Script 2 — `2_Business_Understanding.R`

Objetivo do capítulo (CRISP-DM: Business Understanding)

- **Formulação do problema** (o que estás a prever e porquê)
- **Inventário de variáveis** (tipos, papel, se é binária)
- **Hipóteses iniciais** (sinais esperados / interpretação)
- **Critérios de sucesso** (métricas)
- **Notas de qualidade dos dados** (missing, duplicados, coerência do target)
- **Plano CRISP-DM** (mapa para o relatório)

Tudo isto é guardado em ficheiros (txt/csv) para que seja possivel analisar e usar posteriormente no relatório.

---

1) Função principal: `run_cap1(df, out_dir, target, tol_within)`

Inputs

- `df`: dataset já carregado (no teu pipeline, tipicamente `df_raw`)
- `out_dir`: pasta onde o capítulo vai escrever outputs
- `target`: por default `"score_review"` (escala 1 a 5)
- `tol_within`: tolerância para métrica complementar (por default 0.5)

Output (invisível)

Devolve uma lista (para debug/inspeção) com:

- `dict` (dicionário de variáveis)
- `target_dist` (distribuição do target)
- `missing` (tabela de missing)
- `duplicated_rows`
- `out_dir`

Isto é útil para testes, mas o essencial do capítulo são os ficheiros gerados.

---

2) Passos e metodologia (o que acontece dentro do `run_cap1`)

2.1) Preparação da pasta de outputs

```
ensure_dir(out_dir)
```

Garante que a pasta existe antes de escrever qualquer coisa. Isto evita erros por “no such directory”.

2.2) Identificação do problema (escopo do dataset)

O script calcula:

- `n_obs` = número de linhas
- `n_vars` = número de colunas
- `predictors` = todas as colunas exceto o `target`

Isto documenta logo o “universo” do teu problema: **quantas observações tens** e **quantos atributos tens para prever o score**.

2.3) Tipos de variáveis (classificação simples)

Cria `var_type` com uma tipologia simplificada:

- `"numerica"` se `is.numeric`
- `"logica"` se `is.logical`
- `"categorica_texto"` se `is.character`
- caso contrário, usa `class(x)[1]`

Isto alimenta depois o **dicionário de variáveis** (`dicionario_variaveis.csv`), que é ótimo para o relatório.

2.4) Heurística para detetar variáveis binárias

Cria `is_bin` para as **preditoras** com regras simples:

- **lógica** (`TRUE/FALSE`) → binária
- **numérica** com valores apenas em {0,1} → binária
- **texto** com valores apenas em {"true","false"} (case-insensitive) → binária

> Importante: é uma “deteção por padrão” para documentação e orientação. A conversão/binzarização real é feita mais à frente (Cap. 4).

---

3) Hipóteses iniciais (sinais esperados)

O script cria um data.frame `hypotheses` com uma linha por preditora.

Depois define uma função helper:

```
set_hyp <- function(v, txt) { ... }
```

que preenche a hipótese apenas se a variável existir no dataset (isto é robusto caso o dataset mude um pouco).

Hipóteses específicas que o script já assume

Exemplos:

- `CarimboTripAdvisor`: esperado **positivo** (credibilidade/selo)
- `Tomar_medidas_seguranca`: esperado **positivo** (confiança)
- `Patrocinado`: **incerto** (marketing pode não significar qualidade)
- `logavaliacoes`: **incerto** (pode estabilizar rating ou refletir heterogeneidade)

Grupo "amenities"

Para variáveis como `Breakfast`, `WiFi_gratuito`, etc., a hipótese default é **associação positiva** (mais serviços → melhor experiência).

Fallback

Se alguma variável não tiver hipótese explícita:

> "Hipotese a discutir no relatorio (sinal esperado nao definido a priori)."

Isto é bom metodologicamente: evita inventar sinais sem base e deixa claro que será confirmado na modelação.

---

4) Critérios de sucesso (métricas)

O script escreve um bloco "Success Criteria" com:

- **RMSE, MAE, R^2** (no teste)
- Métrica complementar: **% dentro de ±tol_within** (ex.: 0.5)
- Comparação com **baseline**: prever a média do treino

Isto encaixa perfeito na lógica de avaliação:

- métricas contínuas (RMSE/MAE/R^2)
- uma métrica "interpretável" (acerto a meio ponto)
- baseline como referência mínima

---

5) Restrições e qualidade dos dados (alto nível)

Aqui o script faz "data quality checks" iniciais:

Duplicados

```
n_dup <- sum(duplicated(df))
```

Missing values

```
miss <- missing_summary_df(df)
```

Usa a função do `Utils` para gerar contagens e percentagens por variável. Depois cria uma nota textual (`miss_note`) dependendo se existem missing ou não.

Coerência do target (escala 1–5)

- `target_rng`: range observado
- `target_outside`: quantos estão fora de [1,5]

Isto é super importante para o relatório porque mostra que:

- conheces a escala do target
- verificaste consistência do dataset
- tens noção de possíveis problemas (ex.: dados errados, parsing, etc.)

---

6) Plano CRISP-DM (mapa do relatório)

O bloco `crisp_lines` é literalmente um "índice" do teu trabalho:

1. Business Understanding (este script)
2. Data Understanding (Cap. 3)
3. Data Preparation (Cap. 4)
4. Modeling (Cap. 5)
5. Evaluation (Cap. 6)
6. Deployment (fora do âmbito)

Isto é útil porque o professor consegue "seguir a narrativa" facilmente.

---

7) Outputs que o script guarda (muito importante para documentação)

7.1) `business_understanding.txt`

Um ficheiro texto completo com:

- objetivo do trabalho
- questões analíticas de negócio
- target e lista de preditoras
- hipóteses iniciais
- critérios de sucesso
- notas de qualidade
- resumo CRISP-DM

É o “texto base” para o Capítulo 1 do relatório.

7.2) `dicionario_variaveis.csv`

Tabela com:

- `variavel` (nome)
- `tipo` (a tipologia simplificada)
- `papel` (target ou preditor)
- `binaria` (sim/não, só para preditoras)
- `hipotese` (texto da hipótese, se preditor)

Este CSV é excelente para anexar ao relatório ou referenciar numa tabela.

7.3) `distribuicao_target.csv`

Frequência de cada valor do `score_review` (inclui NA se existirem). Isto documenta o **grau de desbalanceamento** do target (muito relevante para explicar performance e o porquê de usar estratificação).

7.4) `qualidade_dados_resumo.csv`

Leva:

- output do `missing_summary_df`
- uma coluna extra `duplicados_total_dataset` com o mesmo valor em todas as linhas (para ficar registado no ficheiro)

Serve como evidência objetiva de qualidade.

---

8) Bloco final (execução do capítulo)

No fim do script:

```
out_cap1 <- file.path(result_dir, "outputs_cap1")
if (exists("df_raw")) {
  run_cap1(df_raw, out_dir = out_cap1, target = "score_review", tol_within = 0.5)
} else {
  warning(...)
}
```