cada 12)	Componente sazonal — SARIMA

Essa tabela é herança direta do que você já usou em Python com `statsmodels.graphics.tsaplots.plot_acf/plot_pacf` — a leitura visual não muda entre ecossistemas, só a função que desenha o gráfico.

---

5. Modelos ARIMA

5.1 Teoria — os três componentes

$$ \text{AR}(p):\ Y_t = c + \sum_{i=1}^{p}\phi_i Y_{t-i} + \varepsilon_t \qquad\quad \text{MA}(q):\ Y_t = c + \varepsilon_t + \sum_{j=1}^{q} \theta_j \varepsilon_{t-j} $$

- **AR (AutoRegressivo), ordem *p*:** o valor atual é combinação linear dos *p* valores passados da própria série — "o passado da série explica o presente".
- **I (Integrado), ordem *d*:** número de diferenciações aplicadas para alcançar estacionariedade (seção 3.3) — não é um termo com coeficiente próprio, é uma transformação prévia dos dados.
- **MA (Médias Móveis), ordem *q*:** o valor atual é combinação linear dos *q* erros de previsão passados — não confundir com "média móvel" no sentido de suavização (Módulo 2/seção 2 deste módulo); aqui é sobre a estrutura do **ruído**, não do sinal.

ARIMA(p, d, q) combina os três. Para dados sazonais, existe a extensão SARIMA$(p, d, q)(P, D, Q)_s$, com s = período sazonal (12 para mensal com sazonalidade anual).

5.2 Seleção automática via AICc — `fable::ARIMA()`

```
library(fable)

modelo <- painel_tsb %>%
  filter(cidade_id == "CID01") %>%
  model(auto = ARIMA(indice_vendas))

report(modelo)
```

`ARIMA()` sem especificar ordem faz busca automática (algoritmo de Hyndman-Khandakar, uma versão eficiente de busca em grade com poda) sobre combinações de $(p, d, q)(P, D, Q)$, escolhendo pelo **AICc** (Critério de Informação de Akaike corrigido para amostra pequena) — mesma lógica de "penalizar complexidade" já vista com R^2 ajustado no Módulo 4, agora aplicada à verossimilhança do modelo em vez de à variância explicada.

A vantagem estrutural do `fable` sobre `forecast::auto.arima()` aparece aqui, no painel inteiro de uma vez:

```
modelos_todas_cidades <- painel_tsb %>%
  model(auto = ARIMA(indice_vendas))    # ajusta UM modelo ARIMA por
cidade, automaticamente

modelos_todas_cidades   # "mable" (model table) -- uma linha por cidade,
uma coluna por especificação de modelo
```

Isso substitui o padrão Python de `for cidade in cidades:` `pmdarima.auto_arima(...)` num loop manual acumulando resultados numa lista — o `mable` já nasce como tabela, com uma linha por chave do `tsibble`, pronta para `report()`, `forecast()` ou `accuracy()` em lote.

5.3 Ajuste manual, para comparar contra o automático

```
manual <- painel_tsb %>%
  filter(cidade_id == "CID01") %>%
  model(manual = ARIMA(indice_vendas ~ pdq(1,1,1) + PDQ(1,1,0, period =
12)))

glance(bind_rows(modelo, manual))   # compara AICc dos dois -- menor é
melhor
```

Especificar `pdq()` / `PDQ()` manualmente é útil como exercício de leitura de ACF/PACF (seção 4.2) e como sanity check — se o automático encontrar algo com AICc muito pior que uma especificação manual óbvia (a partir da leitura do correlograma), vale desconfiar da busca automática antes de aceitar cegamente.

6. Diagnóstico de resíduos

6.1 O que "resíduo bom" significa aqui

Um modelo ARIMA bem ajustado deixa resíduos que se comportam como **ruído branco**: média zero, variância constante, e — o ponto central — **sem autocorrelação residual**. Se sobrar autocorrelação nos resíduos, significa que existe estrutura previsível na série que o modelo não capturou (ordem p/q insuficiente, ou sazonalidade mal especificada).

6.2 Teste de Ljung-Box

- H_0: os resíduos são ruído branco (sem autocorrelação até o lag testado).
- H_1: existe autocorrelação residual.

```
modelo %>% augment() %>% features(.innov, ljung_box, lag = 24, dof = 3)
```

`dof` (graus de liberdade a descontar) deve refletir o número de parâmetros estimados no modelo ($p + q + P + Q$) — descontar isso é necessário porque a estatística do teste perde graus de liberdade proporcionalmente à complexidade do modelo já ajustado, o mesmo princípio de correção por parâmetros estimados já visto no R^2 ajustado e no AICc.

6.3 Diagnóstico visual completo

```
modelo %>% select(auto) %>% gg_tsresiduals()
```

Gera em um único painel: (1) série de resíduos ao longo do tempo — deve parecer ruído sem padrão; (2) ACF dos resíduos — barras devem estar dentro da faixa de significância, sem picos; (3) histograma dos resíduos — deve parecer razoavelmente normal (não é um requisito rígido do ARIMA

como é da regressão linear clássica, mas grandes desvios de normalidade nos resíduos tornam os intervalos de previsão da seção 7.1 menos confiáveis).

---

7. Previsão e avaliação de acurácia

7.1 Previsão com intervalo de confiança

```
previsao <- modelo %>% forecast(h = "12 months")
previsao %>% autoplot(painel_tsb %>% filter(cidade_id == "CID01"))
```

O intervalo de previsão (por padrão 80% e 95%) alarga conforme o horizonte *h* cresce — reflexo direto de incerteza acumulada: prever 1 mês à frente é sempre mais confiável que prever 24 meses à frente, porque o erro de cada passo se propaga para os passos seguintes. Um forecast sem intervalo (só a linha central) omite exatamente a informação mais importante para decisão sob incerteza — o mesmo argumento já feito para IC no Módulo 8.

7.2 Métricas de acurácia — MAE, RMSE, MAPE

$$ \text{MAE} = \frac{1}{n}\sum |Y_t - \hat{Y}_t| \qquad \text{RMSE} = \sqrt{\frac{1}{n}\sum (Y_t - \hat{Y}_t)^2} \qquad \text{MAPE} = \frac{100\%}{n}\sum \left|\frac{Y_t - \hat{Y}_t}{Y_t}\right| $$

- **MAE:** erro médio na escala original — fácil de interpretar ("erra em média X unidades"), mas trata todo erro linearmente (um erro de 100 pesa exatamente 10x um erro de 10).
- **RMSE:** penaliza erros grandes desproporcionalmente (por causa do quadrado) — prefira quando erros grandes são particularmente custosos no seu contexto de decisão.

- **MAPE:** erro relativo (%) — permite comparar acurácia entre séries de escalas diferentes (ex.: comparar previsão de boi gordo em R/*@comprevisãodemilhoemR*/saca), mas **explode ou fica indefinido quando Y_t está perto de zero** — cuidado real para variáveis que cruzam ou tangenciam zero (não é o caso do preço de commodity, que é estritamente positivo, mas seria um problema real, por exemplo, para uma série de lucro líquido que pode ser negativa).

```
modelo %>% forecast(h = "12 months") %>% accuracy(painel_tsb %>% filter(ci
dade_id == "CID01"))
```

7.3 Validação honesta: treino/teste, não só ajuste dentro da amostra

Acurácia calculada nos mesmos dados usados para ajustar o modelo é sempre otimista — o modelo sempre "acerta melhor" o que já viu. Separe um período final como teste, ajuste só no treino, e avalie a previsão contra o teste (nunca visto pelo modelo):

```
treino <- painel_tsb %>% filter(cidade_id == "CID01", mes_ano <
yearmonth("2024-01"))
teste  <- painel_tsb %>% filter(cidade_id == "CID01", mes_ano >=
yearmonth("2024-01"))

modelo_treino <- treino %>% model(ARIMA(indice_vendas))
modelo_treino %>% forecast(h = nrow(teste)) %>% accuracy(teste)
```

Esse é o equivalente direto do `train_test_split` que você já usa em Python — a única particularidade de série temporal é que o corte **tem que respeitar a ordem cronológica** (nunca `sample()` aleatório, que vazaria informação do futuro para o treino).

---

8. Fonte de dados reais: Indicador do Boi Gordo CEPEA/ESALQ

8.1 Por que esta série

O Indicador do Boi Gordo CEPEA/ESALQ é a referência de preço à vista da arroba de boi gordo no estado de São Paulo, publicada diariamente e ininterruptamente desde 1994 (o Indicador atual, com metodologia revisada, tem série contínua desde 1997). É o preço que qualquer operação de pecuária de corte no Brasil acompanha para decisão de venda de lote — conexão direta com o dia a dia de qualquer operação de pecuária de corte.

8.2 Como obter

Link: https://cepea.org.br/br/indicador/boi-gordo.aspx

Passo a passo:

1. Na página do indicador, clique em **"Série de preços"** (ou acesse diretamente https://cepea.org.br/br/indicador/series/boi-gordo.aspx?id=2 — série à vista).
2. O download completo exige preencher um formulário curto (nome/e-mail) — gratuito, sem custo, apenas cadastro. O arquivo vem em `.xls` / `.xlsx`.
3. Leitura no R:

```
library(readxl)
boi <- read_excel("indicador_boi_gordo.xlsx", skip = 3)   # linhas de
cabeçalho variam -- confira antes de fixar `skip`
names(boi)   # confirme os nomes reais das colunas antes de escrever
qualquer código de análise --
             # mesmo aviso do Módulo 6 (INMET) e Módulo 8 (SIDRA):
planilhas de fontes oficiais
             # mudam formatação entre atualizações, e um nome de coluna
errado quebra silenciosamente
```

Licença de uso: os dados do CEPEA são distribuídos sob licença **CC BY-NC 4.0** (uso não comercial, com atribuição) — adequado para este uso educacional; cite a fonte (CEPEA/ESALQ) em qualquer material derivado.

8.3 Preparo — de diário para mensal

A série bruta é diária (dias úteis). Para o exercício de decomposição/ARIMA sazonal (com período $s = 12$), agregue para mensal:

```
library(lubridate)

boi_mensal <- boi %>%
  rename(data = Data, preco = `À vista R$`) %>%   # ajuste os nomes
conforme o que vier real no arquivo
  mutate(mes_ano = yearmonth(data)) %>%
  group_by(mes_ano) %>%
  summarise(preco_medio = mean(preco, na.rm = TRUE)) %>%
  as_tsibble(index = mes_ano)
```

Nota honesta sobre este dataset, para não vender uma expectativa irreal: preço de commodity segue, em boa aproximação, um processo próximo de passeio aleatório (não-estacionário em nível, primeira diferença aproximadamente estacionária) — diferente do painel sintético, que tem sazonalidade forte e limpa por construção. Não espere um SARIMA com componente sazonal claro e dominante aqui; espere algo mais próximo de um ARIMA(p,1,q) simples, com sazonalidade fraca (existe alguma — entressafra de chuva/pasto afeta oferta, como mencionado nos próprios boletins do CEPEA — mas é sutil, não o pico limpo de nov/dez do varejo sintético). Essa diferença **é o ponto pedagógico**: dado real raramente tem a estrutura didaticamente perfeita do dado sintético, e um bom cientista de dados sabe reconhecer quando a "falta de sazonalidade clara" é achado real, não erro de código.

---

9. Diferenças-chave vindas de Python

Conceito	Python (statsmodels / pmdarima)	R (tidyverts)
Estrutura de dados temporal	`DataFrame` com `DatetimeIndex`	`tsibble` (índice de temp chave para painel, nativos
Múltiplas séries (painel)	loop manual ou `groupby().apply()`	`model()` ajusta um mode por chave automaticamer
Decomposição STL	`statsmodels.tsa.seasonal.STL(series).fit()`	`model(STL(y ~ trend() season())) |> components()`
Teste de estacionariedade (ADF)	`statsmodels.tsa.stattools.adfuller(series)`	`tseries::adf.test(seri`
ACF/PACF	`statsmodels.graphics.tsaplots.plot_acf/plot_pacf`	`ACF()` / `PACF()` + `autoplot()`, ou `gg_tsdisplay()`
Auto-ARIMA	`pmdarima.auto_arima(series)`	`fable::ARIMA(y)` (busca automática por padrão)
Diagnóstico de resíduos (Ljung-Box)	`statsmodels.stats.diagnostic.acorr_ljungbox`	`features(.innov, ljung_box, ...)`
Previsão com IC	`model.get_forecast(steps=h).summary_frame()`	`forecast(model, h = "1 months")`
Métricas de acurácia	cálculo manual ou `sklearn.metrics` (sem MAPE nativo até versões recentes)	`accuracy()` — MAE, RM MAPE, MASE prontos

O padrão mais relevante ao migrar: `pmdarima.auto_arima()` já é bom, mas opera série a série — não existe conceito nativo de "ajustar automaticamente para cada grupo de uma tabela". `fable` foi desenhado

desde a base para dados de painel, o que o torna a escolha natural quando o problema real é “preciso de um modelo por [cidade/produto/estação]”, não “preciso de um modelo para uma série única” — exatamente o cenário do painel sintético deste módulo, e também um cenário plausível em operações de pecuária de corte (uma série de preço/produtividade por propriedade, por exemplo).

---

10. Exercícios

Use `painel_regional_varejo.csv` para os exercícios 1-3 e a série do CEPEA (seção 8) para os exercícios 4-6.

Exercício 1 — Decomposição e verificação contra o gabarito

Rode STL sobre `indice_vendas` de 3 cidades diferentes do painel. Para cada uma, confirme visualmente que o pico do componente sazonal cai em nov/dez (como construído no gerador) e que a tendência é crescente. Alguma cidade foge desse padrão? Investigue `nivel_base_cidade` e `fase_local` no script gerador para explicar.

Solução:

```
tres_cidades <- painel_tsb %>% filter(cidade_id %in% c("CID01", "CID15", "
CID30"))

tres_cidades %>%
  model(STL(indice_vendas ~ trend(window = 13) + season(window = "periodic
"))) %>%
  components() %>%
  autoplot()
```

Todas devem mostrar tendência crescente (o gerador usa `tendencia_local` com média positiva 0.0015) e pico sazonal próximo de nov/dez, com pequena variação de fase entre cidades (`fase_local <- rnorm(1, 0, 0.3)`

desloca o pico ligeiramente ano a ano/cidade a cidade) — nenhuma foge drasticamente do padrão porque o gerador não introduziu exceções estruturais, só ruído.

Exercício 2 — Estacionariedade: nível vs. diferença

Para `CID01`, rode ADF e KPSS sobre `indice_vendas` bruto e sobre a série diferenciada uma vez. A conclusão de estacionariedade muda entre as duas versões? Isso é esperado dado que a série tem tendência de crescimento embutida?

Solução:

```
serie <- painel_tsb %>% filter(cidade_id == "CID01")

adf.test(serie$indice_vendas)
serie %>% features(indice_vendas, unitroot_kpss)

serie_diff <- serie %>% mutate(diff1 = difference(indice_vendas))
adf.test(na.omit(serie_diff$diff1))
serie_diff %>% features(diff1, unitroot_kpss)
```

Esperado: série bruta não-estacionária (ADF não rejeita H_0 de raiz unitária; KPSS rejeita H_0 de estacionariedade) por causa da tendência de crescimento construída no gerador; após uma diferenciação, ambos os testes devem convergir para "estacionária" — a tendência é removida pela diferença de primeira ordem.

Exercício 3 — ARIMA automático em lote no painel inteiro

Ajuste `ARIMA(indice_vendas)` para todas as 50 cidades de uma vez (`model()` sobre `painel_tsb` completo, sem filtrar). Extraia, via `glance()`, a ordem (p, d, q) escolhida para cada cidade. Há uma ordem que se repete na maioria das cidades? Isso é esperado dado que todas as séries compartilham a mesma estrutura geradora (só variam parâmetros locais)?

Solução:

```
modelos_painel <- painel_tsb %>% model(ARIMA(indice_vendas))

glance(modelos_painel) %>% select(cidade_id, AICc)   # AICc não expõe
(p,d,q) direto
modelos_painel %>% pull(.model) %>% map_chr(~ deparse(.x))  # inspeção
mais direta da ordem, por modelo

# alternativa mais simples: report() cidade a cidade, ou extrair via
broom::tidy() em loop
```

Esperado: como todas as séries compartilham a mesma estrutura geradora (tendência + senoide anual + ruído gaussiano), a maioria das cidades deve convergir para ordens ARIMA parecidas (tipicamente algo em torno de (1, 1, 1) ou (0, 1, 1) com componente sazonal), com variação pontual explicada por ruído amostral específico de cada série.

Exercício 4 — Boi gordo: ACF/PACF e escolha de ordem

Baixe e prepare a série mensal do boi gordo (seção 8.3). Plote ACF e PACF da série em nível e da série diferenciada. Com base na tabela da seção 4.2, qual ordem ARIMA você proporia manualmente antes de rodar o automático?

Solução esperada (conceitual, sem gabarito numérico fixo — depende do recorte de datas baixado): a série em nível deve mostrar ACF com decaimento lento (característico de não-estacionariedade/raiz unitária) — típico de preço de commodity. Após diferenciar uma vez, ACF e PACF devem se aproximar de ruído branco ou mostrar no máximo 1-2 lags significativos, sugerindo um AR(1), MA(1) ou ARMA(1,1) sobre a série diferenciada — ou seja, algo como ARIMA(1,1,0), (0,1,1) ou (1,1,1). Compare com o que `ARIMA()` automático escolhe (Exercício 5).

Exercício 5 — Boi gordo: ajuste, diagnóstico e comparação com o automático

Rode `ARIMA()` automático sobre a série mensal do boi gordo. Rode `gg_tsresiduals()` e o teste de Ljung-Box. Os resíduos passam no diagnóstico de ruído branco? Se não passarem, o que isso sugere sobre a adequação do modelo linear ARIMA para essa série especificamente?

Solução esperada: se Ljung-Box rejeitar H_0 (indicando autocorrelação residual), é sinal de que a estrutura linear ARIMA não capturou toda a dependência temporal — plausível para preço de commodity agropecuária, que sofre choques externos (clima, câmbio, demanda de exportação — visível inclusive nos próprios boletins do CEPEA citados na seção 8) não modelados por uma estrutura puramente autorregressiva sobre o próprio preço passado. Isso não invalida o exercício — é um resultado honesto e didaticamente relevante: ARIMA univariado captura só a autocorrelação da própria série, não choques exógenos, e reconhecer esse limite é tão importante quanto saber ajustar o modelo.

Exercício 6 (desafio) — Validação honesta: treino/teste no boi gordo

Separe os últimos 12 meses da série do boi gordo como teste. Ajuste ARIMA só no restante, preveja os 12 meses e calcule MAE/RMSE/MAPE contra o valor real observado. Compare visualmente a previsão com o valor real — o intervalo de 95% cobriu o valor real na maior parte dos meses?

Solução:

```
boi_treino <- boi_mensal %>% filter(mes_ano < max(mes_ano) - 11)
boi_teste  <- boi_mensal %>% filter(mes_ano >= max(mes_ano) - 11)

modelo_boi <- boi_treino %>% model(ARIMA(preco_medio))
prev_boi <- modelo_boi %>% forecast(h = 12)

prev_boi %>% accuracy(boi_teste)
prev_boi %>% autoplot(bind_rows(boi_treino, boi_teste))
```

Esperado: para um horizonte de 12 meses num processo próximo de passeio aleatório, o intervalo de 95% deve alargar consideravelmente ao longo do horizonte, e é plausível (não um erro) que o valor real saia do intervalo em algum ponto se houver choque de mercado no período de teste — a demonstração central do exercício é que **a largura crescente do intervalo é a forma honesta do modelo admitir "quanto mais longe no futuro, menos confiável é a previsão"**, não uma falha a esconder.

---

11. Entrega esperada do módulo

Script `series_temporais_varejo_boi.R` que:

1. Carrega `painel_regional_varejo.csv` como `tsibble` (índice `mes_ano`, chave `cidade_id`) e roda STL para pelo menos 3 cidades, com verificação visual contra a estrutura geradora conhecida
2. Roda ADF + KPSS sobre uma série do painel em nível e diferenciada, confirmando a mudança de estacionariedade
3. Ajusta `ARIMA()` automático sobre o painel inteiro (todas as chaves de uma vez) e reporta a ordem escolhida por pelo menos 5 cidades
4. Baixa, limpa e agrega para mensal a série do boi gordo CEPEA (seção 8)
5. Roda ACF/PACF, ADF/KPSS, `ARIMA()` automático e diagnóstico de resíduos (Ljung-Box + `gg_tsresiduals()`) sobre a série do boi gordo
6. Roda a validação treino/teste (últimos 12 meses) da seção 7.3/Exercício 6, com MAE/RMSE/MAPE e gráfico de previsão vs. real

12. Critério de domínio antes de avançar ao Módulo 11

- ☐ Explica a diferença estrutural entre `ts` e `tsibble`, e por que `key` importa especificamente para dados de painel
- ☐ Sabe escolher entre decomposição aditiva e multiplicativa observando o comportamento da amplitude sazonal
- ☐ Explica por que ARIMA exige estacionariedade, e sabe interpretar ADF e KPSS em conjunto (hipóteses nulas opostas)
- ☐ Lê um correlograma ACF/PACF e propõe uma ordem ARIMA inicial plausível antes de rodar o automático
- ☐ Explica os três componentes do ARIMA (AR, I, MA) sem confundir "médias móveis" do MA com suavização de série
- ☐ Sabe interpretar o teste de Ljung-Box sobre resíduos, e o que fazer se ele indicar autocorrelação residual
- ☐ Distingue MAE, RMSE e MAPE, sabendo quando cada uma é a métrica certa (e quando MAPE quebra)
- ☐ Sabe montar uma validação treino/teste respeitando a ordem cronológica (nunca split aleatório em série temporal)
- ☐ Todos os 6 exercícios rodam sem erro, com interpretação escrita conectando resultado numérico ao comportamento real da série (sintética ou do boi gordo)

13. Nota: decisões deste módulo

1. **Estratégia híbrida mantida, com papéis invertidos em relação a módulos anteriores:** aqui o sintético (`painel_regional_varejo.csv`) carrega a maior carga didática (decomposição, ARIMA em lote, verificação contra gabarito conhecido) porque sua estrutura foi desenhada especificamente para isso (`gerar_datasets_sinteticos.R` já documentava "cobre módulos 8 e 9" desde a criação do dataset). O dado real (boi gordo CEPEA) entra como o "confronto com a realidade" — uma série sem sazonalidade limpa, plausivelmente com resíduo não totalmente branco, que ensina o limite prático do ARIMA univariado.
2. **ADF via `tseries::adf.test()` , não `urca::ur.df()`** — escolhido por ser a interface mais direta e mais próxima em espírito de `statsmodels.tsa.stattools.adfuller()` , já familiar. `urca::ur.df()` foi mencionado como alternativa para quem precisar de mais controle sobre a especificação do teste (com/sem tendência, número de lags), mas não aprofundado para não inflar o módulo com uma variante que não muda a decisão prática.
3. **Nenhuma pendência nova ficou em aberto.** O módulo fecha o ciclo de séries temporais univariadas; o Módulo 11 (Espacial) reaproveita diretamente o mesmo `painel_regional_varejo.csv` (que já carrega colunas de coordenadas espaciais, hoje ainda não usadas) — outra razão pela qual esse dataset foi construído como painel espaço-temporal desde o início, não só temporal.

MÓDULO 11 · APLICAÇÕES

Estatística Espacial

1. Estrutura de dados espaciais em R: o pacote `sf`

1.1 Objetos `sf` — geometria como uma coluna da tabela

```
library(sf)

# sf trata geometria como só mais uma coluna de um data frame comum -- não
uma estrutura à parte
# (diferente do antigo padrão `sp`, que separava dados e geometria em
slots distintos)
class(municipios_sc)   # "sf" "data.frame" -- um data frame que sabe
desenhar a si mesmo
```

Um objeto `sf` é um data frame comum com uma coluna especial (`geometry`, tipicamente `sfc` — simple feature column) contendo a geometria (ponto, linha, polígono) de cada linha. Isso significa que **todo verbo `dplyr` que você já usa funciona diretamente** (`filter()`, `mutate()`, `left_join()`, `group_by()` + `summarise()`) — a geometria "anda junto" automaticamente, inclusive sendo unida/dissolvida corretamente em `group_by() %>% summarise()`. É o equivalente direto de `GeoDataFrame` do `geopandas`, com a mesma filosofia de design (geometria como coluna, não como estrutura à parte).

1.2 Sistema de coordenadas (CRS)

```
st_crs(municipios_sc)                          # verifica o CRS atual
municipios_sc <- st_transform(municipios_sc, crs = 31982)   # SIRGAS
2000 / UTM 22S -- projeção métrica para SC
```

Ponto crítico, o erro mais comum de quem chega de PostGIS/Kepler.gl sem prestar atenção nisso no R: dados baixados como GPS (`EPSG:4326`, lat/lon em graus) não devem ser usados diretamente para cálculo de distância ou área — grau não é uma unidade de distância constante (1 grau de longitude

vale menos km perto dos polos que no equador). Para operações métricas (distância entre centróides, matriz de vizinhança por distância, krigagem), sempre reprojete para um CRS métrico apropriado à região — `31982` (UTM 22S) cobre bem Santa Catarina.

1.3 `geobr` — geometrias oficiais do IBGE, sem download manual

```
library(geobr)

municipios_sc <- read_municipality(code_state = "SC", year = 2020)
```

`geobr` é a forma mais confiável de obter geometria municipal brasileira em R — busca direto do geoserver do IBGE, sem passar por shapefile baixado manualmente (elimina uma fonte inteira de erro: shapefile desatualizado, códigos de município divergentes de fonte para fonte). Retorna um `sf` já pronto, com `code_muni` (código IBGE de 7 dígitos) como chave — a mesma chave usada pelo SIDRA (Módulos 8-9) e pelo Atlas do Desenvolvimento Humano (seção 8), o que torna o join direto e sem ambiguidade de nome de município (problema real: mais de um município com nome igual em estados diferentes, e grafias que variam entre fontes).

---

2. Autocorrelação espacial: Índice de Moran

2.1 A pergunta que a autocorrelação espacial responde

“Municípios com valor alto de X estão espacialmente perto de outros municípios com valor alto de X (mais do que se a distribuição espacial fosse aleatória)?” Se sim, existe autocorrelação espacial positiva — o caso mais comum na prática (a maioria dos fenômenos socioeconômicos e ambientais tem esse padrão). Autocorrelação negativa (vizinhos sistematicamente diferentes, padrão de tabuleiro de xadrez) é rara mas existe, tipicamente em fenômenos competitivos por recurso limitado.

2.2 Matriz de peso espacial — pré-requisito para qualquer estatística espacial

Antes de calcular qualquer coisa, é preciso definir formalmente "quem é vizinho de quem" — decisão que afeta o resultado, não um detalhe técnico neutro (mais detalhe na seção 3).

```
library(spdep)

vizinhos <- poly2nb(municipios_sc, queen = TRUE)   # contiguidade rainha
-- compartilha borda OU vértice
pesos <- nb2listw(vizinhos, style = "W", zero.policy = TRUE)   # "W" =
padronizado por linha (soma 1 por linha)
```

`zero.policy = TRUE` evita erro se algum município não tiver vizinho contíguo (ilha, ou erro de geometria) — nesse caso ele fica sem peso (linha de zeros), o que precisa ser tratado explicitamente na interpretação (município sem vizinho simplesmente não entra no cálculo local ali).

2.3 Índice de Moran Global (Moran's I)

$$ I = \frac{n}{\sum_i \sum_j w_{ij}} \cdot \frac{\sum_i \sum_j w_{ij}(x_i - \bar{x})(x_j - \bar{x})}{\sum_i (x_i - \bar{x})^2} $$

Leitura: é essencialmente uma correlação entre o valor de x em cada unidade e a **média ponderada de x nos vizinhos** dessa unidade (o "lag espacial" de x) — por isso I se comporta de forma similar a um coeficiente de correlação de Pearson, variando aproximadamente entre -1 e $+1$, com $I \approx 0$ sob ausência de autocorrelação (o valor esperado exato sob H_0 é $-1/(n-1)$, praticamente zero para n grande).

- H_0: não há autocorrelação espacial (distribuição espacial aleatória de x).
- H_1: há autocorrelação espacial (positiva ou negativa).

```
moran.test(municipios_sc$esperanca_vida, pesos, zero.policy = TRUE)

# alternativa mais robusta: teste por permutação Monte Carlo, sem depender
de normalidade assintótica
moran.mc(municipios_sc$esperanca_vida, pesos, nsim = 999, zero.policy = TR
UE)
```

`moran.test()` usa aproximação assintótica (válida para *n* razoavelmente grande); `moran.mc()` embaralha os valores de *x* entre as unidades espaciais 999 vezes, recalcula *I* a cada embaralhamento, e compara o *I* observado contra essa distribuição empírica sob H_0 — o mesmo espírito de teste de permutação já mencionado informalmente em módulos anteriores, aqui como ferramenta central porque a distribuição teórica de *I* depende fortemente da estrutura da matriz de pesos, o que a torna menos robusta que testes paramétricos convencionais.

2.4 Geary's C — a alternativa que usa diferença ao quadrado, não produto cruzado

$$ C = \frac{(n-1)}{2\sum_i\sum_j w_{ij}} \cdot \frac{\sum_i \sum_j w_{ij} (x_i - x_j)^2}{\sum_i (x_i - \bar{x})^2} $$

```
geary.test(municipios_sc$esperanca_vida, pesos, zero.policy = TRUE)
```

Diferença de interpretação prática: C varia de 0 a 2, com $C = 1$ sob ausência de autocorrelação, $C < 1$ indicando autocorrelação positiva (padrão inverso de Moran's *I*, que é positivo quando há autocorrelação positiva — cuidado ao trocar de um para o outro na hora de interpretar sinal). Geary's C é mais sensível a diferenças locais bruscas (por usar diferença ao quadrado direta entre vizinhos, não produto cruzado com a média global), então tende a reagir mais a autocorrelação **local** forte mesmo quando o padrão **global** é fraco — na prática, os dois quase sempre concordam na direção, e Moran's I é o mais reportado por convenção e por ter interpretação mais direta como "correlação com o vizinho".

2.5 Verificação no painel sintético — confirmando contra o gabarito conhecido

O painel foi construído com `indice_renda_media` compartilhando um fator regional latente por `polo` (5 clusters espaciais) — deve mostrar Moran's I positivo e significativo. `var_placebo_espacial` é ruído puro sem estrutura de polo — Moran's I deve ficar perto de zero e não significativo.

```
cidades <- painel_tsb %>% as_tibble() %>% distinct(cidade_id, .keep_all =
TRUE)  # um registro por cidade
coords <- as.matrix(cidades[, c("coord_x_km", "coord_y_km")])

vizinhos_knn <- knn2nb(knearneigh(coords, k = 5))   # 5 vizinhos mais
próximos -- pontos, não polígonos aqui
pesos_knn <- nb2listw(vizinhos_knn, style = "W")

moran.mc(cidades$indice_renda_media, pesos_knn, nsim = 999)     #
esperado: I positivo, significativo
moran.mc(cidades$var_placebo_espacial, pesos_knn, nsim = 999)   #
esperado: I perto de zero, não significativo
```

Para dados de **ponto** (coordenada de cidade, não polígono de município), a matriz de pesos usa vizinhança por distância/k-vizinhos mais próximos (`knearneigh`), não contiguidade de borda (`poly2nb` , que exige polígono) — distinção retomada na seção 3.

---

3. Matrizes de vizinhança — aprofundamento

3.1 Contiguidade: rainha (queen) vs. torre (rook)

```
viz_queen <- poly2nb(municipios_sc, queen = TRUE)    # compartilha borda
OU vértice (mais permissivo)
viz_rook  <- poly2nb(municipios_sc, queen = FALSE)   # compartilha só
borda (mais restritivo)
```

`queen` inclui municípios que só se tocam num único ponto (vértice comum, sem fronteira linear compartilhada) como vizinhos; `rook` exige fronteira compartilhada de fato. Na prática, para malha municipal brasileira (polígonos irregulares, poucos casos de toque só por vértice), a diferença entre os dois costuma ser pequena — mas vale inspecionar quantos vizinhos cada método atribui antes de assumir que não importa:

```
summary(viz_queen)
# nº médio de vizinhos, mínimo, máximo -- queen tende a dar nº médio um
pouco maior
```

3.2 Distância: quando contiguidade não faz sentido

Contiguidade de borda só existe para polígonos. Para dados de ponto (estação meteorológica, cidade como coordenada única, endereço geocodificado), as opções são:

```
# k-vizinhos mais próximos -- cada ponto tem exatamente k vizinhos,
independente da distância real
viz_knn <- knn2nb(knearneigh(coords, k = 5))

# banda fixa -- todos os pontos dentro de um raio, número de vizinhos
varia por ponto
viz_banda <- dnearneigh(coords, d1 = 0, d2 = 30, longlat = FALSE)
# raio de 30 km, coords já em km
```

Trade-off real entre os dois: k-vizinhos garante que nenhum ponto fique isolado (sempre tem exatamente *k* vizinhos), mas pode forçar vizinhança entre pontos muito distantes numa área de baixa densidade (poucos pontos numa região grande). Banda fixa respeita a noção física de "perto de verdade", mas pode deixar pontos isolados (sem nenhum vizinho dentro do raio) numa área de baixa densidade — exige `zero.policy = TRUE` e decisão explícita sobre o que fazer com esses pontos.

3.3 Padronização de pesos (`style` em `nb2listw`)

- `"W"` (row-standardized): cada linha soma 1 — o peso de cada vizinho é 1/(nº de vizinhos). Interpretação: o lag espacial é uma **média** dos vizinhos. É o padrão mais usado e mais fácil de interpretar.
- `"B"` (binário): peso 1 para vizinho, 0 para não-vizinho, sem normalizar — o lag espacial vira uma **soma**, não uma média, o que faz unidades com mais vizinhos terem lag sistematicamente maior só por isso (viés artificial se o número de vizinhos varia muito entre unidades).
- `"C"` (globally standardized): normaliza pela soma total de todos os pesos da matriz inteira, não linha a linha.

Para a maioria das aplicações (Moran's I, regressão espacial), `"W"` é a escolha default defensável — é o que faz o lag espacial ter a mesma escala da variável original, facilitando interpretação de coeficiente na regressão espacial (seção 7).

---

4. LISA — Indicadores Locais de Associação Espacial

4.1 Por que Moran's I global não é suficiente

Moran's I global dá **um número para o mapa inteiro** — mas autocorrelação pode ser forte numa região e ausente em outra (heterogeneidade espacial). LISA (Local Indicators of Spatial Association, Anselin 1995) calcula um Moran's I **por unidade espacial**, permitindo identificar exatamente onde estão os clusters e os outliers espaciais, não só que eles existem em algum lugar do mapa.

4.2 Cálculo e classificação em quadrantes

```
municipios_sc$esperanca_vida_z <- as.numeric(scale(municipios_sc$esperanca
_vida))   # padronizado (z-score)
lisa <- localmoran(municipios_sc$esperanca_vida, pesos, zero.policy =
TRUE)

municipios_sc$lisa_i <- lisa[, "Ii"]
municipios_sc$lisa_p <- lisa[, "Pr(z != E(Ii))"]

# lag espacial padronizado -- necessário para classificar o quadrante
municipios_sc$lag_z <- lag.listw(pesos, municipios_sc$esperanca_vida_z, ze
ro.policy = TRUE)

municipios_sc$cluster <- case_when(
  municipios_sc$lisa_p >= 0.05 ~ "Não significativo",
  municipios_sc$esperanca_vida_z > 0 & municipios_sc$lag_z > 0 ~ "Alto-
Alto",
  municipios_sc$esperanca_vida_z < 0 & municipios_sc$lag_z < 0 ~ "Baixo-
Baixo",
  municipios_sc$esperanca_vida_z > 0 & municipios_sc$lag_z < 0 ~ "Alto-
Baixo (outlier)",
  municipios_sc$esperanca_vida_z < 0 & municipios_sc$lag_z > 0 ~ "Baixo-
Alto (outlier)"
)
```

Interpretação de cada quadrante:

Quadrante	Significado
Alto-Alto	valor alto, cercado de vizinhos com valor alto — núcleo de um cluster de valores elevados
Baixo-Baixo	valor baixo, cercado de vizinhos com valor baixo — núcleo de um cluster de valores baixos
Alto-Baixo	valor alto isolado entre vizinhos de valor baixo — outlier espacial local
Baixo-Alto	valor baixo isolado entre vizinhos de valor alto — outlier espacial local

Os dois primeiros são **clusters** (o que a maior parte de uma análise LISA está procurando); os dois últimos são **outliers espaciais** — municípios que destoam fortemente da vizinhança imediata, candidatos naturais a investigação qualitativa (o que há de diferente ali? política pública específica, característica geográfica atípica, erro de medição?).

4.3 Mapa de cluster LISA

```
library(ggplot2)

ggplot(municipios_sc) +
  geom_sf(aes(fill = cluster)) +
  scale_fill_manual(values = c("Alto-Alto" = "#d7191c", "Baixo-Baixo" =
"#2c7bb6",
                               "Alto-Baixo (outlier)" = "#fdae61", "Baixo
-Alto (outlier)" = "#abd9e9",
                               "Não significativo" = "grey90")) +
  labs(title = "LISA -- Esperança de vida ao nascer, municípios de SC", fi
ll = "Cluster")
```

O mapa de cluster LISA é a peça central de qualquer análise de saúde/ambiente por município — é o output que, no seu mestrado, mostraria diretamente onde estão os clusters geográficos de um indicador de saúde, informação que uma tabela de correlação bruta simplesmente não entrega.

---

5. Interpolação geoestatística: variograma e krigagem

5.1 Quando isso se aplica

Diferente das seções anteriores (dado já existe para toda unidade espacial — todo município tem um valor), krigagem resolve o problema oposto: você tem uma variável medida só em **pontos amostrados** (estações meteorológicas, poços de sondagem, pontos de coleta) e quer estimar o valor em locais **não amostrados**, explorando a estrutura de correlação espacial (pontos próximos tendem a ter valores parecidos) para interpolar

de forma estatisticamente informada — diferente de interpolação geométrica simples (IDW, vizinho mais próximo), que ignora completamente a estrutura de correlação real dos dados.

5.2 O variograma — a peça central

$$ \gamma(h) = \frac{1}{2N(h)} \sum_{i=1}^{N(h)} \left(z(s_i) - z(s_i + h) \right)^2 $$

O variograma empírico mede a semivariância $\gamma(h)$ — em média, o quanto dois pontos separados por distância h diferem entre si — para várias distâncias h. Três parâmetros lidos do variograma ajustado:

- **Alcance (range):** distância a partir da qual pontos deixam de ter correlação espacial entre si (o variograma "estabiliza"). Além do alcance, mais próximo ou mais distante dá no mesmo para fins de predição.
- **Patamar (sill):** o nível de semivariância onde o variograma estabiliza — aproximadamente a variância total dos dados quando não há mais correlação espacial a explorar.
- **Efeito pepita (nugget):** a semivariância em $h \to 0$, que deveria ser zero num mundo sem erro de medição — um nugget positivo indica variabilidade em escala menor que a menor distância amostrada, ou ruído de medição.

```
library(gstat)
library(sf)

# estações INMET-SC (Módulo 6) com coordenadas e temperatura média anual
estacoes_sf <- st_as_sf(estacoes_inmet, coords = c("longitude",
"latitude"), crs = 4326) %>%
  st_transform(31982)

variograma_emp <- variogram(temp_media ~ 1, estacoes_sf)
# ~1 = krigagem ordinária, sem tendência externa
plot(variograma_emp)

modelo_ajustado <- fit.variogram(variograma_emp, model = vgm(psill = NA, m
odel = "Sph", range = NA, nugget = NA))
plot(variograma_emp, modelo_ajustado)
```

`vgm("Sph")` especifica o modelo teórico esférico (um dos mais comuns para fenômenos ambientais — crescimento suave da semivariância até o alcance, depois platô) — outras opções comuns são `"Exp"` (exponencial, decaimento mais rápido no início) e `"Gau"` (gaussiano, mais suave). `fit.variogram()` ajusta os parâmetros (patamar, alcance, pepita) do modelo escolhido aos pontos empíricos por mínimos quadrados.

5.3 Krigagem ordinária — interpolação para uma grade

```
grade_sc <- st_make_grid(municipios_sc, cellsize = 5000, what =
"centers") %>% st_as_sf()   # grade de 5km

krig_resultado <- krige(temp_media ~ 1, estacoes_sf, newdata = grade_sc, m
odel = modelo_ajustado)

ggplot() +
  geom_sf(data = krig_resultado, aes(color = var1.pred)) +
  scale_color_viridis_c() +
  labs(title = "Temperatura média interpolada -- Santa Catarina", color =
"°C (predito)")
```

A krigagem retorna, para cada célula da grade, `var1.pred` (o valor interpolado) **e** `var1.var` (a variância da predição — incerteza) — essa segunda saída é a diferença central em relação a IDW ou vizinho mais próximo, que só dão o valor interpolado sem quantificar confiança. Áreas

longe de qualquer estação amostrada têm `var1.var` mais alta — informação que qualquer decisão baseada no mapa interpolado deveria considerar antes de tratar a superfície inteira como igualmente confiável.

---

6. Análise de padrões de pontos: `spatstat`

6.1 A pergunta de Aleatoriedade Espacial Completa (CSR)

Diferente das seções anteriores (que analisam **valores** medidos em localizações), esta seção analisa a **localização em si** como o dado: dado um conjunto de pontos no espaço (estações meteorológicas, ocorrências de um evento), eles estão distribuídos de forma aleatória, agrupada (clusterizada) ou regular (mais espaçada que o acaso explicaria)?

- H_0 (CSR — Complete Spatial Randomness): os pontos seguem um processo de Poisson homogêneo — mesma intensidade esperada em qualquer sub-região, sem interação entre pontos.
- H_1: existe atração (clusterização — pontos tendem a estar perto uns dos outros) ou repulsão (padrão regular — pontos tendem a se espalhar mais do que o acaso).

6.2 Preparo e teste de quadrantes

```
library(spatstat)

janela <- as.owin(st_union(municipios_sc))   # janela de observação =
contorno de SC
pontos_ppp <- as.ppp(st_coordinates(estacoes_sf), W = janela)

quadrat.test(pontos_ppp, nx = 4, ny = 4)   # divide a janela numa grade
4x4, compara contagem observada vs. esperada sob CSR
```

O teste de quadrantes divide a área em células, conta pontos por célula, e compara contra a distribuição de Poisson esperada sob CSR (mesma lógica de teste qui-quadrado de aderência) — simples e interpretável, mas sensível ao tamanho de célula escolhido (`nx` / `ny`), o que motiva a função K como alternativa mais robusta.

6.3 Função K de Ripley — mais robusta que quadrantes

```
K_obs <- Kest(pontos_ppp)
plot(K_obs)

# envelope de simulação -- gera N realizações de CSR e compara a curva K
observada contra a faixa esperada
envelope_csr <- envelope(pontos_ppp, Kest, nsim = 99)
plot(envelope_csr)
```

A função K conta, para cada distância *r*, o número médio de outros pontos dentro de um raio *r* de cada ponto — sem depender de uma grade arbitrária como o teste de quadrantes. Se a curva K observada estiver **acima** da envelope de simulação sob CSR, há evidência de clusterização (mais vizinhos próximos que o esperado ao acaso); se estiver **abaixo**, há evidência de padrão regular (repulsão).

Aplicação real: estações meteorológicas tendem a mostrar padrão **não aleatório** — concentradas perto de centros urbanos e ao longo de rodovias (onde é mais fácil manter e acessar a estação), com “vazios” em áreas remotas. Isso não é uma falha da rede de estações, é uma decisão logística — mas é exatamente o tipo de viés de amostragem espacial que precisa ser reconhecido antes de assumir que a krigagem da seção 5 representa igualmente bem toda a área do estado.

---

7. Regressão espacial: `spatialreg`

7.1 Por que OLS clássico falha aqui

Regressão linear clássica (Módulo 4) assume resíduos independentes. Se a variável dependente (ou uma variável omitida correlacionada com ela) tem estrutura espacial, os resíduos de um `lm()` comum tendem a ficar espacialmente autocorrelacionados — o que não invalida os coeficientes estimados (continuam não-viesados sob certas condições), mas **invalida os erros padrão, e portanto os testes de significância e intervalos de confiança**, porque a fórmula clássica de erro padrão assume informação independente ponto a ponto — com autocorrelação espacial, o número **efetivo** de observações independentes é menor que *n*, e o erro padrão calculado como se fossem *n* observações independentes fica artificialmente pequeno (inflando significância — o mesmo mecanismo de pseudo-replicação da seção 0, agora com nome técnico).

```
modelo_ols <- lm(esperanca_vida ~ saneamento, data = municipios_sc)
summary(modelo_ols)

lm.morantest(modelo_ols, pesos, zero.policy = TRUE)
# Moran's I aplicado aos RESÍDUOS do OLS, não à variável bruta
```

Se `lm.morantest()` indicar autocorrelação espacial significativa nos resíduos, o OLS clássico está estruturalmente inadequado para esse dado — hora de considerar um modelo espacial.

7.2 Diagnóstico: qual modelo espacial usar — Multiplicadores de Lagrange

```
testes_lm <- lm.LMtests(modelo_ols, pesos, test = c("LMerr", "LMlag", "RLM
err", "RLMlag"), zero.policy = TRUE)
summary(testes_lm)
```

Os testes de Multiplicador de Lagrange (Anselin) orientam a escolha entre os dois modelos espaciais principais:

- **LMerr/LMlag significativos, mas só um dos robustos (RLMerr/RLMlag) permanece significativo:** siga o robusto — ele é a versão que já descontou a possibilidade do outro tipo de dependência estar presente também.
- **Ambos os robustos significativos:** situação ambígua; considere um modelo que combine os dois efeitos (SAC/SARAR) ou reporte os dois modelos lado a lado com ressalva explícita.
- **Nenhum significativo:** a autocorrelação nos resíduos do OLS pode não justificar um modelo espacial completo — revise se a matriz de pesos está bem especificada antes de descartar espacialidade de vez.

7.3 Spatial Lag Model (SAR) — a variável dependente dos vizinhos afeta a própria variável dependente

$$y = \rho W y + X\beta + \varepsilon$$

```
library(spatialreg)

modelo_lag <- lagsarlm(esperanca_vida ~ saneamento, data = municipios_sc,
listw = pesos, zero.policy = TRUE)
summary(modelo_lag)
```

Interpretação: o valor de y (esperança de vida) num município é diretamente influenciado pelo valor de y nos municípios vizinhos (ponderado por ρ), além do efeito direto de X (saneamento). Faz sentido teórico quando há um mecanismo de transbordamento real — ex.: infraestrutura de saúde regional atende mais de um município, fluxo de pacientes entre municípios vizinhos.

7.4 Spatial Error Model (SEM) — a dependência espacial está no que ficou de fora do modelo

$y = X\beta + u, \quad u = \lambda W u + \varepsilon$

```
modelo_erro <- errorsarlm(esperanca_vida ~ saneamento, data =
municipios_sc, listw = pesos, zero.policy = TRUE)
summary(modelo_erro)
```

Interpretação: não é *y* em si que se propaga espacialmente, é o **erro** — variáveis omitidas com estrutura espacial (ex.: qualidade do ar, renda regional, algo não medido mas que também varia suavemente no espaço) que afetam municípios vizinhos de forma correlacionada. É frequentemente a escolha mais defensável quando a autocorrelação vem de "fatores de fundo compartilhados regionalmente" — que é exatamente o mecanismo desenhado no painel sintético (`polo_efeito_renda`) e o cenário mais plausível para variável ambiental × indicador de saúde.

7.5 Aplicação central: desconstruindo a pseudo-replicação no painel sintético

```
cidades_sf <- st_as_sf(cidades, coords = c("coord_x_km", "coord_y_km"))
pesos_cidades <- nb2listw(knn2nb(knearneigh(coords, k = 5)), style = "W")

# passo 1 -- correlação bruta / OLS ingênuo
modelo_ingenuo <- lm(ticket_medio_regional ~ indice_renda_media, data =
cidades)
summary(modelo_ingenuo)   # coeficiente de renda deve sair significativo

# passo 2 -- os resíduos desse modelo têm autocorrelação espacial?
lm.morantest(modelo_ingenuo, pesos_cidades, zero.policy = TRUE)

# passo 3 -- reajustar como spatial error model, já que a dependência vem
do fator regional OMITIDO (polo)
modelo_sem <- errorsarlm(ticket_medio_regional ~ indice_renda_media, data
= cidades, listw = pesos_cidades, zero.policy = TRUE)
summary(modelo_sem)
```

Como o gerador constrói `ticket_medio_regional` e `indice_renda_media` a partir do **mesmo** fator latente `polo_efeito_renda` (seção 2.5), espera-se: (1) Moran's I significativo nos resíduos do OLS ingênuo — a estrutura de polo ficou de fora do modelo e vaza para o erro; (2) o coeficiente de `indice_renda_media` no `errorsarlm` deve vir **atenuado** (menor magnitude, possivelmente perdendo significância) em relação ao OLS ingênuo — porque parte da correlação bruta era, de fato, efeito do fator regional compartilhado, não relação direta entre as duas variáveis. Essa é a demonstração mais importante do módulo inteiro: o número que muda entre `modelo_ingenuo` e `modelo_sem` é, literalmente, o tamanho do viés de pseudo-replicação espacial que a análise ingênua carregava.

---

8. Fonte de dados reais

8.1 Municípios de Santa Catarina — geometria (`geobr`) + indicadores (Atlas Brasil)

Geometria: já resolvida via `geobr::read_municipality(code_state = "SC", year = 2020)` (seção 1.3) — sem download manual.

Indicadores (esperança de vida e saneamento):

Link: https://www.atlasbrasil.org.br/consulta

Passo a passo:

1. Acesse o link, selecione "Consulta" > escolha a Unidade Territorial "Municípios" e o estado "Santa Catarina".
2. Selecione os indicadores: `ESPVIDA` (Esperança de vida ao nascer) e um indicador de saneamento/habitação disponível na plataforma (ex.: percentual de domicílios com abastecimento de água adequado, ou

coleta de lixo adequada — o nome exato da variável pode variar entre edições do Atlas, confira na tela de seleção).

3. Exporte como planilha (`.xlsx` / `.csv`). O identificador de município no arquivo exportado deve trazer o código IBGE de 7 dígitos — confirme isso antes de tentar o join com `geobr` (mesmo aviso repetido desde o Módulo 6: confira nomes/códigos de coluna reais antes de escrever o código de join).

```
library(readxl)

indicadores_sc <- read_excel("atlas_brasil_sc.xlsx") %>%
  rename(code_muni = Codigo, esperanca_vida = ESPVIDA, saneamento = <nome_
real_da_coluna>)

municipios_sc <- municipios_sc %>%
  left_join(indicadores_sc, by = "code_muni")
```

Os dados do Atlas são baseados no Censo Demográfico de 2010 (a base municipal mais completa e amplamente usada para IDHM) — uma limitação real a declarar: não são dados correntes, mas seguem sendo a referência padrão para indicadores socioeconômicos municipais no Brasil, e servem perfeitamente ao objetivo deste módulo (ilustrar a técnica de autocorrelação/regressão espacial), mesmo que não sejam o ano mais recente disponível.

8.2 Estações INMET-SC — reaproveitadas do Módulo 6

Para as seções 5 (krigagem) e 6 (padrão de pontos), reaproveite o mesmo processo de obtenção de dados do INMET já documentado no Módulo 6 (portal.inmet.gov.br/dadoshistoricos), filtrando estações do estado de Santa Catarina e calculando a temperatura/precipitação média anual por estação como a variável a interpolar.

---

9. Diferenças-chave vindas de Python

Conceito	Python (geopandas / pysal / libpysal)	R (sf / spdep / spatialreg)
Estrutura de dados espaciais	`GeoDataFrame` (geopandas)	`sf` — mesma filosofia (geometria como coluna)
Reprojeção de CRS	`gdf.to_crs(epsg=31982)`	`st_transform(x, crs = 31982)`
Matriz de vizinhança (contiguidade)	`libpysal.weights.Queen.from_dataframe(gdf)`	`poly2nb(x, queen = TRUE)`
Matriz de vizinhança (k-vizinhos)	`libpysal.weights.KNN.from_dataframe(gdf, k=5)`	`knn2nb(knearneigh(coords, k=5))`
Moran's I global	`esda.Moran(y, w)`	`moran.test()` / `moran.mc()`
LISA	`esda.Moran_Local(y, w)`	`localmoran()`
Variograma/ krigagem	`pykrige.OrdinaryKriging`	`gstat::variogram()` + `krige()`
Padrão de pontos / CSR	`pointpats` (K de Ripley: `pointpats.k_test`)	`spatstat::Kest()` + `envelope()`
Regressão espacial (lag/ erro)	`spreg.ML_Lag` / `spreg.ML_Error` (pacote `spreg`, parte do PySAL)	`spatialreg::lagsarlm()` / `errorsarlm()`
Testes de Multiplicador de Lagrange	`spreg` reporta como parte do output de diagnóstico	`lm.LMtests()`

O padrão mais relevante para quem já usou PostGIS/Kepler.gl (como você) mas não `pysal` / `spreg`: a parte de **visualização** espacial que você já domina (Deck.gl, Kepler.gl) não tem equivalente direto de necessidade aqui — `ggplot2 + geom_sf()` cobre bem o mapa estático para relatório, mas o que este módulo adiciona é a camada de **inferência** que PostGIS/Kepler.gl não fazem: testar formalmente se um padrão visual é estatisticamente distinguível de aleatoriedade, e ajustar modelos que corrigem para dependência espacial — o "por que" estatístico por trás do mapa bonito, não o mapa em si.

---

10. Exercícios

Use `painel_regional_varejo.csv` para os exercícios 1-2 e 6, e os municípios de SC (seção 8) para os exercícios 3-5.

Exercício 1 — Moran's I: confirmando o gabarito sintético

Calcule Moran's I (via `moran.mc`, 999 permutações) para `indice_renda_media`, `ticket_medio_regional` e `var_placebo_espacial` no painel de cidades. Os resultados batem com o esperado pela construção do gerador (seção 2.5)? Reporte o *I* e o p-valor de cada variável.

Solução:

```
for (v in c("indice_renda_media", "ticket_medio_regional", "var_placebo_es
pacial")) {
  resultado <- moran.mc(cidades[[v]], pesos_knn, nsim = 999)
  cat(v, "-- I:", round(resultado$statistic, 3), " p-valor:", round(result
ado$p.value, 4), "\n")
}
```

Esperado: `indice_renda_media` e `ticket_medio_regional` com *I* positivo e p-valor < 0.05 (ambas herdam a estrutura de `polo_efeito_renda`); `var_placebo_espacial` com *I* próximo de zero e p-valor não significativo (ruído puro, sem estrutura de polo por construção).

Exercício 2 — LISA no painel sintético

Rode `localmoran()` para `indice_renda_media` no painel de cidades e classifique cada cidade em quadrante (seção 4.2). As cidades classificadas como "Alto-Alto" correspondem majoritariamente a um ou dois `polo`s específicos? Isso é esperado?

Solução: ver código da seção 4.2, adaptado para `pesos_knn` em vez de `pesos` (municípios). Esperado: sim — como `polo_efeito_renda` é sorteado uma vez por polo (não por cidade), cidades do mesmo polo com efeito positivo tendem a formar o cluster "Alto-Alto" coeso, e cidades de um polo com efeito negativo tendem a formar "Baixo-Baixo" — o LISA deveria recuperar visualmente a estrutura de 5 polos usada na geração dos dados.

Exercício 3 — Moran's I real: esperança de vida em SC

Calcule Moran's I para `esperanca_vida` nos municípios de SC (contiguidade queen). Compare o valor com o obtido no Exercício 1 para `indice_renda_media` (sintético). A magnitude é parecida? O que isso sugere sobre o quão realista foi o desenho do dataset sintético em termos de força de autocorrelação espacial?

Solução esperada (conceitual, sem gabarito numérico fixo por depender do indicador exato exportado): indicadores de saúde municipal tipicamente mostram Moran's I moderado a forte (dado que saúde pública, infraestrutura e características regionais realmente se agrupam geograficamente no Brasil) — se a magnitude for parecida com a do painel sintético, é evidência de que o desenho do dataset didático capturou razoavelmente bem a intensidade de autocorrelação espacial observada em dado real; se for muito mais fraca ou mais forte, vale registrar como limite conhecido do dataset didático.

Exercício 4 — OLS ingênuo vs. modelo espacial em SC

Rode `lm(esperanca_vida ~ saneamento)`, teste os resíduos com `lm.morantest()`, rode `lm.LMtests()` para decidir entre SAR e SEM, e ajuste o modelo indicado. O coeficiente de `saneamento` muda de magnitude ou significância entre o OLS ingênuo e o modelo espacial?

Solução: ver seções 7.1-7.4 adaptadas para os dados reais de SC. Reporte os três coeficientes (OLS, lag, erro) lado a lado numa tabela e escreva 2-3 frases de interpretação — este exercício é, propositalmente, o equivalente direto da aplicação de mestrado descrita na seção 0.

Exercício 5 — Krigagem: interpolando temperatura em SC

Baixe estações INMET de SC (seção 8.2), ajuste um variograma para temperatura média anual, e krige para uma grade cobrindo o estado. Onde a variância de predição (`var1.var`) é mais alta? Isso corresponde a áreas com menos estações próximas?

Solução: ver seção 5.3. Esperado: `var1.var` mais alta em áreas do interior/serra com menor densidade de estações (reflexo direto do padrão de pontos não aleatório discutido na seção 6.3) — a variância de krigagem funciona como um mapa direto de "onde confiar menos" na superfície interpolada.

Exercício 6 (desafio) — Sensibilidade à especificação da matriz de pesos

Repita o cálculo de Moran's I para `indice_renda_media` (painel sintético) usando três matrizes de peso diferentes: k=3, k=5 e k=10 vizinhos mais próximos. O valor de *I* e sua significância mudam de forma relevante? O que isso implica sobre a importância de justificar a escolha da matriz de pesos em qualquer análise espacial real (não só reportar "usei contiguidade" sem mais explicação)?

Solução:

```
for (k in c(3, 5, 10)) {
  pesos_k <- nb2listw(knn2nb(knearneigh(coords, k = k)), style = "W")
  resultado <- moran.mc(cidades$indice_renda_media, pesos_k, nsim = 999)
  cat("k =", k, "-- I:", round(resultado$statistic, 3), " p-valor:",
round(resultado$p.value, 4), "\n")
}
```

Esperado: a direção (positivo, significativo) deve se manter estável nos três casos, dado que a estrutura de polo é forte o suficiente para sobreviver a variações razoáveis de *k* — mas a magnitude exata de *I* tende a variar (tipicamente *k* menor captura estrutura mais local/forte, *k* maior "dilui" a vizinhança e tende a suavizar *I* para baixo). O ponto central do exercício: a matriz de pesos é uma escolha analítica com consequência real no resultado, não um passo técnico neutro — todo relatório de análise espacial deveria justificar a escolha e, idealmente, mostrar robustez a especificações alternativas, exatamente como este exercício fez.

---

11. Entrega esperada do módulo

Script `espacial_painel_sc.R` que:

1. Carrega `painel_regional_varejo.csv`, constrói matriz de pesos k-NN, e calcula Moran's I para `indice_renda_media`, `ticket_medio_regional` e `var_placebo_espacial` (verificação contra gabarito)
2. Gera mapa de cluster LISA para `indice_renda_media` no painel sintético
3. Carrega municípios de SC via `geobr`, junta com indicadores do Atlas Brasil (esperança de vida + saneamento), calcula Moran's I e gera mapa de cluster LISA
4. Roda OLS ingênuo, `lm.morantest()`, `lm.LMtests()`, e ajusta o modelo espacial indicado (lag ou erro) para os dados de SC — reportando explicitamente a mudança no coeficiente de interesse

5. Ajusta variograma e roda krigagem ordinária de uma variável climática do INMET-SC sobre uma grade cobrindo o estado
6. Testa CSR (quadrantes + função K) sobre as coordenadas das estações INMET-SC

---

12. Critério de domínio antes de encerrar o plano de 9 módulos

- ☐ Explica por que autocorrelação espacial viola a premissa de independência de OLS clássico, e qual consequência prática isso tem (erro padrão, não o coeficiente em si)
- ☐ Sabe escolher entre contiguidade e distância/k-vizinhos para construir a matriz de pesos, dado o tipo de dado (polígono vs. ponto)
- ☐ Interpreta corretamente Moran's I (sinal, magnitude, significância) e sabe diferenciar do Geary's C
- ☐ Consegue produzir e interpretar um mapa de cluster LISA, nomeando os 4 quadrantes corretamente
- ☐ Explica os três parâmetros de um variograma (alcance, patamar, pepita) e o que cada um significa fisicamente
- ☐ Sabe quando krigagem é apropriada (dado pontual, quer superfície contínua) vs. quando não é (dado já é areal/poligonal)
- ☐ Entende a diferença conceitual entre CSR de padrão de pontos e autocorrelação espacial de valores — são perguntas diferentes sobre o espaço
- ☐ Sabe escolher entre spatial lag e spatial error model usando os testes de Multiplicador de Lagrange, e explica a diferença conceitual entre os dois mecanismos

- ☐ Consegue demonstrar, com dado próprio, o efeito de pseudo-replicação espacial: uma correlação/coeficiente que muda de magnitude ao controlar por estrutura espacial
- ☐ Todos os 6 exercícios rodam sem erro, com interpretação escrita conectando cada resultado ao mecanismo espacial subjacente (não só "o Moran's I deu significativo")

---

13. Nota: decisões deste módulo e fechamento do plano de 9 módulos

1. **Escolha do par saneamento × esperança de vida em vez de ozônio × glaucoma diretamente:** dado georreferenciado de incidência de glaucoma e depleção de ozônio por município não está disponível de forma limpa e pública na granularidade exigida por este exercício (a maior parte do dado epidemiológico fino do DATASUS exige acesso restrito ou agregação que perde a variação municipal necessária para Moran's I ter poder). O par escolhido (indicador de saúde × proxy ambiental/infraestrutura, ambos do Atlas Brasil, mesma fonte, mesmo nível de granularidade) preserva a lógica metodológica central do mestrado (autocorrelação espacial entre variável ambiental e indicador de saúde) com dado verificadamente acessível — a técnica ensinada aqui é diretamente transferível para o dataset real do mestrado quando ele estiver disponível.
2. **`errorsarlm` / `lagsarlm` (estimação por máxima verossimilhança) foram preferidos a `spatialreg::sacsarlm` ou a estimação Bayesiana** — mantém o módulo no nível de complexidade compatível com o restante do plano (frequentista, sem introduzir MCMC), com nota de que modelos SAC (combinando lag e erro) existem para o caso ambíguo da seção 7.2, mas não aprofundados.

3. **Nenhuma pendência nova ficou em aberto.** Este é o módulo final do plano original de 9 módulos — fecha a sequência completa (visualização → descritiva → bivariada → não-paramétrica → multivariada → inferencial I/II → séries temporais → espacial), com o painel sintético (`painel_regional_varejo.csv`) tendo cumprido seu papel duplo (Módulo 10 temporal, Módulo 11 espacial) desde que foi desenhado.

PARTE 5

Tópicos Avançados

Quatro extensões que ampliam o alcance do que foi construído até aqui: GLM generalizado, sobrevivência com censura, inferência bayesiana e delineamento experimental.

MÓDULO 12 · TÓPICOS AVANÇADOS

GLM Generalizado

Poisson e Gamma

1. A estrutura unificada do GLM

Todo GLM tem três componentes:

1. **Componente aleatório:** a distribuição assumida para y — normal (regressão linear clássica), binomial (Módulo 5), Poisson, Gamma, entre outras. Todas pertencem à família exponencial de distribuições, o que garante que a estimação por máxima verossimilhança tenha propriedades matemáticas bem comportadas.
2. **Componente sistemático (preditor linear):** $\eta = \beta_0 + \beta_1 x_1 + \ldots + \beta_k x_k$ — sempre linear nos coeficientes, igual em toda a família GLM.
3. **Função de ligação (*link function*):** conecta o preditor linear (que pode ser qualquer número real) à média da distribuição de y (que pode ter restrições — Poisson e Gamma exigem média positiva, binomial exige média em $[0, 1]$). $g(\mu) = \eta$.

Família	Uso típico	Ligação canônica	Restrição de y
Normal (Gaussiana)	Regressão linear clássica (Módulo 4)	Identidade ($g(\mu) = \mu$)	Qualquer real
Binomial	Resposta binária (Módulo 5)	Logit	$\{0, 1\}$
Poisson	Contagens	Log	Inteiros não-negativos

Família	Uso típico	Ligação canônica	Restrição de *y*
Gamma	Contínua positiva, assimétrica	Log (mais comum na prática, embora a canônica seja o inverso)	Positivos

O ponto conceitual central: `lm()` **é** um GLM (família normal, ligação identidade) — não é uma técnica separada de `glm()`, é o caso mais simples da mesma família. Escolher a família certa é escolher a distribuição que descreve corretamente a variável resposta, exatamente como você escolheria entre Binomial e Poisson no Módulo 8 (seção 4) para modelar um fenômeno, agora aplicado a regressão.

---

2. Regressão de Poisson: modelando contagens

2.1 Por que não usar regressão linear para contagens

Contagens (`num_reclamacoes_mes`, número de casos de distocia — Módulo 8, seção 4.2) têm os mesmos três problemas estruturais já vistos no Módulo 5 para binária, adaptados: (1) previsões podem sair negativas, o que não existe para uma contagem; (2) variância cresce com a média (Var(X) = λ na Poisson) — heterocedasticidade intrínseca, não incidental; (3) a relação entre preditores e contagem esperada tipicamente é multiplicativa, não aditiva (dobrar um preditor tende a multiplicar a taxa esperada por um fator, não somar uma quantidade fixa) — a ligação log captura exatamente esse padrão.

2.2 O modelo

$$\log(\lambda_i) = \beta_0 + \beta_1 x_{1i} + \ldots + \beta_k x_{ki}, \qquad y_i \sim \text{Poisson}(\lambda_i)$$

Exponenciando: $\lambda_i = e^{\beta_0 + \beta_1 x_{1i} + \cdots}$ — o preditor linear entra na exponencial, garantindo $\lambda_i > 0$ sempre, mesmo com coeficientes negativos.

2.3 Construção da variável resposta

`num_reclamacoes_mes` (real, `varejo_sintetico.csv`) foi gerada por `rpois(n, lambda = 1.3)` — mesmo problema já documentado no Módulo 5, seção 3: sorteio independente de qualquer outra variável, sem relação real com preditores. Para ilustrar a mecânica com uma relação genuína e verificável, este módulo constrói uma contagem nova — mesmo princípio pedagógico do Módulo 5:

```
library(dplyr)

varejo <- read.csv("varejo_sintetico.csv", stringsAsFactors = TRUE)

set.seed(2026)
log_lambda_construido <- 0.3 + 0.00012 * varejo$fluxo_clientes_mensal - 0.
008 * varejo$ticket_medio
lambda_construido <- exp(log_lambda_construido)
varejo$incidentes_operacionais <- rpois(nrow(varejo), lambda =
lambda_construido)

mean(varejo$incidentes_operacionais)
var(varejo$incidentes_operacionais)   # compare com a média -- assinatura
da Poisson (seção 2.5)
```

2.4 Ajustando e interpretando

```
modelo_poisson <- glm(incidentes_operacionais ~ fluxo_clientes_mensal + ti
cket_medio,
                      data = varejo, family = poisson)
summary(modelo_poisson)

exp(coef(modelo_poisson))   # razão de taxas (rate ratio), interpretação
análoga ao odds ratio
```

Interpretação — razão de taxas (rate ratio), não odds ratio: e^{β_j} aqui é o fator multiplicativo sobre a **taxa esperada de eventos** para cada aumento de 1 unidade em x_j, mantendo os demais preditores fixos. $e^{\beta_j} = 1.10$ significa

"cada unidade adicional de x_j associa-se a um aumento de 10% na taxa esperada de incidentes" — mesma lógica multiplicativa do odds ratio (Módulo 5, seção 5), mas sobre uma taxa, não sobre uma chance.

2.5 Sobredispersão: quando a Poisson não é suficiente

A Poisson assume $E[X] = \text{Var}(X)$ (Módulo 8, seção 4.2) — no mundo real, contagens frequentemente têm variância **maior** que a média (*overdispersion*), violando essa premissa. Diagnóstico direto:

```
deviance(modelo_poisson) / df.residual(modelo_poisson)
```

Uma razão deviance/graus-de-liberdade **bem acima de 1** é sinal de sobredispersão — os erros padrão do `glm(family = poisson)` ficam artificialmente pequenos (mesmo problema conceitual da heterocedasticidade não corrigida no Módulo 4, seção 7.1: os coeficientes continuam utilizáveis, a inferência sobre eles fica errada). Duas correções comuns:

```
# Quase-Poisson: mesma estimativa de beta, erro padrão corrigido pela
dispersão estimada
modelo_quasipoisson <- glm(incidentes_operacionais ~
fluxo_clientes_mensal + ticket_medio,
                           data = varejo, family = quasipoisson)
summary(modelo_quasipoisson)

# Binomial Negativa: modela a sobredispersão explicitamente como parte da
distribuição
install.packages("MASS")   # já usado no Módulo 4 para Box-Cox
library(MASS)
modelo_binom_neg <- glm.nb(incidentes_operacionais ~
fluxo_clientes_mensal + ticket_medio, data = varejo)
summary(modelo_binom_neg)
```

`quasipoisson` é a correção mais simples (mesmos coeficientes, erro padrão maior); Binomial Negativa é estruturalmente mais correta quando a sobredispersão é substancial, porque modela a variância extra como parte da distribuição, não como um ajuste posterior sobre a Poisson.

3. Regressão Gamma: contínua positiva e assimétrica

3.1 Quando usar

Variáveis contínuas, estritamente positivas, com assimetria à direita — o mesmo perfil de variável que motivou Box-Cox no Módulo 4 (seção 7.3). `receita_mensal` é candidata natural: positiva por definição de negócio, com cauda à direita plausível (poucas lojas com receita muito alta).

```
hist(varejo$receita_mensal, main = "Distribuição de receita_mensal")
```

3.2 Diferença de abordagem: transformar *y* (Box-Cox) vs. modelar a família certa (Gamma GLM)

Box-Cox (Módulo 4, seção 7.3) mantém a estrutura de regressão linear e transforma *y* para que os resíduos fiquem aproximadamente normais e homocedásticos — o modelo passa a prever $y^{(\lambda)}$, exigindo transformação inversa para voltar à escala original (com viés de retransformação, um detalhe técnico frequentemente ignorado na prática). GLM Gamma modela diretamente *y* na escala original, assumindo que a variância cresce com o quadrado da média (assinatura da distribuição Gamma) em vez de forçar variância constante via transformação — estrutural, não um remendo sobre o modelo linear.

3.3 Ajustando no R

```
modelo_gamma <- glm(receita_mensal ~ fluxo_clientes_mensal + ticket_medio
+ area_m2,
                    data = varejo, family = Gamma(link = "log"))
summary(modelo_gamma)

exp(coef(modelo_gamma))
# fator multiplicativo sobre a média de y, mesma lógica de rate ratio
```

`link = "log"` (em vez da ligação canônica, o inverso) é a escolha prática mais comum para Gamma — coeficientes interpretáveis como efeito multiplicativo (mesma leitura da seção 2.4), e evita o problema de a ligação inversa poder gerar médias previstas negativas, que não existem para uma Gamma.

3.4 Comparando com OLS clássico

```
modelo_ols <- lm(receita_mensal ~ fluxo_clientes_mensal + ticket_medio + a
rea_m2, data = varejo)

par(mfrow = c(1, 2))
plot(modelo_ols, which = 1, main = "OLS")
plot(modelo_gamma, which = 1, main = "Gamma GLM")
par(mfrow = c(1, 1))
```

Se `receita_mensal` já não for severamente assimétrica neste dataset (o Módulo 4 não sinalizou heterocedasticidade extrema — seção 7.1 daquele módulo), espera-se que os dois modelos produzam diagnósticos parecidos, e a escolha entre eles importa menos na prática. A lição not deste dataset especificamente, mas geral: Gamma GLM ganha relevância clara quando a assimetria é mais pronunciada (dados de sinistro/seguro, tempo de permanência, gasto de saúde) — vale ter a ferramenta disponível mesmo quando este dataset específico não força a mão.

---

4. Deviance: a métrica de ajuste do GLM

R^2 (Módulo 4, seção 4) não se generaliza diretamente para fora da família normal. A métrica análoga em qualquer GLM é a **deviance** — duas vezes a diferença de log-verossimilhança entre o modelo saturado (um parâmetro por observação, ajuste perfeito) e o modelo ajustado:

```
summary(modelo_poisson)$deviance          # deviance residual -- menor é
melhor ajuste
summary(modelo_poisson)$null.deviance     # deviance de um modelo só com
intercepto
```

Pseudo-R^2 (McFadden), um análogo aproximado ao R^2:

$$ R^2_{McFadden} = 1 - \frac{\text{deviance residual}}{\text{deviance nula}} $$

```
1 - (modelo_poisson$deviance / modelo_poisson$null.deviance)
```

Não tem a mesma interpretação direta de "proporção de variância explicada" do R^2 da regressão linear — trate como métrica comparativa entre modelos GLM aninhados (igual R^2 ajustado no Módulo 4, nunca compare pseudo-R^2 entre famílias diferentes), não como número com significado absoluto isolado.

---

5. Diferenças-chave vindas de Python

Conceito	Python (statsmodels)	R
GLM genérico	`statsmodels.api.GLM(y, X, family=sm.families.Poisson())`	`glm(y ~ x, family = poisson)`
Poisson	`sm.families.Poisson()`	`family = poisson`
Gamma	`sm.families.Gamma(link=sm.families.links.log())`	`family = Gamma(link = "log")`
Binomial Negativa	`statsmodels.discrete.discrete_model.NegativeBinomial`	`MASS::glm.nb()`

Conceito	Python (statsmodels)	R
Quase-Poisson	`sm.families.Poisson()` + `cov_type='HC1'` (aproximação, não idêntico)	`family = quasipoisson` (mais direto)
Deviance	`model.deviance`	`model$deviance`
Razão de taxa / rate ratio	`np.exp(model.params)` (manual)	`exp(coef(model))`

O mesmo padrão já visto nos Módulos 4 e 5: R trata `glm()` como uma família unificada de primeira classe, com `family =` trocando a distribuição sem mudar a sintaxe de fórmula; Python distribui a funcionalidade entre `statsmodels` (inferência completa) e implementações específicas por família (`NegativeBinomial` é uma classe separada, não um parâmetro de família dentro de `GLM`).

---

6. Exercícios

Exercício 1 — Regressão linear vs. Poisson em contagem

Ajuste `lm(incidentes_operacionais ~ fluxo_clientes_mensal, data = varejo)` e `glm(..., family = poisson)`. Compare os valores previstos mínimos de cada modelo — o linear produz alguma previsão negativa?

Solução:

```
mod_lin <- lm(incidentes_operacionais ~ fluxo_clientes_mensal, data =
varejo)
mod_poi <- glm(incidentes_operacionais ~ fluxo_clientes_mensal, data =
varejo, family = poisson)

min(fitted(mod_lin))
min(fitted(mod_poi))
```

Dependendo do range de `fluxo_clientes_mensal` observado, o modelo linear pode produzir previsão negativa para fluxo baixo — sem sentido para uma contagem. O modelo de Poisson, por construção da ligação log, nunca produz previsão negativa.

Exercício 2 — Diagnosticando sobredispersão

Calcule a razão deviance/graus-de-liberdade de `modelo_poisson` (seção 2.4). Há sinal de sobredispersão? Reajuste com `quasipoisson` e compare o erro padrão de `fluxo_clientes_mensal` entre os dois modelos.

Solução:

```
deviance(modelo_poisson) / df.residual(modelo_poisson)

modelo_qp <- glm(incidentes_operacionais ~ fluxo_clientes_mensal + ticket_
medio,
                 data = varejo, family = quasipoisson)

summary(modelo_poisson)$coefficients["fluxo_clientes_mensal", "Std.
Error"]
summary(modelo_qp)$coefficients["fluxo_clientes_mensal", "Std. Error"]
```

Como `incidentes_operacionais` foi construída diretamente de uma Poisson pura (sem fonte adicional de variância além da própria distribuição), espera-se razão deviance/gl próxima de 1 — pouca ou nenhuma sobredispersão, e os erros padrão de `quasipoisson` devem ficar próximos dos da Poisson comum. Em dado real de contagem, é comum essa razão sair bem acima de 1.

Exercício 3 — Gamma vs. OLS: coeficientes e interpretação

Ajuste `modelo_gamma` e `modelo_ols` (seção 3) com os mesmos preditores. Compare o sinal e a significância de cada coeficiente entre os dois modelos. A conclusão qualitativa (quais preditores importam) muda entre as duas abordagens?

Solução: ver seções 3.3-3.4 para o código-base; compare `summary(modelo_gamma)$coefficients` com `summary(modelo_ols)$coefficients`. Dado que os preditores usados (`fluxo_clientes_mensal`, `ticket_medio`, `area_m2`) têm relação real e forte com `receita_mensal` por construção do dataset (Módulo 4), espera-se que ambos os modelos concordem na direção e significância dos efeitos — a diferença entre as duas abordagens aparece mais na forma como a incerteza é modelada (variância constante vs. crescente com a média) do que na conclusão qualitativa sobre quais preditores importam.

Exercício 4 (desafio) — Pseudo-R² e comparação de modelos aninhados

Ajuste um `modelo_poisson_simples` (só `fluxo_clientes_mensal`) e o `modelo_poisson` completo (seção 2.4, com `ticket_medio` também). Calcule o pseudo-R² de McFadden dos dois. Rode `anova(modelo_poisson_simples, modelo_poisson, test = "Chisq")` — o preditor adicional melhora o ajuste de forma estatisticamente distinguível?

Solução:

```
modelo_poisson_simples <- glm(incidentes_operacionais ~ fluxo_clientes_men
sal, data = varejo, family = poisson)

1 - (modelo_poisson_simples$deviance / modelo_poisson_simples$null.devianc
e)
1 - (modelo_poisson$deviance / modelo_poisson$null.deviance)

anova(modelo_poisson_simples, modelo_poisson, test = "Chisq")
```

O pseudo-R² do modelo completo deve ser maior (mesma lógica de "adicionar preditor nunca piora o ajuste dentro da amostra", já vista no Módulo 4, seção 4.3, e no Módulo 5, Exercício 4). O teste qui-quadrado de deviances aninhadas é o análogo do `anova()` sobre modelos `lm` (Módulo 4, seção 5.5) — um p-valor baixo indica que `ticket_medio` contribui além do

que se explicaria só por acaso amostral, consistente com a forma como `incidentes_operacionais` foi construída (seção 2.3, com `ticket_medio` explicitamente no preditor linear gerador).

---

7. Entrega esperada do módulo

Script `glm_generalizado_varejo.R` que recebe `varejo_sintetico.csv` e produz:

1. Construção de `incidentes_operacionais` (seção 2.3) com `set.seed(2026)` fixo
2. Modelo Poisson ajustado, com razão de taxas (rate ratio) e diagnóstico de sobredispersão (deviance/gl)
3. Modelo quase-Poisson ou Binomial Negativa, comparando erro padrão com o modelo Poisson simples
4. Modelo Gamma para `receita_mensal`, comparado lado a lado com o `lm()` equivalente
5. Pseudo-R^2 de McFadden reportado para pelo menos dois modelos comparáveis

---

8. Critério de domínio

- ☐ Explica a estrutura de três componentes do GLM (aleatório, sistemático, ligação) e localiza `lm()` e `glm(family=binomial)` (Módulo 5) dentro dela
- ☐ Sabe por que a ligação log é a escolha natural para Poisson e Gamma, e interpreta coeficientes exponenciados como razão de taxa/efeito multiplicativo
- ☐ Diagnostica sobredispersão via razão deviance/gl e sabe escolher entre quase-Poisson e Binomial Negativa
- ☐

Entende a diferença conceitual entre transformar y (Box-Cox, Módulo 4) e modelar a família de distribuição correta (Gamma GLM)

- ☐ Sabe que pseudo-R^2 de McFadden não tem a mesma interpretação de "proporção de variância explicada" do R^2 clássico
- ☐ Todos os 4 exercícios rodam sem erro e a interpretação conecta o resultado à mecânica do GLM, não só reporta o número

---

MÓDULO 13 · TÓPICOS AVANÇADOS

Análise de Sobrevivência

1. Censura: o problema que só a análise de sobrevivência resolve corretamente

Censura à direita (o caso mais comum): a observação não sofreu o evento até o fim do período observado — não significa "nunca vai sofrer o evento", significa "não sabemos quando, só sabemos que ainda não aconteceu até aquele momento". Exemplos no seu domínio: uma loja que continua aberta ao fim do período de estudo (não sabemos quando vai fechar, se é que vai); um equipamento que continua funcionando quando a coleta de dados termina.

Por que ignorar censura é um erro grave, não um detalhe técnico: duas abordagens ingênuas, ambas erradas: 1. **Descartar as observações censuradas** (listwise deletion, o mesmo erro do Módulo 3, seção 3, aplicado a este contexto) — joga fora exatamente as observações que sobreviveram mais tempo (por definição, ainda não sofreram o evento), enviesando a amostra restante para os casos de sobrevivência mais curta. 2. **Tratar o tempo censurado como se fosse o tempo real do evento** — subestima sistematicamente o tempo até o evento, porque o tempo real (não observado) é necessariamente maior que o tempo censurado registrado.

A análise de sobrevivência usa toda a informação disponível de cada observação — inclusive as censuradas, que contribuem "sei que sobreviveu **pelo menos** até aqui" — sem descartar nem distorcer.

2. Construção dos dados para este módulo

Nenhum dataset já usado no plano tem estrutura de tempo até evento com censura. Este módulo constrói um cenário sintético grounded no domínio agropecuário — tempo até uma loja fictícia encerrar operação —, com relação conhecida a covariáveis, seguindo o mesmo princípio de transparência já usado nos Módulos 5 e 12:

```
library(dplyr)
library(survival)

set.seed(2026)
n <- 1300

porte <- factor(sample(c("Pequena", "Media", "Grande"), n, replace =
TRUE, prob = c(0.5, 0.35, 0.15)),
                levels = c("Pequena", "Media", "Grande"))
fluxo_clientes_mensal <- round(rnorm(n, mean = 4000, sd = 1500))
fluxo_clientes_mensal[fluxo_clientes_mensal < 500] <- 500

# Tempo "verdadeiro" até o evento (fechamento), via distribuição
exponencial com taxa
# dependente de covariáveis -- lojas maiores e com mais fluxo sobrevivem
mais tempo
taxa_risco <- exp(-4.5 - 0.00025 * fluxo_clientes_mensal -
                    0.6 * (porte == "Media") - 1.2 * (porte == "Grande"))
tempo_verdadeiro <- rexp(n, rate = taxa_risco)   # em meses

periodo_observacao <- 36   # estudo acompanhado por 36 meses
tempo_observado <- pmin(tempo_verdadeiro, periodo_observacao)
evento <- as.integer(tempo_verdadeiro <= periodo_observacao)   # 1 =
fechou, 0 = censurado

lojas_sobrevivencia <- data.frame(
  loja_id = sprintf("LJ%04d", seq_len(n)),
  porte, fluxo_clientes_mensal,
  tempo = round(tempo_observado, 1),
  evento
)

mean(lojas_sobrevivencia$evento)   # proporção de lojas que "fecharam"
dentro do período observado
```

`rexp()` com taxa dependente de covariáveis é a forma mais simples de simular tempo até evento com estrutura conhecida (a taxa de risco constante ao longo do tempo é a premissa da distribuição exponencial — uma simplificação didática; dado real raramente tem risco constante, seção 5 trata disso). `pmin(tempo_verdadeiro, periodo_observacao)` implementa a censura: qualquer tempo verdadeiro além do período de observação vira censura no tempo do fim do estudo.

---

3. Kaplan-Meier: estimando a curva de sobrevivência

3.1 O estimador

Kaplan-Meier estima $S(t) = P(\text{sobreviver além de } t)$ de forma não-paramétrica — sem assumir nenhuma distribuição para o tempo até evento —, usando corretamente tanto eventos quanto censuras.

```
ajuste_km <- survfit(Surv(tempo, evento) ~ 1, data = lojas_sobrevivencia)
summary(ajuste_km)

plot(ajuste_km, xlab = "Meses", ylab = "P(sobrevivência)",
     main = "Curva de sobrevivência (Kaplan-Meier) — todas as lojas")
```

`Surv(tempo, evento)` empacota as duas informações essenciais (tempo observado + indicador de evento/censura) no formato que todas as funções do pacote `survival` esperam — o mesmo papel estrutural que `y ~ x` tem para `lm()` / `glm()`, mas com uma resposta que carrega duas colunas em vez de uma.

3.2 Curvas por grupo

```
ajuste_km_porte <- survfit(Surv(tempo, evento) ~ porte, data =
lojas_sobrevivencia)

plot(ajuste_km_porte, col = c("firebrick", "steelblue", "forestgreen"),
     xlab = "Meses", ylab = "P(sobrevivência)",
     main = "Curva de sobrevivência por porte")
legend("bottomleft", legend = levels(lojas_sobrevivencia$porte),
       col = c("firebrick", "steelblue", "forestgreen"), lty = 1)
```

Dado como `taxa_risco` foi construída (seção 2), espera-se que a curva de "Grande" fique consistentemente acima das demais (sobrevivência mais longa) — a leitura visual direta antecipa o que o log-rank (seção 3.3) vai confirmar formalmente.

3.3 Log-rank test: comparação formal entre curvas

Equivalente, para dado censurado, do teste t/ANOVA (Módulo 9) para comparar grupos — mas operando sobre toda a curva de sobrevivência, não sobre uma média simples:

```
survdiff(Surv(tempo, evento) ~ porte, data = lojas_sobrevivencia)
```

H_0: as curvas de sobrevivência são iguais entre os grupos. Mesma lógica de decisão do Módulo 9 (seção 3): p-valor baixo rejeita H_0, evidência de que ao menos um grupo tem sobrevivência sistematicamente diferente. O log-rank não diz *quanto* a sobrevivência difere em unidade interpretável — isso é o papel do modelo de Cox (seção 4).

---

4. Modelo de Cox: regressão para tempo até evento

4.1 O modelo (riscos proporcionais)

$$h(t \mid x) = h_0(t) \cdot e^{\beta_1 x_1 + \ldots + \beta_k x_k}$$

$h(t)$ é a **função de risco** (*hazard*) — taxa instantânea de o evento ocorrer no instante t, dado que sobreviveu até ali. $h_0(t)$ é o risco basal (não especificado, o que torna Cox um modelo **semi-paramétrico** — não assume uma forma para como o risco muda ao longo do tempo, só que o efeito multiplicativo das covariáveis é constante ao longo do tempo, a premissa de **riscos proporcionais**, seção 4.3).

```
modelo_cox <- coxph(Surv(tempo, evento) ~ porte + fluxo_clientes_mensal, d
ata = lojas_sobrevivencia)
summary(modelo_cox)
```

4.2 Interpretando o hazard ratio

`summary(modelo_cox)` retorna `exp(coef)` diretamente — o **hazard ratio** (HR), mesma estrutura de interpretação do odds ratio (Módulo 5, seção 5) e da razão de taxas (Módulo 12, seção 2.4):

- $HR = 1$: sem efeito sobre o risco.
- $HR > 1$: aumenta o risco do evento (sobrevivência **menor**).
- $HR < 1$: reduz o risco do evento (sobrevivência **maior**).

Para `porte`, um $HR_{Grande} < 1$ (esperado, dado como a taxa de risco foi construída) significa "lojas Grandes têm risco de fechamento X% menor que lojas Pequenas (referência), mantendo fluxo de clientes constante" — mesma estrutura de frase do Módulo 5, seção 5, trocando "chance" por "risco instantâneo".

4.3 Premissa de riscos proporcionais: verificação

O modelo assume que o HR de cada preditor é **constante ao longo do tempo** — uma loja Grande tem o mesmo risco relativo em relação a uma Pequena no mês 2 e no mês 30. Isso é testável:

```
cox.zph(modelo_cox)
plot(cox.zph(modelo_cox))
```

H_0 do teste: a premissa de riscos proporcionais se sustenta para aquele preditor. P-valor baixo é sinal de alerta — o efeito daquele preditor pode estar mudando ao longo do tempo, o que violaria a premissa central do modelo de Cox e exigiria uma extensão (interação com o tempo, ou estratificação — fora do escopo introdutório deste módulo).

---

5. Limitação da simulação deste módulo: risco constante

A construção da seção 2 usa `rexp()`, que implica risco **constante** ao longo do tempo (a taxa de "fechar este mês, dado que sobreviveu até aqui" não muda com a idade da loja) — uma simplificação conveniente para gerar dado didático, mas raramente realista. Dado real de sobrevivência tipicamente tem risco que muda com o tempo (risco de falha de equipamento cresce com o uso; risco de recidiva de doença muda por fase do tratamento) — o que exigiria uma distribuição geradora mais rica (Weibull, por exemplo) para simular com mais realismo. Isso não invalida o uso pedagógico de Kaplan-Meier e Cox aqui (ambos não assumem risco constante ao ajustar — só a simulação que gerou o dado assume isso), mas vale registrar a diferença entre "o método que você está aprendendo" e "a forma simplificada como os dados de exemplo foram gerados".

---

6. Diferenças-chave vindas de Python

Conceito	Python (lifelines)	R
Estrutura de dados de sobrevivência	Colunas separadas `duration`, `event_observed`	`Surv(tempo, evento)` — empacotado num objeto único
Kaplan-Meier	`lifelines.KaplanMeierFitter().fit(duration, event)`	`survfit(Surv(tempo, evento) ~ 1)`
Log-rank	`lifelines.statistics.logrank_test()`	`survdiff(Surv(tempo, evento) ~ grupo)`
Cox	`lifelines.CoxPHFitter().fit(df, duration_col=..., event_col=...)`	`coxph(Surv(tempo, evento) ~ x)`
Hazard ratio	`cph.hazard_ratios_`	`exp(coef(modelo_cox))`
Verificação de riscos proporcionais	`cph.check_assumptions(df)`	`cox.zph(modelo_cox)`

`lifelines` é o pacote de referência em Python para sobrevivência — API orientada a objetos (`.fit()`), mais próxima do estilo `sklearn` que da tradição de fórmula estatística do R. `survival` (R) é o pacote-base há décadas, com a vantagem de `Surv()` empacotar tempo+evento de forma que qualquer função subsequente do pacote reconhece automaticamente, sem precisar repetir os dois argumentos em toda chamada.

---

7. Exercícios

Exercício 1 — Kaplan-Meier: mediana de sobrevivência

A partir de `ajuste_km` (seção 3.1), extraia a mediana de sobrevivência estimada (tempo em que $S(t) = 0.5$). Repita para `ajuste_km_porte`, por grupo. A mediana de "Grande" é maior que a de "Pequena", como esperado?

Solução:

```
print(ajuste_km)          # coluna "median" no output resumido
print(ajuste_km_porte)
```

A mediana de sobrevivência de "Grande" deve aparecer maior que a de "Pequena" (ou não estimável/ `NA`, se a curva daquele grupo não cruzar $S(t) = 0.5$ dentro do período observado — o que por si só já é evidência de sobrevivência longa: menos da metade do grupo sofreu o evento durante todo o estudo).

Exercício 2 — Log-rank e Cox: mesma pergunta, duas ferramentas

Rode `survdiff()` para `porte` (seção 3.3) e `coxph()` com `porte` como único preditor. O log-rank rejeita H_0? O HR de `porte` no Cox é consistente com a direção que o log-rank sugere?

Solução:

```
survdiff(Surv(tempo, evento) ~ porte, data = lojas_sobrevivencia)

modelo_cox_porte <- coxph(Surv(tempo, evento) ~ porte, data =
lojas_sobrevivencia)
summary(modelo_cox_porte)
```

Espera-se p-valor baixo no log-rank (rejeita igualdade entre curvas) e HR < 1 para `porteMedia` e, mais fortemente, para `porteGrande` no Cox — os dois testes devem contar a mesma história porque estão testando, com ferramentas diferentes, o mesmo padrão real construído na simulação (seção 2): lojas maiores têm risco de fechamento menor.

Exercício 3 — Verificação de riscos proporcionais

Rode `cox.zph()` sobre `modelo_cox` (seção 4.1). Algum preditor viola a premissa de riscos proporcionais (p-valor baixo no teste)? Isso é esperado ou não, dado como os dados foram simulados (seção 5)?

Solução:

```
cox.zph(modelo_cox)
```

Como a simulação (seção 2) usa `rexp()` com taxa de risco **constante** ao longo do tempo por construção, é esperado que `cox.zph()` **não** rejeite a premissa de riscos proporcionais para nenhum preditor (p-valores altos) — o dado foi gerado exatamente sob a premissa que o teste verifica. Em dado real, onde o risco tipicamente muda com o tempo (seção 5), rejeitar a premissa para algum preditor seria comum, não um sinal de erro de modelagem.

Exercício 4 (desafio) — Efeito da censura administrativa

Reajuste a simulação da seção 2 com `periodo_observacao = 12` (em vez de 36) e refaça o Kaplan-Meier geral. Compare a proporção de eventos observados (`mean(evento)`) e a mediana estimada de sobrevivência entre os dois cenários (36 meses vs. 12 meses de acompanhamento). O que isso demonstra sobre a relação entre duração do estudo e proporção de censura?

Solução:

```
periodo_observacao_curto <- 12
tempo_observado_curto <- pmin(tempo_verdadeiro, periodo_observacao_curto)
evento_curto <- as.integer(tempo_verdadeiro <= periodo_observacao_curto)

mean(evento_curto)   # compare com mean(lojas_sobrevivencia$evento),
período de 36 meses

ajuste_km_curto <- survfit(Surv(tempo_observado_curto, evento_curto) ~ 1)
print(ajuste_km_curto)
```

Com período de observação mais curto, a proporção de eventos observados cai (mais lojas ainda "sobrevivendo" ao fim do estudo, mesmo que fossem fechar depois) — mais censura, menos informação de evento efetivamente capturada. A mediana estimada pode ficar menos precisa ou não-estimável, dependendo de quantos eventos ocorreram dentro da janela mais curta. Isso ilustra por que a duração do acompanhamento é uma decisão de desenho de estudo com impacto direto na precisão das estimativas — não um parâmetro neutro.

---

8. Entrega esperada do módulo

Script `sobrevivencia_lojas.R` que:

1. Constrói o dataset de sobrevivência (seção 2) com `set.seed(2026)` fixo
2. Ajusta Kaplan-Meier geral e por `porte`, com gráfico comparativo
3. Roda log-rank test entre grupos de `porte`
4. Ajusta modelo de Cox com pelo menos 2 preditores, reportando hazard ratio e IC
5. Roda `cox.zph()` e reporta se a premissa de riscos proporcionais se sustenta

---

9. Critério de domínio

- ☐ Explica o que é censura à direita e por que descartar ou tratar como evento real são ambos errados
- ☐ Sabe construir e ler uma curva de Kaplan-Meier, inclusive extrair a mediana de sobrevivência
- ☐ Roda e interpreta log-rank test para comparar sobrevivência entre grupos
- ☐ Interpreta um hazard ratio corretamente, na mesma família de raciocínio de odds ratio (Módulo 5) e rate ratio (Módulo 12)
- ☐ Sabe verificar a premissa de riscos proporcionais via `cox.zph()` e explicar o que fazer se ela falhar
- ☐ Todos os 4 exercícios rodam sem erro e a interpretação conecta o resultado à estrutura de censura dos dados

---

MÓDULO 14 · TÓPICOS AVANÇADOS

Estatística Bayesiana

Introdução

1. Prior, verossimilhança, posterior

$$ \underbrace{P(\theta \mid \text{dados})}_{\text{posterior}} \propto \underbrace{P(\text{dados} \mid \theta)}_{\text{verossimilhança}} \times \underbrace{P(\theta)}_{\text{prior}} $$

- **Prior** $P(\theta)$: sua crença sobre o parâmetro **antes** de ver os dados — pode vir de conhecimento de domínio, estudos anteriores, ou ser deliberadamente vago/não-informativo quando você quer que os dados dominem a conclusão.
- **Verossimilhança** $P(\text{dados} \mid \theta)$: a mesma função de verossimilhança usada em estimação por máxima verossimilhança (Módulo 5, seção 2.2) — quão prováveis são os dados observados, para cada valor possível de θ.
- **Posterior** $P(\theta \mid \text{dados})$: a crença **atualizada** sobre θ depois de incorporar os dados — o resultado final de uma análise Bayesiana, uma distribuição de probabilidade inteira sobre θ, não um único número.

Esta é exatamente a fórmula do Módulo 8, seção 2.2, com $A = \theta$ e $B =$ dados — a mesma estrutura lógica, aplicada não a um evento pontual (doente/sadio) mas a um parâmetro contínuo (uma média, uma proporção, um coeficiente de regressão).

---

2. Exemplo conjugado: Beta-Binomial para uma proporção

O caso mais simples de se calcular analiticamente, sem precisar de simulação — bom ponto de entrada antes de `brms` (seção 4).

2.1 O cenário

Você quer estimar a taxa de concepção em IATF (Módulo 8, seção 4.1) usando um prior informado por literatura veterinária anterior, atualizado com dados de um novo lote.

Prior: Beta(α_0, β_0) — a distribuição conjugada natural para uma proporção (o parâmetro da Binomial). Literatura anterior sugere taxa em torno de 45%, com incerteza moderada:

```
alpha_prior <- 9
beta_prior <- 11    # média do prior = alpha/(alpha+beta) = 9/20 = 0.45

curve(dbeta(x, alpha_prior, beta_prior), from = 0, to = 1,
      main = "Prior: Beta(9, 11)", xlab = "Taxa de concepção", ylab = "Den
sidade")
```

2.2 Atualizando com dados novos

Propriedade da conjugação Beta-Binomial: se o prior é Beta(α_0, β_0) e você observa k sucessos em n tentativas, o posterior é, **analiticamente, sem nenhuma simulação**, Beta($\alpha_0 + k, \beta_0 + n - k$):

```
n_novo <- 50
k_novo <- 27   # 27 vacas prenhas em 50 inseminadas

alpha_posterior <- alpha_prior + k_novo
beta_posterior <- beta_prior + (n_novo - k_novo)

curve(dbeta(x, alpha_prior, beta_prior), from = 0, to = 1, col =
"gray50", lty = 2,
      main = "Prior (cinza) vs. Posterior (azul)", xlab = "Taxa de
concepção", ylab = "Densidade")
curve(dbeta(x, alpha_posterior, beta_posterior), add = TRUE, col = "steelb
lue", lwd = 2)
```

O posterior fica entre o prior e a taxa observada nos dados novos (27/50 = 0.54) — puxado na direção dos dados, mas ainda ancorado pelo prior. Quanto mais dados novos entrarem, menos o prior pesa relativamente (o mesmo princípio de "a amostra domina conforme *n* cresce" já visto informalmente no TLC, Módulo 8, seção 7) — um prior fraco (poucos "pseudo-dados" implícitos, como $\alpha_0 + \beta_0$ pequeno) é dominado rapidamente; um prior forte exige mais dados novos para mudar de opinião de forma substancial.

2.3 Ponto estimado e intervalo de credibilidade

```
media_posterior <- alpha_posterior / (alpha_posterior + beta_posterior)
media_posterior

qbeta(c(0.025, 0.975), alpha_posterior, beta_posterior)   # intervalo de
credibilidade de 95%
```

---

3. Intervalo de credibilidade vs. intervalo de confiança: a diferença real

Interpretação do intervalo de credibilidade Bayesiano (seção 2.3): "dado o prior e os dados observados, há 95% de probabilidade de que a taxa de concepção verdadeira esteja neste intervalo." Uma afirmação de probabilidade direta sobre o parâmetro — exatamente a interpretação que a

maioria das pessoas **erroneamente** atribui ao intervalo de confiança frequentista (Módulo 8, seção 9.1 e seção 9.5, que formalizou por que essa interpretação está errada no caso frequentista).

Por que a interpretação Bayesiana é válida e a frequentista não: no arcabouço frequentista, μ é um número fixo — "95% de probabilidade de μ estar no intervalo" não faz sentido matemático, porque μ não é aleatório (Módulo 8, seção 0). No arcabouço Bayesiano, θ **é** tratado como uma variável aleatória com distribuição (o posterior) — então "95% de probabilidade de θ estar neste intervalo" é uma afirmação matematicamente coerente sobre essa distribuição.

```
# Comparação lado a lado com o intervalo frequentista equivalente
(prop.test, Módulo 8, seção 9.4)
prop.test(k_novo, n_novo)$conf.int
qbeta(c(0.025, 0.975), alpha_posterior, beta_posterior)
```

Os dois intervalos costumam ficar numericamente próximos quando o prior é fraco e n é razoável (o posterior Bayesiano converge para a mesma verossimilhança que sustenta o IC frequentista) — a diferença não está tipicamente na posição/largura do intervalo, está na **interpretação válida** de cada um.

---

4. Além do conjugado: MCMC e `brms`

A conjugação Beta-Binomial (seção 2) só funciona porque a combinação prior+verossimilhança tem solução analítica fechada. Para a maioria dos modelos do mundo real (regressão com múltiplos preditores, hierárquicos, não-lineares), não existe fórmula fechada para o posterior — a solução é aproximá-lo por simulação, via **MCMC** (*Markov Chain Monte Carlo*): um algoritmo que gera milhares de amostras de valores plausíveis de θ, cuja distribuição empírica aproxima o posterior real.

`brms` é a interface R mais acessível para MCMC via Stan (o motor de amostragem por trás), com sintaxe de fórmula quase idêntica a `lm()` / `glm()`:

```
install.packages("brms")   # nota: instala e compila Stan por baixo --
pode levar vários minutos na primeira vez
library(brms)
```

Aviso prático, não cosmético: `brms` depende de compilação C++ via Stan — a instalação é bem mais pesada que qualquer pacote usado até aqui no plano (`car`, `lmtest`, `pwr` são instalação trivial; `brms` não). Rode a instalação com tempo disponível, fora de uma sessão em que você precisa de resultado imediato.

---

5. Regressão Bayesiana com `brms`: comparando com o Módulo 4

Mesma fórmula, mesmos dados do Módulo 4 (regressão linear simples), agora do lado Bayesiano:

```
varejo <- read.csv("varejo_sintetico.csv", stringsAsFactors = TRUE)

modelo_freq <- lm(receita_mensal ~ fluxo_clientes_mensal, data = varejo)

modelo_bayes <- brm(receita_mensal ~ fluxo_clientes_mensal, data = varejo,
                    family = gaussian(), chains = 4, iter = 2000, seed =
2026)

summary(modelo_freq)
summary(modelo_bayes)
```

O que muda na leitura do output: onde `summary(lm())` reporta `Estimate`, `Std. Error`, `t value`, `Pr(>|t|)`, `summary(brm())` reporta `Estimate` (a média do posterior, não uma estimativa pontual única de máxima verossimilhança), `Est.Error` (desvio padrão do posterior, papel análogo ao erro padrão), e `l-95% CI` / `u-95% CI` (limites do intervalo de **credibilidade**, não confiança — seção 3). Não há p-valor no output padrão do `brms` — a pergunta “esse

coeficiente é distinguível de zero" é respondida verificando se o intervalo de credibilidade inclui zero, uma leitura direta sobre a distribuição posterior do coeficiente, sem a maquinaria de hipótese nula/alternativa do Módulo 9.

Com prior não-informativo (o default do `brms` para a maioria dos parâmetros) e *n* grande, os coeficientes de `modelo_freq` e `modelo_bayes` devem ficar muito próximos — o mesmo padrão da seção 3: a diferença filosófica entre as duas abordagens não implica necessariamente números muito diferentes quando o prior não domina a conclusão.

---

6. Diferenças-chave vindas de Python

Conceito	Python (PyMC)	R
Modelo Bayesiano genérico	`pymc.Model()` + especificação manual de prior/verossimilhança	`brm(formula, family, prior=...)` — sintaxe de fórmula, mais próxima de `lm()`
Amostragem MCMC	`pymc.sample()`	Automático dentro de `brm()` (chama Stan por baixo)
Conjugação Beta-Binomial	`scipy.stats.beta` (cálculo manual, igual seção 2)	`dbeta` / `qbeta` / `pbeta` (mesmo padrão d/p/q/r do R)
Resumo do posterior	`arviz.summary(trace)`	`summary(modelo_brm)`
Diagnóstico de convergência MCMC	`arviz.plot_trace()`, `Rhat`	`plot(modelo_brm)`, `Rhat` no `summary()`

PyMC é o pacote de referência em Python, com filosofia parecida ao `brms` (especificação declarativa de modelo, amostragem automática) — a diferença mais notável na prática é que `brms` aproveita a mesma sintaxe de fórmula que `lm()` / `glm()` já usam desde o Módulo 4, tornando a transição

conceitualmente mais suave para quem já percorreu o resto deste plano; PyMC tem sintaxe própria, mais explícita sobre a estrutura do modelo mas mais distante da tradição `y ~ x`.

---

7. Exercícios

Exercício 1 — Prior forte vs. prior fraco

Repita a atualização Beta-Binomial da seção 2.2, mas com um prior muito mais fraco: `alpha_prior <- 1`, `beta_prior <- 1` (uniforme, "sem opinião prévia"). Compare a média posterior resultante com a da seção 2.2 (prior mais informativo). Qual dos dois posteriores fica mais próximo da proporção observada nos dados novos (27/50)?

Solução:

```
alpha_prior_fraco <- 1
beta_prior_fraco <- 1

alpha_post_fraco <- alpha_prior_fraco + k_novo
beta_post_fraco <- beta_prior_fraco + (n_novo - k_novo)

alpha_post_fraco / (alpha_post_fraco + beta_post_fraco)
alpha_posterior / (alpha_posterior + beta_posterior)   # seção 2.3, prior
mais informativo
```

O posterior com prior uniforme deve ficar bem mais próximo de 27/50 = 0.54 — sem opinião prévia, o posterior é dominado quase inteiramente pelos dados observados. O posterior da seção 2.2 (prior Beta(9,11), média 0.45) fica numa posição intermediária entre o prior e os dados — demonstração direta de como a força do prior afeta o resultado final, o ponto central da seção 2.2.

Exercício 2 — Intervalo de credibilidade vs. confiança, lado a lado

Para os dados da seção 2 (27/50), calcule o IC frequentista via `prop.test()` e o intervalo de credibilidade Bayesiano (prior da seção 2.2) via `qbeta()`. Os dois intervalos se sobrepõem substancialmente? Escreva a interpretação correta de cada um, destacando a diferença de linguagem.

Solução: ver seção 3 para o código-base. A interpretação do IC frequentista deve seguir a linguagem do Módulo 8 (seção 9.1: propriedade do procedimento repetido); a do intervalo de credibilidade deve afirmar probabilidade diretamente sobre o parâmetro (seção 3). Espera-se sobreposição substancial numericamente, mesmo com a diferença de interpretação sendo real e não apenas semântica.

Exercício 3 — Regressão Bayesiana: intervalo de credibilidade do coeficiente

A partir de `modelo_bayes` (seção 5), extraia o intervalo de credibilidade de 95% do coeficiente de `fluxo_clientes_mensal`. Ele inclui zero? Compare com o teste de significância (`Pr(>|t|)`) do `modelo_freq` equivalente — as duas abordagens levam à mesma conclusão sobre se o preditor "importa"?

Solução:

```
fixef(modelo_bayes)   # inclui Estimate, Est.Error, e os limites do
intervalo de credibilidade
summary(modelo_freq)$coefficients["fluxo_clientes_mensal", ]
```

Dado que `fluxo_clientes_mensal` tem relação forte e real com `receita_mensal` neste dataset (Módulo 4), espera-se que o intervalo de credibilidade fique inteiramente acima de zero (mesmo sinal de "efeito claramente presente" que o p-valor baixíssimo do modelo frequentista já indicava) — as duas abordagens concordam na conclusão prática, ainda que cheguem lá por caminhos de inferência filosoficamente diferentes.

Exercício 4 (desafio) — Sensibilidade ao prior em amostra pequena

Repita a atualização Beta-Binomial (seção 2) para uma amostra bem menor: $n = 5$, $k = 3$ (3 sucessos em 5). Compare a média posterior usando o prior informativo (seção 2.2) contra o prior uniforme (Exercício 1). A diferença entre os dois posteriores é maior ou menor do que era com $n = 50$? Isso é consistente com a lógica de "o prior pesa mais quando há menos dados"?

Solução:

```
n_pequeno <- 5
k_pequeno <- 3

# Prior informativo
alpha_post_pequeno_inf <- alpha_prior + k_pequeno
beta_post_pequeno_inf <- beta_prior + (n_pequeno - k_pequeno)
alpha_post_pequeno_inf / (alpha_post_pequeno_inf + beta_post_pequeno_inf)

# Prior uniforme
alpha_post_pequeno_fraco <- 1 + k_pequeno
beta_post_pequeno_fraco <- 1 + (n_pequeno - k_pequeno)
alpha_post_pequeno_fraco / (alpha_post_pequeno_fraco + beta_post_pequeno_f
raco)
```

Com $n = 5$, a diferença entre os dois posteriores deve ser bem maior do que era com $n = 50$ (Exercício 1) — com poucos dados novos, o prior tem proporcionalmente mais peso na composição final do posterior. Essa é a demonstração direta e quantitativa de por que a escolha do prior importa mais em amostras pequenas, e importa cada vez menos conforme n cresce — o mesmo padrão qualitativo do TLC (Módulo 8): mais dados, menos dependência de suposições de partida.

---

8. Entrega esperada do módulo

Script `bayesiano_intro.R` que:

1. Implementa a atualização Beta-Binomial (seção 2) com pelo menos dois priors diferentes (informativo e uniforme), com gráfico comparando prior e posterior
2. Compara IC frequentista e intervalo de credibilidade Bayesiano para o mesmo cenário de proporção
3. Ajusta `modelo_freq` (lm) e `modelo_bayes` (brm) sobre os mesmos dados, com tabela comparando coeficientes e intervalos lado a lado

---

9. Critério de domínio

- ☐ Explica a relação prior → verossimilhança → posterior, e localiza o Teorema de Bayes do Módulo 8 dentro dela
- ☐ Sabe calcular analiticamente o posterior Beta-Binomial e explica por que a conjugação evita precisar de MCMC nesse caso específico
- ☐ Interpreta corretamente um intervalo de credibilidade (probabilidade direta sobre o parâmetro) e explica por que essa interpretação não é válida para o IC frequentista
- ☐ Entende por que a maioria dos modelos Bayesianos reais precisa de MCMC, e o que `brms` /Stan fazem por baixo, mesmo sem entender o algoritmo em detalhe
- ☐ Sabe ler o output de `summary(brm())` e compará-lo estruturalmente com `summary(lm())`
- ☐ Explica por que a influência do prior diminui conforme *n* cresce
- ☐

Todos os 4 exercícios rodam sem erro e a interpretação distingue claramente linguagem frequentista de linguagem Bayesiana

---

MÓDULO 15 · TÓPICOS AVANÇADOS

Delineamento Experimental

DOE

1. Os três princípios de Fisher

Todo delineamento experimental clássico se apoia em três princípios, formalizados por Ronald Fisher (o mesmo por trás do Teste Exato de Fisher, Módulo 9, seção 11.4, e da distribuição F por trás da ANOVA):

1. **Casualização (randomização):** a alocação de tratamentos às unidades experimentais deve ser aleatória, não por conveniência ou julgamento do pesquisador — elimina viés sistemático de alocação (ex.: dar sistematicamente o tratamento "novo" às melhores parcelas) e é o que justifica formalmente usar testes de hipótese e ANOVA sobre os dados resultantes.
2. **Repetição (replicação):** cada tratamento precisa aparecer em múltiplas unidades experimentais — sem repetição, não há como estimar a variabilidade dentro do tratamento (o "erro" da ANOVA, Módulo 9, seção 7.1), e portanto não há como distinguir efeito real de ruído.
3. **Controle local (blocagem):** quando se sabe de antemão que as unidades experimentais não são homogêneas (parcelas em posições diferentes do terreno, lotes de animais de idades diferentes), agrupar unidades semelhantes em blocos e aplicar todos os tratamentos dentro de cada bloco reduz a variância não explicada — o mesmo princípio de "estratificar reduz variância da estimativa" já demonstrado empiricamente no Módulo 8, Exercício 7 (AAS vs. estratificada), agora aplicado ao desenho de experimento em vez de à amostragem.

2. Delineamento Inteiramente Casualizado (DIC)

O desenho mais simples: tratamentos alocados **aleatoriamente**, sem nenhuma estrutura de bloco — apropriado quando as unidades experimentais são homogêneas (ou a heterogeneidade entre elas é desprezível frente ao efeito que se quer detectar).

```
library(dplyr)

set.seed(2026)
tratamentos <- c("Controle", "Suplemento_A", "Suplemento_B")
n_repeticoes <- 12

dic <- data.frame(
  parcela_id = seq_len(length(tratamentos) * n_repeticoes),
  tratamento = sample(rep(tratamentos, each = n_repeticoes))   #
casualização explícita
)

# Ganho de peso simulado (kg), com efeito real de tratamento
efeito_tratamento <- c(Controle = 0, Suplemento_A = 3.5, Suplemento_B = 5.
2)
dic$ganho_peso <- 20 + efeito_tratamento[dic$tratamento] + rnorm(nrow(dic)
, 0, 4)

table(dic$tratamento)   # confirma repetição balanceada -- 12 por
tratamento
```

`sample(rep(tratamentos, each = n_repeticoes))` implementa a casualização diretamente: monta a lista completa de rótulos de tratamento (repetidos o número certo de vezes) e embaralha a ordem — cada parcela recebe seu tratamento por sorteio, não por escolha.

Análise: ANOVA de um fator, exatamente como no Módulo 9:

```
modelo_dic <- aov(ganho_peso ~ tratamento, data = dic)
summary(modelo_dic)
TukeyHSD(modelo_dic)
```

Nenhuma novidade estatística aqui — é o Módulo 9, seção 7, aplicado a dado gerado sob desenho experimental explícito em vez de dado observacional. A novidade é conceitual: como os tratamentos foram casualizados, a diferença observada (se significativa) pode ser atribuída ao tratamento com confiança causal — algo que dado puramente observacional (como o `varejo_sintetico.csv` ou a PAM/IBGE dos módulos anteriores) não permite, exatamente pelo alerta de confundimento repetido desde o Módulo 4, seção 1.5.

---

3. Delineamento em Blocos Casualizados (DBC)

Quando as unidades experimentais **não** são homogêneas — parcelas em diferentes posições do terreno com fertilidade de solo variável, ou lotes de animais de propriedades diferentes —, blocar controla essa fonte de variação:

```
set.seed(2026)
blocos <- paste0("Bloco_", 1:4)
n_blocos <- length(blocos)

# Dentro de cada bloco, os 3 tratamentos aparecem exatamente uma vez, em
ordem casualizada
dbc <- expand.grid(bloco = blocos, tratamento = tratamentos) %>%
  group_by(bloco) %>%
  slice_sample(prop = 1) %>%   # casualiza a ordem dentro de cada bloco
  ungroup()

# Efeito de bloco real (ex.: bloco = talhão, com fertilidade de solo
diferente)
efeito_bloco <- c(Bloco_1 = -2, Bloco_2 = 0, Bloco_3 = 1.5, Bloco_4 = 3)
dbc$ganho_peso <- 20 + efeito_tratamento[dbc$tratamento] + efeito_bloco[db
c$bloco] +
  rnorm(nrow(dbc), 0, 4)
```

Análise: ANOVA de dois fatores, mesma estrutura do Módulo 9, seção 9 — mas com papel conceitual diferente para cada fator:

```
modelo_dbc <- aov(ganho_peso ~ tratamento + bloco, data = dbc)
summary(modelo_dbc)
```

Repare: `tratamento + bloco`, **sem** o termo de interação `tratamento:bloco` que a seção 9 do Módulo 9 exploraria em outro contexto — em DBC clássico, assume-se que blocos e tratamentos não interagem (o efeito do tratamento é o mesmo em todos os blocos, só o nível basal muda), e o bloco entra no modelo **só para remover sua variância do erro residual**, não porque haja interesse científico no efeito de bloco em si. Isso é diferente da ANOVA de dois fatores do Módulo 9, onde ambos os fatores (`regiao`, `cultura`) eram de interesse científico direto, incluindo a interação entre eles.

Por que blocar quando a fonte de variação é conhecida: comparando a soma de quadrados residual entre `modelo_dic` (sem controlar por bloco) e `modelo_dbc` (controlando), o DBC deve mostrar erro residual menor — a variação que antes ficava misturada no "ruído" agora está explicitamente atribuída ao bloco, deixando o teste do efeito de tratamento mais **poderoso** (Módulo 8, seção 10.3 — mesma lógica de poder estatístico, agora como consequência direta de uma decisão de desenho, não de aumentar *n*).

---

4. Delineamento em Quadrado Latino (DQL): controlando duas fontes de variação

Quando existem **duas** fontes de variação conhecidas simultaneamente (ex.: posição no terreno em uma direção **e** outra — gradiente de umidade E-O e gradiente de fertilidade N-S), o Quadrado Latino controla as duas ao mesmo tempo, com a restrição de que cada tratamento aparece exatamente uma vez em cada linha e em cada coluna:

```
set.seed(2026)
k <- 3   # número de tratamentos = número de linhas = número de colunas,
restrição do DQL

quadrado_base <- matrix(c("Controle", "Suplemento_A", "Suplemento_B",
                          "Suplemento_A", "Suplemento_B", "Controle",
                          "Suplemento_B", "Controle", "Suplemento_A"),
                        nrow = k, byrow = TRUE)

dql <- expand.grid(linha = 1:k, coluna = 1:k) %>%
  mutate(tratamento = quadrado_base[cbind(linha, coluna)])

efeito_linha <- c(-1, 0, 1.5)      # ex.: gradiente de umidade
efeito_coluna <- c(0.5, -0.5, 2)   # ex.: gradiente de fertilidade

dql$ganho_peso <- 20 + efeito_tratamento[dql$tratamento] +
  efeito_linha[dql$linha] + efeito_coluna[dql$coluna] + rnorm(nrow(dql),
0, 4)

modelo_dql <- aov(ganho_peso ~ tratamento + factor(linha) + factor(coluna)
, data = dql)
summary(modelo_dql)
```

Custo do DQL: exige k^2 unidades experimentais para k tratamentos (restrição rígida de linha × coluna) — funciona bem para poucos tratamentos, fica logisticamente inviável para muitos. É o desenho certo especificamente quando as duas fontes de variação são conhecidas, ortogonais entre si, e o número de tratamentos é pequeno — fora desse cenário, DBC (uma fonte de variação) ou delineamentos mais avançados (fora do escopo aqui) são mais apropriados.

---

5. Dimensionamento: quantas repetições por tratamento

Antes de ir a campo, a mesma pergunta do Módulo 8, seção 10.3, aplicada a um cenário de ANOVA (mais de dois grupos) em vez de comparação de duas médias:

```
install.packages("pwr")
library(pwr)

# Quantas repetições por tratamento para detectar um effect size f = 0.3
(médio, convenção de
# Cohen para ANOVA) com poder de 80%, alfa = 0.05, k = 3 tratamentos?
pwr.anova.test(k = 3, f = 0.3, sig.level = 0.05, power = 0.80)
```

`f` aqui é o tamanho de efeito específico para ANOVA (não o mesmo `d` do teste t, Módulo 8, seção 10.3) — relacionado a η^2 (Módulo 9, seção 12.2) pela transformação $f = \sqrt{\eta^2/(1-\eta^2)}$. Planejar repetições insuficientes é o erro mais caro em experimento de campo: ao contrário de dado observacional (onde *n* maior às vezes é "só" mais tempo de coleta), repetições insuficientes num ensaio agropecuário significam repetir a safra inteira ou o ciclo reprodutivo inteiro para corrigir.

---

6. Diferenças-chave vindas de Python

Conceito	Python	R
ANOVA (base de qualquer análise DOE)	`statsmodels.formula.api.ols(...).fit()` + `anova_lm`	`aov(y ~ tratamento)`
Geração de desenho experimental	Sem pacote de referência único — `pyDOE2` para desenhos fatoriais, menos maduro	`agricolae` — pacote dedicado a DOE agronômico (`design.crd`, `design.rcbd`, `design.lsd` para DIC/DBC/DQL)
Power analysis para ANOVA	`statsmodels.stats.power.FTestAnovaPower`	`pwr::pwr.anova.test()`

Conceito	Python	R
Blocagem (DBC)	Termo adicional no modelo `ols`, sem suporte dedicado a "desenho"	`aov(y ~ tratamento + bloco)`, mesma estrutura, com `agricolae` para gerar o desenho em si

O ecossistema R tem vantagem histórica clara aqui: `agricolae` é um pacote dedicado inteiramente a delineamento experimental agronômico, com funções que geram o desenho (`design.crd()` , `design.rcbd()` , `design.lsd()`) já casualizado corretamente — reduzindo o risco de erro na casualização manual (como feito neste módulo, para fins didáticos). Python não tem equivalente direto tão maduro, refletindo a origem histórica do R em bioestatística e experimentação agrícola (o próprio Fisher, seção 1, trabalhou em estatística agrícola na Rothamsted Experimental Station).

---

7. Exercícios

Exercício 1 — DIC: ANOVA e Tukey completos

Rode a ANOVA do `modelo_dic` (seção 2) e, se significativa, o Tukey HSD. `Suplemento_B` difere significativamente do `Controle` ? O tamanho do efeito observado é consistente com `efeito_tratamento` usado para gerar os dados?

Solução:

```
summary(modelo_dic)
TukeyHSD(modelo_dic)
```

Dado que `efeito_tratamento["Suplemento_B"] = 5.2` (o maior efeito construído) contra um desvio padrão de ruído de 4, espera-se que o par Suplemento_B–Controle apareça significativo no Tukey, com diferença de médias estimada próxima de 5.2 kg — a ANOVA recupera, dentro da variação amostral esperada, a estrutura que gerou os dados.

Exercício 2 — DBC: ganho de eficiência ao blocar

Compare a soma de quadrados do erro residual entre `modelo_dic` (seção 2, sem bloco) e `modelo_dbc` (seção 3, com bloco) — usando dados gerados com a mesma estrutura de tratamento. O DBC reduz o erro residual, como esperado pela remoção da variação de bloco?

Solução:

```
summary(modelo_dic)[[1]]["Residuals", "Sum Sq"]
summary(modelo_dbc)[[1]]["Residuals", "Sum Sq"]
```

Como `efeito_bloco` foi construído com magnitude real (variando de -2 a +3), e o `modelo_dbc` explicitamente separa essa fonte de variação do erro residual (enquanto `modelo_dic`, sem bloco, deixa toda essa variação misturada no resíduo), espera-se soma de quadrados residual menor no `modelo_dbc` — a demonstração direta de por que blocar aumenta o poder de detectar o efeito de tratamento quando a fonte de variação de bloco é real.

Exercício 3 — DQL: verificando a restrição de desenho

Confirme, para `dql` (seção 4), que cada tratamento aparece exatamente uma vez em cada linha e exatamente uma vez em cada coluna — a restrição estrutural que define um Quadrado Latino. Se essa restrição fosse violada (ex.: um tratamento aparecendo duas vezes na mesma linha), por que isso comprometeria a análise?

Solução:

```
table(dql$linha, dql$tratamento)    # cada célula deve ser exatamente 1
table(dql$coluna, dql$tratamento)   # idem
```

Ambas as tabelas devem mostrar exatamente um “1” por célula, confirmando a restrição. Se um tratamento aparecesse duas vezes na mesma linha (e outro nenhuma), o efeito de linha deixaria de ser balanceado em relação a tratamento — a soma de quadrados de linha e de tratamento deixariam de ser estatisticamente independentes (ortogonais) uma da outra, contaminando a estimativa de ambos os efeitos, exatamente o tipo de confundimento estrutural que a restrição do DQL existe para evitar por construção.

Exercício 4 (desafio) — Dimensionando um DBC a partir do zero

Usando `pwr.anova.test()` (seção 5), calcule quantas repetições por tratamento seriam necessárias para detectar um effect size pequeno (f = 0.15) com poder de 90%, para k = 4 tratamentos. Compare com o n necessário para f = 0.4 (efeito grande) no mesmo poder. Qual a implicação prática de planejar um experimento de campo para detectar efeitos pequenos?

Solução:

```
pwr.anova.test(k = 4, f = 0.15, sig.level = 0.05, power = 0.90)
pwr.anova.test(k = 4, f = 0.40, sig.level = 0.05, power = 0.90)
```

O n por grupo necessário para f = 0.15 deve ser substancialmente maior que para f = 0.4 — mesmo padrão já visto no Módulo 8, Exercício 8(b): effect sizes pequenos exigem amostras muito maiores para o mesmo poder. Na prática de campo, isso significa que um experimento dimensionado ingenuamente (sem power analysis prévia) para detectar diferenças sutis

entre tratamentos pode sair caro ou logisticamente inviável — informação que deveria vir **antes** de plantar/inseminar/alocar recursos, não depois de já ter coletado dados insuficientes.

---

8. Entrega esperada do módulo

Script `doe_ensaio.R` que:

1. Gera um DIC casualizado (seção 2) com pelo menos 3 tratamentos e ANOVA + Tukey completos
2. Gera um DBC (seção 3) com estrutura de bloco real, comparando erro residual com o DIC equivalente
3. Gera um DQL (seção 4) e confirma a restrição de linha/coluna
4. Calcula, via `pwr.anova.test()`, o *n* por tratamento necessário para pelo menos dois cenários de effect size diferentes

---

9. Critério de domínio

- ☐ Explica os três princípios de Fisher (casualização, repetição, controle local) e por que cada um é necessário, não decorativo
- ☐ Sabe quando usar DIC vs. DBC vs. DQL, com base no número de fontes de variação conhecidas a controlar
- ☐ Entende por que o DBC não inclui interação tratamento×bloco por padrão, diferente da ANOVA de dois fatores do Módulo 9
- ☐ Demonstra, com número concreto, por que blocar reduz erro residual quando a fonte de variação de bloco é real
- ☐ Sabe dimensionar repetições necessárias via `pwr.anova.test()` antes de planejar um experimento

- ☐ Todos os 4 exercícios rodam sem erro e a interpretação conecta o resultado ao princípio de desenho correspondente

---

Estatística Aplicada com R

Dezesseis capítulos, do zero de sintaxe à inferência bayesiana e ao delineamento experimental — teoria matemática, premissas verificáveis e código R pronto para rodar, com dados reais e sintéticos documentados com transparência.

Sérgio Berlotto Jr.

Formado em Big Data e Inteligência Analítica
Pós-graduado em Arquitetura de Software

github.com/berlotto/estatistica-aplicada-r-materiais