

## Script 4 — 4\_Data\_Preparation.R

### Objetivo do capítulo (CRISP-DM: Data Preparation)

- limpeza base (target observado, duplicados, coerência 1–5)
  - binarização padronizada de variáveis lógicas/texto (0/1)
  - split estratificado em **train/val/test**
  - “fit” de parâmetros **apenas no treino** (imputação, winsorização, seleção de features raras)
  - aplicação desses parâmetros a val/test (**evita data leakage**)
  - criação de features derivadas simples
  - escalamento (fit no treino, apply nos outros)
  - exportação de CSVs finais + logs de qualidade de classificação (QC) de dados.
- 

## 1) **prep\_basic(df)**: limpeza mínima + binarização inicial

---

### Finalidade

Executa as transformações que **não dependem** de aprender parâmetros estatísticos do conjunto de dados (ou que são seguras antes do split), e guarda métricas de qualidade para análises/auditorias posteriores.

#### 1.1 Controlo de qualidade “antes”

Cria uma lista com:

- **n\_raw**: nº linhas do df original
- **n\_dup\_raw**: nº de duplicados exatos
- **n\_missing\_target\_raw**: nº de **NA** no target

#### 1.2 Remoção de **NA** no target

```
df <- filter(!is.na(score_review))
```

#### 1.3 Remoção de duplicados exatos

```
df <- df[ !duplicated(df), ]
```

Remove linhas 100% iguais para evitar “repetição” artificial de observações.

#### 1.4 Conversão do target para numérico + saneamento

- converte **score\_review** para numeric (com **suppressWarnings**)
- se existirem valores que viram NA após conversão (ex.: strings), remove esses casos

## 1.5 Garantia da escala do target (1–5)

Conta quantos valores estão fora de [1,5] e, se existirem, faz **clamp**:

```
score_review <- pmin(5, pmax(1, score_review))
```

Regista `qc$n_target_outside_1_5`.

## 1.6 Binarização inicial das features

Para todas as colunas exceto:

- `score_review` (target)
- `logavaliacoes`

aplica:

```
df[[col]] <- to_binary01(df[[col]])
```

### O que isto faz na prática (do Utils):

- logical TRUE/FALSE → 1/0
- numéricas já {0,1} mantêm

## 1.7 QC como atributo

```
attr(df, "cap4_qc") <- qc
```

Isto é um detalhe muito bom: permite ao `run_cap4` escrever logs QC sem recalcular.

## 2) `preprocess_fit(train, target, rare_thr)`: aprender parâmetros no treino (anti-leakage)

Finalidade

Aprender tudo o que envolve “estatísticas do dataset” **só no treino**, para depois aplicar de forma idêntica em val/test.

## 2.1 Identificação de tipos de colunas

- `bin_cols`: colunas binárias 0/1 (exceto target)
- `num_cols`: numéricas
- `char_cols`: character

Isto define que estratégia de imputação aplicar.

## 2.2 Imputação por moda (binárias e texto)

Cria `mode_map` para:

- todas as binárias
- todas as colunas character

Cada entrada guarda:

```
mode_value(train[[col]])
```

**Justificação:** moda é apropriada para binárias/categóricas (preserva a classe dominante).

## 2.3 Imputação por mediana (numéricas não-binárias)

Define `num_nonbin = numéricas - {target, binárias}` e calcula mediana por coluna.

**Justificação:** mediana é robusta a outliers.

## 2.4 Winsorização (fit no treino)

Se `logavaliacoes` existir e for numérica:

```
winsor_limits <- winsorize_iqr_fit(train$logavaliacoes, k=1.5)
```

Ou seja, os limites `lo/hi` são aprendidos **apenas no treino**.

## 2.5 Remoção de binárias raras (NZV simples)

Para cada binária:

- calcula `prop = mean(x==1)`
- calcula `min_prop = min(prop, 1-prop)`
- se `min_prop < rare_thr` (default 0.02) → entra em `drop_cols`

**Motivação:** colunas quase constantes têm pouca informação e podem:

- aumentar ruído
- atrapalhar regularização
- induzir splits artificiais em árvores

## Output do `preprocess_fit`

Uma lista `params` contendo:

- colunas por tipo
- mapas de imputação

- limites de winsor
  - `drop_cols`
  - `rare_thr`
- 

## 3) `add_features(df, params)`: feature engineering simples e interpretável

---

### 3.1 `n_bin_true`

Cria uma feature com a soma de colunas binárias disponíveis:

```
n_bin_true <- rowSums(bin_cols == 1)
```

Interpretação: "quantas flags positivas aquele hotel tem?".

### 3.2 Interação `CarimboTripAdvisor × logavaliacoes`

Se ambas existirem e forem numéricas:

```
CarimboTripAdvisor_x_logavaliacoes <- CarimboTripAdvisor * logavaliacoes
```

**Interpretação possível:** o efeito do "CarimboTripAdvisor" pode "crescer" em hotéis mais populares (mais avaliações), ou vice-versa.

---

## 4) `preprocess_apply(df, params)`: aplicar o que foi fit no treino

---

Este é o passo anti-leakage mais crítico.

### 4.1 Drop de colunas raras

Remove as `params$drop_cols` em qualquer conjunto (train/val/test) de forma consistente.

### 4.2 Imputação por moda (binárias + character)

Para cada coluna em `mode_map`:

- se existe no df, substitui `NA` pela moda
- se a coluna for numérica, força moda para numeric quando possível

### 4.3 Imputação por mediana (numéricas não-binárias)

Substitui **NA** pela mediana aprendida no treino.

#### 4.4 Winsorização de `logavaliacoes`

Aplica limites de fit do treino:

```
winsorize_apply(df$logavaliacoes, params$winsor_limits)
```

---

## 5) Escalamento (standardization) fit no treino, apply nos restantes

---

### `scale_fit(train, target, bin_cols)`

- seleciona preditores numéricos
- exclui `target` e `bin_cols`
- calcula `mu` e `sd`
- substitui `sd=0` por 1 (evita divisão por zero)

**Porquê excluir binárias:** binárias já estão em escala comparável (0/1). Escalá-las pode atrapalhar interpretação e alguns modelos.

### `scale_apply(df, scaler)`

Aplica:

```
(x - mu)/sd
```

somente nas colunas listadas em `scaler$predictors_num`.

---

## 6) Orquestração completa: `run_cap4(...)`

---

### Inputs

- `df_raw`: dataset bruto
- `out_dir`: pasta de outputs
- `seed, train_frac, val_frac`: controlam split estratificado
- `rare_thr`: limiar de raridade para remover colunas binárias

### 6.1 Limpeza base + QC

- chama `prep_basic(df_raw)`
- extrai `qc` do atributo

- grava:
  - missing após limpeza base: `cap4_missing_after_basic.csv`
  - QC em CSV: `cap4_qc_basic.csv`
  - QC em TXT: `cap4_qc_basic.txt`

## 6.2 Split estratificado 3-way

```
split3 <- stratified_split_3way(...)
```

Divide em train/val/test preservando a distribuição do `score_review`.

## 6.3 Fit/apply de preprocessing (anti-leakage)

- `params <- preprocess_fit(train, ...)`
- `train2 <- preprocess_apply(train, params)`
- `val2/test2 <- preprocess_apply(val/test, params)`

Ou seja: **imputações, winsor e seleção de features** são aprendidos no treino.

## 6.4 Identificação das binárias finais (após o processamento)

Recalcula binárias em `train2` para saber quais se devem excluir.

## 6.5 Fit/apply de escalamento (anti-leakage)

- `scaler <- scale_fit(train2, ...)`
- aplica em train/val/test

## 6.6 Exportação de datasets finais

### Sem escalamento

- `train.csv`
- `val.csv`
- `test.csv`

### Com escalamento

- `train_scaled.csv`
- `val_scaled.csv`
- `test_scaled.csv`
- `cap4_drop_cols_raras.csv` (quais as colunas binárias que foram removidas)

## 6.7 Resumo final

`cap4_resumo_final.txt` com:

- seed e frações
- rare\_thr

- lista de colunas removidas
- tamanhos de train/val/test
- diretório de outputs