# Introdução à Manipulação e Tratamento de Dados com R

Idris da Silva Santos

Departamento de Estatística e Ciências Atuariais – CCET – UFS

26 de Novembro de 2025

# Roteiro da Aula

# Introdução e Objetivos

# Cronograma do Workshop

Este curso foi desenhado para cobrir o fluxo completo de trabalho científico e analítico:

## Dia 1: Hoje (Manipulação de Dados)

**Foco:** Tratamento de dados e preparação.
Aprenderemos a transformar dados brutos (”sujos”) em uma base sólida para análise usando **R**.

## Dia 2: Amanhã (Visualização Interativa)

**Foco:** Business Intelligence (BI).
Criação de dashboards dinâmicos com dados usando **PowerBI**.

## Dia 3: Encerramento (Escrita Técnica)

**Foco:** Comunicação e Publicação.
Como estruturar e formatar relatórios técnicos profissionais e acadêmicos utilizando LaTeX.

# Contextualização

- **O Problema Real:**
  - Dados do mundo real raramente vêm prontos para análise.
  - Eles são ”sujos”, desorganizados e inconsistentes.

- **O Fluxo de Ciência de Dados:**

  Importar → **Limpar/Arrumar** → Explorar → Modelar → Comunicar

- **Ferramenta:** Linguagem R + Ecossistema *Tidyverse*.

# Objetivos de Aprendizagem

Ao final desta oficina, você será capaz de:

1. **Importar dados** diagnosticando erros comuns (separadores, encoding).
2. **Limpar dados** usando o dialeto *Tidyverse* (`dplyr`, `tidyr`, `stringr`).
3. **Tratar inconsistências** como tipos errados, duplicatas e valores ausentes (`NA`).
4. **Visualizar** distribuições básicas com `ggplot2`.
5. **Treinar um modelo** simples de Árvore de Decisão.

# O Dialeto Tidyverse

# O Ecossistema Tidyverse

**O que é?**

- Uma coleção de pacotes R desenhados para ciência de dados.
- Compartilham uma filosofia comum de design, gramática e estrutura de dados.
- **Filosofia:** O código deve ser legível por humanos, não apenas por computadores.

**O Operador Pipe (%>%)**

- Lê-se como ”então”ou ”e depois”.
- Conecta as ações em uma sequência lógica.
- Evita o aninhamento confuso de funções: $f(g(h(x)))$ vira $x \to h \to g \to f$.

# A Gramática de Manipulação: Os Verbos

O pacote `dplyr` propõe que a manipulação de dados seja feita através de **verbos** intuitivos. Imagine que você está escrevendo uma frase:

- **`filter()`**: *Filtra* linhas baseadas em condições (seleciona casos).
- **`select()`**: *Seleciona* colunas pelo nome (seleciona variáveis).
- **`mutate()`**: *Modifica* ou cria novas colunas (transformação).
- **`arrange()`**: *Arranja* (ordena) as linhas.
- **`summarise()`**: *Sumariza* os dados em uma estatística (média, total).

**Exemplo de "Frase" em R:**

```r
dados %>%
  filter(diagnostico == "M") %>%  # Pegue os malignos, E DEPOIS...
  select(area_mean, texture_mean) %>% # Selecione essas colunas, E DEPOIS...
  mutate(razao = area_mean / texture_mean) # Crie esta nova variavel.
```

# Ambiente e Pacotes

# Preparando o Ambiente

Vamos instalar e carregar as bibliotecas essenciais.

- `tidyverse`: Manipulação e visualização.
- `janitor`: Limpeza de nomes de colunas.
- `vroom`: Leitura rápida de dados.
- `tidymodels`: Framework de modelagem.

```
pacotes <- c("janitor", "tidyverse", "tidymodels", "vroom", "corrr", "GGally", "rpart
    .plot")
instalados <- installed.packages()
for(pacote in pacotes) {
  if (!(pacote %in% instalados)) install.packages(pacote)
}
library(tidyverse)
library(tidymodels)
library(janitor)
library(vroom)
library(corrr)
library(GGally)
library(rpart.plot)
```

# Importação e Diagnóstico

# Importação Inicial (O Caos)

Tentativa ingênua de ler o arquivo CSV. O que pode dar errado?

```
# Leitura inicial
dados <- vroom::vroom("./wbcd_dirty.csv")

# Visualizar a estrutura bruta
dados %>% pillar::glimpse()
```

**Problemas observados:**

- Colunas numéricas lidas como texto (`chr`).
- Mistura de separadores decimais (. e ,).
- Aviso de *parsing error*.

# Diagnosticando Erros de Estrutura

O R nos avisou sobre problemas. Vamos investigar onde eles estão.

```
# Investigando falhas de leitura
dados %>% vroom::problems()
```

**O Diagnóstico:**

- O erro aponta para a linha 3.
- Esperava-se 32 colunas, mas o parser encontrou mais (ou caracteres inesperados que quebraram a estrutura).

# Intervenção Cirúrgica (Strings)

Às vezes, o erro está na formação do arquivo (aspas não fechadas, por exemplo). Vamos tratar o arquivo como texto puro para corrigir.

```
# Lendo como texto puro
linhas <- readr::read_lines("./wbcd_dirty.csv")

# Identificando tokens problematicos (Regex Avancado)
tokens <- stringr::str_split(linhas[3], ',')
tokens %>%
  purrr::map(grep, pattern = '(?:(?!").(,|$))+', perl = TRUE) %>%
  purrr::map(~tokens[[1]][.])

# Corrigindo aspas desbalanceadas na linha 3
linhas[3] <- linhas[3] %>%
  stringr::str_replace('("\\d+\\.\\d+),(\\d+\\.\\d+)', '\\1",\\2')

# Salvando a versao corrigida
linhas %>% readr::write_lines("./wbcd_01.csv")
```

# Limpeza de Dados (Data Wrangling)

# Reimportação e Nomes de Colunas

Agora importamos o arquivo corrigido, especificando que o decimal é ponto (padrão US). Em seguida, padronizamos os nomes das colunas.

```r
# Importando com locale correto
dados <- vroom::vroom(
  "./wbcd_01.csv",
  locale = locale(decimal_mark = ".") # Importante!
)

dados %>% pillar::glimpse()

# Verificando nomes originais (ruins)
dados %>% names()

# Limpando nomes automaticamente (snake_case)
dados <- janitor::clean_names(dados)
dados %>% names()
```

# Tratando Strings "N/A"

Temos valores "N/A"(texto) que deveriam ser NA (valor nulo do R). Isso impede cálculos numéricos.

```r
# Substituindo "N/A" por NA real em todas as colunas de texto
dados <- dados %>%
  dplyr::mutate(across(where(is_character), ~na_if(., "N/A")))

# Verificando a mudanca
dados %>% pillar::glimpse()
```

# Convertendo Texto para Números

Muitas colunas ainda são texto (`chr`) por causa de sujeiras (vírgulas no lugar de pontos, caracteres estranhos). Vamos forçar a conversão (parse).

```
# Aplicando parse_number em colunas de texto (exceto ID e Diagnosis)
dados <- dados %>%
  dplyr::mutate(
    across(
      where(is.character) & !c(id, diagnosis),
      ~parse_number(
        str_replace_all(.x, ",", ".") # troca "," por "."
      )
    )
  )

dados %>% pillar::glimpse()
```

# Removendo Duplicatas

Linhas repetidas enviesam a análise estatística.

```
# Verificando duplicatas
dados %>% janitor::get_dupes()

# Removendo duplicatas (mantendo apenas linhas distintas)
dados <- dados %>% dplyr::distinct()

# Confirmando limpeza
dados %>% janitor::get_dupes()
```

# Padronizando Categorias (Diagnosis)

A coluna alvo possui inconsistências: ”M”, ”m”, ”B”, ”b”, ”Benign”. Precisamos uniformizar.

```r
# Verificando valores unicos
dados$diagnosis %>% str_unique()

# Padronizando com case_when
dados <- dados %>%
  dplyr::mutate(
    diagnosis = case_when(
      diagnosis %in% c("M", "m") ~ "M",
      diagnosis %in% c("B", "b", "Benign") ~ "B"
    ),
    diagnosis = forcats::as_factor(diagnosis) # Convertendo para Fator
  )

dados$diagnosis %>% unique()
```

# Imputação de Dados Faltantes

Remover linhas com NA pode reduzir demais o dataset. Vamos imputar pela mediana de cada grupo (Benigno/Maligno).

```r
# Imputacao por mediana por grupo
dados <- dados %>%
  dplyr::group_by(diagnosis) %>%
  dplyr::mutate(across(
    where(is.numeric),
    ~replace_na(.x, median(.x, na.rm = TRUE)))
  ) %>%
  ungroup()

dados %>% pillar::glimpse()
```

# Análise Exploratória

# Sumarizando os dados

Agora que tratamos os problemas, podemos usar a função `summary` (sumário) para exibir algumas medidas descritivas. Aqui selecionamos as colunas que terminam com `_mean`.

```
dados %>%
  dplyr::select(ends_with("_mean")) %>%
  summary()

```

# O Desafio da Visualização em Massa

Façamos o histograma do raio médio, `radius_mean`.

```
dados %>%
  ggplot(aes(x = smoothness_mean)) +
  geom_histogram(bins = nclass.Sturges(dados$radius_mean)) +
  labs(
    x = "Raio médio",
    y = "Frequência"
  ) +
  theme_classic()

```

# O Desafio da Visualização em Massa

**Cenário:** Temos 30 colunas numéricas de medidas (`radius_mean`, `texture_mean`, etc).

**O Problema:** O `ggplot2` espera *uma* coluna específica para mapear no eixo X. Se quisermos ver a distribuição de todas as variáveis, teríamos que copiar e colar o código do gráfico 30 vezes:

```
# Inviavel fazer isso 30 vezes:
ggplot(dados, aes(x = radius_mean)) + geom_histogram()
ggplot(dados, aes(x = texture_mean)) + geom_histogram()
# ...

```

**A Solução:** Transformar a estrutura dos dados de **Wide** (Largo) para **Long** (Longo), transformando os *nomes das colunas* em uma variável categórica.

# Transformação: Wide → Long

Usamos a função `pivot_longer` do pacote `tidyr`. Ela "empilha"as colunas selecionadas.

- **cols:** Quais colunas vamos empilhar?
- **names_to:** Nome da nova coluna que guardará os "títulos".
- **values_to:** Nome da nova coluna que guardará os números.

```r
# Criando um dataset longo
dados_longos <- dados %>%
  tidyr::pivot_longer(
    cols = matches("_(mean|se|worst)"), # Regra para selecionar colunas
    names_to = "variavel",              # Coluna com o nome das variaveis
    values_to = "valor"                 # Coluna com as medidas
  )

# Resultado: O dataset fica mais estreito, mas muito mais comprido!
```

# Visualização em Massa: Primeira Tentativa

Vamos usar o `facet_wrap` para criar um gráfico por variável. Por padrão, o ggplot obriga todos os gráficos a terem os **mesmos eixos**.

```
dados_longos %>%
  dplyr::filter(variavel %>% grepl("_mean", x = .)) %>%
  ggplot() +
  geom_histogram(
    aes(x = valor),
    bins = nclass.Sturges(dados$radius_mean), # regra de Sturges
    color = "black", fill = "white"
  ) +
  facet_wrap(~variavel) + # O padrao e scales = "fixed"
  theme_classic()
```

**O Problema:**

- `texture_mean` vai até 2500.
- `concavity_mean` vai até 0.08.
- Resultado: As variáveis pequenas ficam "esmagadas"em uma única barra à esquerda.

# Ajustando as Escalas (Scales Free)

Para corrigir, usamos o argumento `scales = "free"`. Isso libera cada "faceta"para ter seus próprios limites de eixo X e Y.

```
dados_longos %>%
  dplyr::filter(grepl("_mean", variavel)) %>%
  ggplot() +
  geom_histogram(
    aes(x = valor),
    bins = nclass.Sturges(dados$radius_mean),
    color="black", fill="white"
  ) +
  # A correcao magica:
  facet_wrap(~variavel, scales = "free") +
  theme_classic()
```

**Resultado:** Agora conseguimos ver a forma da distribuição de cada variável, independentemente da sua magnitude.

# Análise de Distribuição e Outliers

Antes de separar por diagnósticos, vamos olhar a distribuição global de cada medida usando **Boxplots**.
**Objetivo:** Identificar a variabilidade total e a presença de valores extremos (*outliers*).

```
dados_longos %>%
  dplyr::filter(grepl("_mean", variavel)) %>%
  ggplot() +
  geom_boxplot(aes(y = valor)) +
  facet_wrap(~variavel, scales = "free") +
  theme_classic()
```

*Nota: Vemos muitos outliers (pontos pretos), mas não sabemos se são casos graves ou apenas ruído.*

# Identificando Padrões (Separação por Grupo)

**Motivação:** Para construir um classificador, precisamos de variáveis que se comportem de maneira **diferente** entre Benignos e Malignos.
Vamos separar os boxplots por cor (`diagnosis`). Se as caixas se separarem verticalmente, a variável pode ser uma boa preditora.

```
dados_longos %>%
  dplyr::filter(grepl("_mean", variavel)) %>%
  ggplot() +
  # Adicionamos cor baseados no diagnostico
  geom_boxplot(aes(y = valor, color = diagnosis)) +
  facet_wrap(~variavel, scales = "free") +
  labs(color = "Diagnóstico") +
  theme_classic()
```

**Conclusão Visual:** Variáveis como `radius_mean` e `fractal_dim_mean` separam bem os grupos.

# Visualização Multivariada (Pairs Plot)

Às vezes, queremos ver a relação entre várias variáveis ao mesmo tempo (dispersão) e suas distribuições. O pacote `GGally` automatiza isso.
*Nota: Selecionamos apenas algumas colunas, pois fazer isso com todas as 30 travaria o plot.*

```
dados %>%
  # Selecionando variaveis de interesse
  dplyr::select(diagnosis, radius_mean, texture_mean, fractal_dim_mean) %>%

  # ggpairs cria uma matriz de graficos
  GGally::ggpairs(
    aes(color = diagnosis, alpha = 0.5) # Cores por grupo
  ) +
  theme_classic()
```

# Correlação Não-Paramétrica (Spearman)

O gráfico anterior sugere relações não-lineares. Vamos quantificar a associação entre o **Raio** e a **Textura** usando correlação de posto (Spearman), que é mais robusta a outliers que a de Pearson.
Usaremos o pacote `corrr`, que devolve um *tibble* (tabela) em vez de uma matriz, facilitando a leitura.

```
# Calculando a correlacao de Spearman
dados %>%
  dplyr::select(radius_mean, texture_mean) %>%
  corrr::correlate(method = "spearman", quiet = TRUE)
```

**Por que Spearman?** Porque nossas variáveis podem não ser linearmente associadas.

# Modelagem: Árvore de Decisão

# Divisão Treino e Teste

Preparando os dados para o aprendizado de máquina.

```r
set.seed(20251125) # Reprodutibilidade

# Divisao 80/20
split <- rsample::initial_split(dados, prop = 0.8)
treino <- rsample::training(split)
teste <- rsample::testing(split)
```

# Treinando o Modelo

Usando a engine `rpart` (Recursive Partitioning).

```r
# Especificacao do modelo
tree_spec <- parsnip::decision_tree(
  mode = "classification",
  engine = "rpart",
  tree_depth = 4 # Profundidade maxima
)

# Ajuste do modelo
tree_fit <- tree_spec %>%
  fit(diagnosis ~ ., data = treino)
```

# Visualizando a Árvore

Interpretando as regras criadas pelo modelo.

```
# Plot da arvore
rpart.plot::rpart.plot(tree_fit$fit, type = 4, roundint = FALSE)
```

# Avaliação de Desempenho

# Entendendo a Matriz de Confusão

A base de toda avaliação em classificação é comparar o **Predito** vs **Real**.

| | | **Realidade** | |
|---|---|---|---|
| | | Maligno (M) | Benigno (B) |
| **Predição** | Maligno (M) | **Verdadeiro Positivo** (O modelo acertou o câncer) | Falso Positivo (Alarme falso) |
| | Benigno (B) | Falso Negativo (Erro grave: câncer não detectado) | **Verdadeiro Negativo** (O modelo acertou que é saudável) |

- **Verdadeiro Positivo/Negativo (Verde):** O modelo acertou.
- **Falso Negativo (Erro Tipo II):** O erro mais perigoso na medicina (deixar um paciente doente ir para casa).
- **Falso Positivo (Erro Tipo I):** Gera estresse, mas pode ser verificado com exames adicionais.

# Métricas Resumo: Acurácia e Kappa

Olhar a matriz inteira pode ser difícil. Usamos métricas para resumir a qualidade:

**1. Acurácia (Accuracy)**

$$\frac{\text{Total de Acertos}}{\text{Total de Casos}}$$

- *Pró:* Intuitiva.
- *Contra:* Pode enganar em dados desbalanceados (ex: se 99% dos pacientes são saudáveis, chutar "saudável"sempre dá 99% de acurácia).

**2. Kappa de Cohen**

$$\frac{\text{Acurácia Observada} - \text{Esperada ao Acaso}}{1 - \text{Esperada ao Acaso}}$$

- Mede a concordância **além do acaso**.
- 1 = Perfeito; 0 = Equivalente a jogar uma moeda.
- Útil para dados médicos e crucial para psicométricos.

# Calculando no R (Pacote Yardstick)

O pacote `yardstick` (parte do tidymodels) padroniza esses cálculos.

```r
# Gerar Predicoes no conjunto de Teste
predicoes <- tree_fit %>%
  predict(teste) %>%
  dplyr::pull(.pred_class)

# Gerar Matriz de Confusao
teste %>%
  dplyr::mutate(predicoes = predicoes) %>%
  conf_mat(truth = diagnosis, estimate = predicoes)

# Calcular Metricas (Acuracia e Kappa)
metricas <- metric_set(accuracy, kap) # acuracia e \kappa
teste %>%
  dplyr::mutate(predicoes = predicoes) %>%
  metricas(truth = diagnosis, estimate = predicoes)
```

# Conclusão

# Recapitulando

1. **Importamos** dados brutos e identificamos erros de estrutura.
2. **Corrigimos** problemas de codificação e estrutura do CSV.
3. **Limpamos** nomes, tipos de dados, strings inconsistentes e valores nulos.
4. **Exploramos** os dados visualmente para entender as relações.
5. **Modelamos** o problema usando uma Árvore de Decisão simples e interpretável.

**Próximos Passos:** Boosting, testar modelos mais robustos (Florestas Aleatórias) e refinar o tratamento de dados.

# Dúvidas?

Link para o material: https://github.com/nueidris/semac2025