

PROGRAMAÇÃO AVANÇADA EM BIG DATA

PROGRAMAÇÃO AVANÇADA EM BIG DATA

Público-alvo: Estudantes do Centro Universitário FMU.

Objetivo: Esta apostila foi desenvolvida sob medida para os estudantes ingressos no curso de Ciência de Dados. Seu propósito fundamental é construir uma ponte sólida e acessível entre a lógica de programação essencial e as complexas arquiteturas de Big Data, Processamento Distribuído e Aprendizado de Máquinas.

Com base nas diretrizes do Plano de Ensino institucional e na experiência executiva e acadêmica de seu autor, este material didático adota uma abordagem gradual ("direto ao ponto"), desmistificando conceitos avançados por meio de linguagem fluida, ilustrações cotidianas, metáforas didáticas e múltiplos estudos de caso em Python.

Ao longo de 16 Unidades, o estudante aprenderá a tratar, manipular, analisar e modelar grandes volumes de dados. O material capacita o futuro cientista de dados não apenas a escrever algoritmos, mas a compreender o fluxo completo de dados (Data Pipelines), da análise exploratória básica aos sistemas de recomendação, deep learning e bancos NoSQL, alinhando a teoria com as reais exigências do mercado corporativo.

Professor: Robson Carmo, mestre pelo programa de mestrado profissional em Gestão e Tecnologia em Sistemas Produtivos do Centro Paula Souza, concluiu 3 MBAs (Inovação e Transformação Digital, Gestão de Empresarial, Engenharia de Software). Instrutor de certificações da SAFE, autor e coautor de vários livros e ebooks.

Lattes: <http://lattes.cnpq.br/8753112683856964>

Orcid: <https://orcid.org/0009-0002-7102-4993>

LLM utilizada: Gemini Flash – Agosto de 2026.

RESPOSTAS DOS EXERCÍCIOS

Unidade 1: Introdução ao Universo do Big Data e Análise Exploratória Primária

Questão 1: Alternativa A

Justificativa:

1. Grau de satisfação (Baixo, Médio, Alto): É uma qualidade categorizada que possui uma ordem clara -> Qualitativa Ordinal.
2. Tempo de navegação (em segundos): É uma medida contínua (pode ter casas decimais, ex: 12,45 segundos) -> Quantitativa Contínua.
3. Quantidade de itens: É resultado de uma contagem de números inteiros -> Quantitativa Discreta.
4. Estado (SP, RJ, MG): É uma categoria sem ordem hierárquica entre as siglas -> Qualitativa Nominal.

Questão 2: Alternativa A

Justificativa:

A asserção I é verdadeira: o valor 98 ms distorceu a média aritmética para 28,5 ms, valor muito superior a 6 dos 7 dados da amostra. A asserção II também é verdadeira e justifica perfeitamente a I: a mediana (que para este conjunto ordenado é 18 ms) não é afetada por outliers, sendo a medida central adequada para distribuições assimétricas ou com valores discrepantes.

Questão 3: Alternativa C

Justificativa:

I está Correta: Monitoramento contínuo em tempo real refere-se à Velocidade.

II está Correta: A convivência de múltiplos formatos (imagens, textos, tabelas) refere-se à Variedade.

III está Incorreta: O descarte de dados duplicados ou inconsistentes refere-se à Veracidade (qualidade do dado) e não ao Volume.

Questão 4: Alternativa C

Justificativa:

Primeiro, calcula-se o Intervalo Interquartil (IQR):

$$\text{IQR} = Q_3 - Q_1 = 8.000 - 4.000 = 4.000$$

Em seguida, aplica-se a fórmula do Limite Superior (LS):

$$\text{LS} = Q_3 + 1,5 \times \text{IQR}$$

$$\text{LS} = 8.000 + 1,5 \times (4.000) = 8.000 + 6.000 = 14.000$$

Portanto, qualquer salário acima de R\$ 14.000,00 é considerado um outlier.

Questão 5: Alternativa A

Justificativa:

1. Média ($\bar{x}$):

$$\bar{x} = \frac{10 + 12 + 14 + 16 + 18}{5} = \frac{70}{5} = 14$$

2. Desvios em relação à média ($x_i - \bar{x}$):

$$(10 - 14) = -4; \quad (12 - 14) = -2; \quad (14 - 14) = 0; \quad (16 - 14) = 2; \quad (18 - 14) = 4$$

3. Quadrado dos desvios ($x_i - \bar{x}$)²:

$$16 + 4 + 0 + 4 + 16 = 40$$

4. Variância Amostral (s^2): Dividida por $n-1 = 4$:

$$s^2 = \frac{40}{4} = 10$$

5. Desvio Padrão Amostral (s):

$$s = \sqrt{10} \approx 3,16$$

Questão 6: Alternativa B

Justificativa:

- Texto de avaliações limpo e convertido -> Dado textual não estruturado (String).
- Estrelas de 1 a 5 -> Categoria com ordem implícita (Qualitativa Ordinal).
- Contagem do número de palavras -> Valor inteiro proveniente de contagem (Quantitativa Discreta).

Questão 7: Alternativa E

Justificativa:

I está Correta: As barras mais altas estão localizadas nas faixas etárias centrais/iniciais (25 a 39 anos).

II está Correta: O Histograma é o gráfico padrão para representar distribuições de frequências de dados quantitativos contínuos ou agrupados.

III está Correta: A variação entre as faixas marcadas no eixo horizontal é constante: $25 - 18 = 7$; $32 - 25 = 7$; $39 - 32 = 7$, ou seja, amplitude de classe igual a 7 anos.

EXERCÍCIOS PRÁTICOS EM PYTHON

EXERCÍCIO PRÁTICO 1

Resolução Comentada em Python:

```
# Exercício Prático 1 - Classificação de Variáveis
print("=== CADASTRO DE ALUNO / USUÁRIO ===")

# Entrada de dados com conversão de tipos nativos
nome = input("Digite seu nome completo: ") # str
idade = int(input("Digite sua idade (em anos inteiros): ")) # int
escolaridade = input("Escolaridade (Médio, Graduação, Pós): ") # str
salario_pretendido = float(input("Digite sua pretensão salarial (R$): ")) # float

print("\n" + "="*40)
print("=== RESUMO E CLASSIFICAÇÃO DOS DADOS ===")
print("="*40)

# Exibição e rotulagem das variáveis
print(f"Nome: {nome} | Tipo: Qualitativa Nominal")
print(f"Idade: {idade} anos | Tipo: Quantitativa Discreta")
print(f"Escolaridade: {escolaridade} | Tipo: Qualitativa Ordinal")
print(f"Salário Pretendido: R$ {salario_pretendido:.2f} | Tipo: Quantitativa Contínua")
```

EXERCÍCIO PRÁTICO 2

Resolução Comentada em Python:

```
# Exercício Prático 2 - Calculadora de Medidas de Posição
tempos = [150, 120, 200, 120, 180, 120, 300, 150, 220]

# 1. Ordenação dos dados (ROL)
tempos_ordenados = sorted(tempos)
n = len(tempos_ordenados)

print(f"Dados Ordenados: {tempos_ordenados}")
print(f"Total de Observações (n): {n}")

# 2. Cálculo da Média
soma_total = sum(tempos_ordenados)
media = soma_total / n

# 3. Cálculo da Mediana
# Como n = 9 (ímpar), o elemento central fica no índice n // 2
indice_central = n // 2
mediana = tempos_ordenados[indice_central]

# 4. Cálculo da Moda (usando o método count da lista)
moda = tempos_ordenados[0]
max_frequencia = 0

for valor in tempos_ordenados:
    freq = tempos_ordenados.count(valor)
    if freq > max_frequencia:
        max_frequencia = freq
        moda = valor

print("\n--- RESULTADOS ESTATÍSTICOS ---")
print(f"Média Aritmética: {media:.2f} ms")
print(f"Mediana: {mediana:.2f} ms")
print(f"Moda: {moda} ms (Aparece {max_frequencia} vezes)")
```

EXERCÍCIO PRÁTICO 3

Resolução Comentada em Python:

```
# Exercício Prático 3 - Cálculo de Variabilidade
import math # Para utilizar a função de raiz quadrada math.sqrt()

faturamento = [10, 12, 14, 16, 18]
n = len(faturamento)

# 1. Média
media = sum(faturamento) / n

# 2. Soma dos Quadrados dos Desvios
soma_quadrados_desvios = 0
for x in faturamento:
    desvio = x - media
    soma_quadrados_desvios += desvio ** 2

# 3. Variância Amostral (divisão por n - 1)
variância_amostral = soma_quadrados_desvios / (n - 1)

# 4. Desvio Padrão Amostral
desvio_padrao_amostral = math.sqrt(variância_amostral)

print(f"Dados de Faturamento: {faturamento}")
print(f"Média de Faturamento: R$ {media:.2f} mil")
print(f"Variância Amostral: {variância_amostral:.2f}")
print(f"Desvio Padrão Amostral: R$ {desvio_padrao_amostral:.2f} mil")
```

EXERCÍCIO PRÁTICO 4

Resolução Comentada em Python:

```
# Exercício Prático 4 - Detector de Outliers via IQR
dados = [40, 42, 45, 48, 50, 52, 55, 58, 60, 180]
dados_ordenados = sorted(dados)
n = len(dados_ordenados)

# Cálculo simplificado de quartis por posição de índice
# Q1 representa a posição aproximada de 25% dos dados
# Q3 representa a posição aproximada de 75% dos dados
q1_idx = int(n * 0.25)
q3_idx = int(n * 0.75)

q1 = dados_ordenados[q1_idx]
q3 = dados_ordenados[q3_idx]

# Intervalo Interquartil
iqr = q3 - q1

# Limites do Boxplot
limite_inferior = q1 - 1.5 * iqr
limite_superior = q3 + 1.5 * iqr

# Identificação de Outliers
outliers = []
dados_limpos = []

for x in dados_ordenados:
    if x < limite_inferior or x > limite_superior:
        outliers.append(x)
    else:
        dados_limpos.append(x)

print("=== DETECÇÃO DE OUTLIERS COM REGRA DO BOXPLOT ===")
print(f"Dados Analisados: {dados_ordenados}")
print(f"Quartil 1 (Q1): {q1}")
print(f"Quartil 3 (Q3): {q3}")
print(f"Intervalo Interquartil (IQR): {iqr}")
print(f"Limite Inferior (LI): {limite_inferior}")
print(f"Limite Superior (LS): {limite_superior}")
print("-" * 45)
print(f"Outliers Detectados: {outliers}")
print(f"Dados Sem Outliers: {dados_limpos}")
```

EXERCÍCIOS DE MÚLTIPLA ESCOLHA

Questão 1: Alternativa B

Justificativa:

Tuplas (tuple) são estruturas imutáveis em Python, ideais para armazenar dados fixos que não devem ser alterados durante a execução do programa (como coordenadas geográficas de centros de distribuição). Dicionários (dict) trabalham com pares chave-valor, sendo a estrutura nativa perfeita para mapear códigos de produtos a seus respectivos preços.

Questão 2: Alternativa E

Justificativa:

I está Correta: Quando $X = 0$, $\hat{Y} = 15,0$ (ou seja, R\$ 15.000,00).

II está Correta: O coeficiente angular $\beta_1 = 4,5$ representa a taxa de variação marginal de Y em relação a X.

III está Correta: Substituindo $X = 10$ na equação: $\hat{Y} = 15,0 + 4,5(10) = 15,0 + 45,0 = 60,0$ (R\$ 60.000,00).

Questão 3: Alternativa C

Justificativa:

O Coeficiente de Determinação é o quadrado do Coeficiente de Correlação de Pearson ($R^2 = r^2$). Assim: $R^2 = (0,85)^2 = 0,7225$, o que significa que 72,25% da variação da variável dependente é explicada pela variável independentes. Vale ressaltar que **correlação não implica causalidade direta e exclusiva** (eliminando as alternativas B e E).

Questão 4: Alternativa A

Justificativa:

Ambas as asserções são verdadeiras e a II justifica a I. No Modelo B, ao adicionar 5 novas variáveis, o R^2 bruto subiu ligeiramente (de 0,82 para 0,83), mas o R^2 Ajustado despencou (de 0,80 para 0,74). Isso indica que as variáveis adicionadas não trouxeram ganho explicativo real e apenas adicionaram complexidade desnecessária (ruído). O Modelo A é mais parcimonioso e eficiente.

Questão 5: Alternativa C

Justificativa:

Quando a variância dos erros não é constante ao longo das observações (apresentando variabilidade crescente ou decrescente, como o formato de leque), ocorre a Heterocedasticidade. A premissa ideal da regressão linear é a Homocedasticidade (variância constante dos resíduos).

Questão 6: Alternativa A

Justificativa:

1. Vamos calcular as diferenças $(X_i - \bar{X})$ e $(Y_i - \bar{Y})$:
 - Para $i = 1$: $(1 - 2,5) = -1,5$ e $(3 - 6) = -3 \rightarrow (-1,5) \times (-3) = 4,5$; $(-1,5)^2 = 2,25$
 - Para $i = 2$: $(2 - 2,5) = -0,5$ e $(5 - 6) = -1 \rightarrow (-0,5) \times (-1) = 0,5$; $(-0,5)^2 = 0,25$
 - Para $i = 3$: $(3 - 2,5) = 0,5$ e $(7 - 6) = 1 \rightarrow (0,5) \times (1) = 0,5$; $(0,5)^2 = 0,25$
 - Para $i = 4$: $(4 - 2,5) = 1,5$ e $(9 - 6) = 3 \rightarrow (1,5) \times (3) = 4,5$; $(1,5)^2 = 2,25$
2. Somatório do produto dos desvios (Covariância): $4,5 + 0,5 + 0,5 + 4,5 = 10,0$
3. Somatório dos quadrados do desvio de X (Variância): $2,25 + 0,25 + 0,25 + 2,25 = 5,0$
4. Coeficiente Angular β_1 : $\beta_1 = \frac{10,0}{5,0} = 2,0$
5. Coeficiente Linear β_0 : $\beta_0 = \bar{Y} - \beta_1 \bar{X} = 6,0 - (2,0 \times 2,5) = 6,0 - 5,0 = 1,0$

Questão 7: Alternativa C

Justificativa:

O laço for percorre a lista de dicionários dados_vendas. Ele verifica se o valor associado à chave "cliente" é igual a "A". A condição é satisfeita no primeiro elemento (valor 100.0) e no terceiro elemento (valor 150.0). A variável total_A acumula a soma: 100.0 + 150.0 = 250.0.

EXERCÍCIOS PRÁTICOS EM PYTHON

EXERCÍCIO PRÁTICO 1

Resolução Comentada em Python:

```
# Exercício Prático 1 - Regressão Linear Simples via MQO
x = [10, 20, 30, 40, 50]
y = [12, 22, 32, 42, 52]
n = len(x)

# 1. Médias
media_x = sum(x) / n
media_y = sum(y) / n

# 2. Cálculo da Covariância e Variância de X
num_cov = 0.0
den_var_x = 0.0

for i in range(n):
    num_cov += (x[i] - media_x) * (y[i] - media_y)
    den_var_x += (x[i] - media_x) ** 2

# 3. Coeficientes MQO
beta_1 = num_cov / den_var_x
beta_0 = media_y - (beta_1 * media_x)

# 4. Projeção para X = 35
x_novo = 35
y_previsto = beta_0 + (beta_1 * x_novo)

print("=== REGRESSÃO LINEAR SIMPLES (MQO) ===")
print(f"Equação da Reta: Y = {beta_0:.2f} + {beta_1:.2f} * X")
print(f"Projeção para {x_novo} mil linhas: {y_previsto:.2f} minutos")
```

EXERCÍCIO PRÁTICO 2

Resolução Comentada em Python:

```
# Exercício Prático 2 - Cálculo do R² do Modelo
y_real = [10, 15, 20, 25, 30]
y_previsto = [11, 14, 21, 24, 30]
n = len(y_real)

# 1. Média dos valores reais
media_y = sum(y_real) / n

# 2. Soma dos Quadrados Totais (SQT) e dos Resíduos (SQR)
sqt = 0.0
```

```
sqr = 0.0

for i in range(n):
    sqt += (y_real[i] - media_y) ** 2
    sqr += (y_real[i] - y_previsto[i]) ** 2

# 3. Cálculo do R²
r2 = 1 - (sqr / sqt)

print("=== AVALIAÇÃO DE QUALIDADE DO MODELO ===")
print(f"Soma dos Quadrados Totais (SQT): {sqt:.2f}")
print(f"Soma dos Quadrados dos Resíduos (SQR): {sqr:.2f}")
print(f"Coefficiente de Determinação (R²): {r2:.4f} ({r2 * 100:.2f}%)")
```

EXERCÍCIO PRÁTICO 3

Resolução Comentada em Python:

```
# Exercício Prático 3 - Previsão Múltipla com Funções e Estruturas Nativas

# Coeficientes pré-calculados do modelo
coeficientes = {
    'beta_0': 150.0,
    'beta_1': 12.50,
    'beta_2': 8.00
}

# Novos cenários de teste: (X1: Transações em milhares, X2: Temp em °C)
novos_cenarios = [
    (10, 22), # Cenário 1: 10 mil transações, 22°C
    (15, 25), # Cenário 2: 15 mil transações, 25°C
    (20, 20) # Cenário 3: 20 mil transações, 20°C
]

def projetar_fatura(coef, cenarios):
    previsoes = []
    for x1, x2 in cenarios:
        y_hat = coef['beta_0'] + (coef['beta_1'] * x1) + (coef['beta_2'] * x2)
        previsoes.append(y_hat)
    return previsoes

# Execução da projeção
faturas = projetar_fatura(coeficientes, novos_cenarios)

print("=== PROJEÇÃO DE FATURA DE ENERGIA (REGRESSÃO MÚLTIPLA) ===")
for i, fatura in enumerate(faturas, start=1):
    trans, temp = novos_cenarios[i-1]
    print(f"Cenário {i} ({trans}k transações, {temp}°C) -> Fatura Estimada: R$ {fatura:.2f}")
```

EXERCÍCIO PRÁTICO 4

Resolução Comentada em Python:

Resolução Comentada em Python:

```
# Exercício Prático 4 - Diagnóstico de Resíduos
y_real = [10, 12, 15, 18, 22, 30, 35, 40]
y_prev = [10.5, 11.8, 15.2, 17.5, 20.0, 25.0, 28.0, 31.0]

# 1. Vetor de Resíduos (Erro = Real - Previsto)
residuos = [y_real[i] - y_prev[i] for i in range(len(y_real))]

# 2. Divisão em duas metades para análise de variância
meio = len(residuos) // 2
metade_1 = residuos[:meio]
metade_2 = residuos[meio:]

# 3. Cálculo do Desvio Médio Absoluto das duas metades
dma_1 = sum(abs(e) for e in metade_1) / len(metade_1)
dma_2 = sum(abs(e) for e in metade_2) / len(metade_2)

print("=== ANÁLISE DIAGNÓSTICA DOS RESÍDUOS ===")
print(f"Vetor de Resíduos: {[round(r, 2) for r in residuos]}")
print(f"Erro Médio Absoluto (Metade Inicial): {dma_1:.2f}")
print(f"Erro Médio Absoluto (Metade Final): {dma_2:.2f}")

if abs(dma_1 - dma_2) > 2.0:
    print("Alerta: Indícios de HETEROCEDASTICIDADE (Variância dos erros não é constante).")
else:
    print("Sucesso: Os resíduos indicam HOMOCEDASTICIDADE (Variância aproximadamente constante).")
```

EXERCÍCIOS DE MÚLTIPLA ESCOLHA

Questão 1: Alternativa B

Justificativa:

1. idade $\geq 18 \rightarrow 25 \geq 18 \rightarrow \text{True}$.
2. autenticado $\rightarrow \text{True}$.
3. not bloqueado $\rightarrow \text{not False} \rightarrow \text{True}$.
4. Avaliação lógica: $(\text{True and True}) \text{ and True} \rightarrow \text{True and True} \rightarrow \text{True}$.

Como a condição inteira é verdadeira, o bloco do if é executado, atribuindo "Acesso Permitido" à variável status e exibindo-a no terminal.

Questão 2: Alternativa A

Justificativa:

Ambas as asserções são verdadeiras e a II justifica perfeitamente a I. A Regressão Linear Simples pode projetar valores fora dos limites $[0, 1]$, o que viola a definição de probabilidade. A Regressão Logística soluciona exatamente esse problema ao utilizar a função sigmoide, restringindo o resultado ao intervalo probabilístico de 0 a 1.

Questão 3: Alternativa C

Justificativa:

Substituindo $z = 0$ na fórmula da função sigmoide:

$$p = \frac{1}{1 + e^0} = \frac{1}{1 + 1} = \frac{1}{2} = 0,50 \quad (50\%)$$

Com $p = 0,50$ e adotando o limiar de decisão padrão $p \geq 0,5$, o modelo classifica o caso como pertencente à classe positiva (1 - Cancelado).

Questão 4: Alternativa B

Justificativa:

A fórmula da Sensibilidade (Recall) é dada por:

$$\text{Sensibilidade} = \frac{VP}{VP + FN} = \frac{40}{40 + 5} = \frac{40}{45} \approx 0,8888 \quad (88,8\%)$$

Questão 5: Alternativa C

Justificativa:

Como o código utiliza múltiplos blocos if independentes (sem o encadeamento com elif):

1. if pontos > 200: (150 > 200 é Falso) → Ignorado.
2. if pontos > 100: (150 > 100 é Verdadeiro) → categoria recebe "Prata".
3. if pontos > 50: (150 > 50 é Verdadeiro) → Sobrescreve categoria para "Bronze".

Ao final, o valor mantido na memória e impresso é "Bronze". Isso ilustra na prática a importância de utilizar a estrutura if-elif-else para condições mutuamente exclusivas.

Questão 6: Alternativa B

Justificativa:

A transformação Logit é definida como o logaritmo natural do Odds (razão entre a probabilidade de ocorrência **p** e a probabilidade de não ocorrência **1 - p**):

$$\text{Logit}(p) = \ln \left(\frac{p}{1 - p} \right)$$

Questão 7: Alternativa D

Justificativa:

Em conjuntos de dados altamente desbalanceados, a acurácia é uma métrica enganosa. Se um modelo classificar todas as observações como pertencentes à classe majoritária, ele obterá uma alta acurácia (95%), mas falhará em 100% dos casos da classe de interesse (os doentes real), resultando em uma Sensibilidade de 0%.

EXERCÍCIOS PRÁTICOS EM PYTHON

EXERCÍCIO PRÁTICO 1

Resolução Comentada em Python:

```
# Exercício Prático 1 - Avaliador de Regras de Crédito

print("=== MOTOR DE REGRAS DE CRÉDITO ===")

# Entrada de dados
idade = int(input("Digite a idade do cliente: "))
renda = float(input("Digite a renda mensal (R$): "))
inadimplente_input = input("Possui histórico de inadimplência? (s/n): ").strip().lower()

# Conversão para booleano
inadimplente = True if inadimplente_input == 's' else False

# Avaliação condicional com operadores lógicos (and, or, not)
if idade >= 21 and renda >= 5000.0 and not inadimplente:
    classificacao = "Risco Baixo - Aprovado com Taxa Preferencial"
elif idade >= 21 and renda >= 2500.0:
    classificacao = "Risco Médio - Aprovado com Garantia"
else:
    classificacao = "Risco Alto - Crédito Negado"

print("\n" + "="*45)
print(f"Resultado da Análise: {classificacao}")
print("="*45)
```

EXERCÍCIO PRÁTICO 2

Resolução Comentada em Python:

```
# Exercício Prático 2 - Função Sigmoid e Classificação Probabilística
import math

def calcular_probabilidade_sigmoide(z):
    """Aplica a função sigmoide a um valor real z."""
    return 1.0 / (1.0 + math.exp(-z))

# Valores de z obtidos da combinação linear (z = beta_0 + beta_1*X1 + ...)
valores_z = [-3.5, -1.0, 0.0, 1.2, 4.0]
limiar_decisao = 0.5

print("=== CONVERSÃO SIGMOIDE E CLASSIFICAÇÃO BINÁRIA ===")
print(f"{'Valor z':<10} | {'Probabilidade (p)':<20} | {'Classe Prevista'}")
print("-" * 50)

for z in valores_z:
    p = calcular_probabilidade_sigmoide(z)
    # Aplicação do limiar condicional
    classe_prevista = 1 if p >= limiar_decisao else 0
    print(f"{z:<10.1f} | {p:<20.4f} ({p*100:.2f}%) | {classe_prevista}")
```

EXERCÍCIO PRÁTICO 3

Resolução Comentada em Python:

```
# Exercício Prático 3 - Cálculo de Odds e Logit
import math

def analisar_chance_e_logit(p):
    if p <= 0 or p >= 1:
        return "Erro: A probabilidade deve estar estritamente entre 0 e 1."

    # 1. Calculation of Odds (Razão de Chances)
    odds = p / (1.0 - p)

    # 2. Calculation of Logit (Logaritmo Natural das Odds)
    logit = math.log(odds)

    return odds, logit

# Teste com probabilidade de Churn de 80% (0.80)
probabilidade = 0.80
resultado = analisar_chance_e_logit(probabilidade)

if isinstance(resultado, tuple):
    odds, logit = resultado
    print("=== ANÁLISE DE CHANCE (ODDS) E LOGIT ===")
    print(f"Probabilidade (p): {probabilidade * 100:.1f}%")
    print(f"Razão de Chances (Odds): {odds:.2f} (A chance de cancelar é {odds:.0f}x maior do que a de permanecer)")
    print(f"Valor da Transformação Logit: {logit:.4f}")
```

EXERCÍCIO PRÁTICO 4

Resolução Comentada em Python:

```
# Exercício Prático 4 - Matriz de Confusão e Métricas de Avaliação

y_real = [1, 0, 1, 1, 0, 0, 1, 0, 0, 1]
y_pred = [1, 0, 1, 0, 0, 1, 1, 0, 0, 1]

# Inicialização dos contadores da Matriz de Confusão
vp = 0 # Verdadeiro Positivo (Real=1, Pred=1)
fp = 0 # Falso Positivo (Real=0, Pred=1)
vn = 0 # Verdadeiro Negativo (Real=0, Pred=0)
fn = 0 # Falso Negativo (Real=1, Pred=0)

# Comparação elemento a elemento
for i in range(len(y_real)):
    if y_real[i] == 1 and y_pred[i] == 1:
        vp += 1
    elif y_real[i] == 0 and y_pred[i] == 1:
        fp += 1
    elif y_real[i] == 0 and y_pred[i] == 0:
        vn += 1
    elif y_real[i] == 1 and y_pred[i] == 0:
        fn += 1

# Cálculo das métricas operacionais
total = len(y_real)
acuracia = (vp + vn) / total
precisao = vp / (vp + fp) if (vp + fp) > 0 else 0
sensibilidade = vp / (vp + fn) if (vp + fn) > 0 else 0

print("=== MATRIZ DE CONFUSÃO E AVALIAÇÃO DE DESEMPENHO ===")
print(f"Verdadeiros Positivos (VP): {vp}")
print(f"Falsos Positivos (FP): {fp}")
print(f"Verdadeiros Negativos (VN): {vn}")
print(f"Falsos Negativos (FN): {fn}")
print("-" * 45)
print(f"Acurácia Geral: {acuracia * 100:.2f}%")
print(f"Precisão: {precisao * 100:.2f}%")
print(f"Sensibilidade (Recall): {sensibilidade * 100:.2f}%")
```

Unidade 4: Estruturas de Repetição e Árvores de Decisão

EXERCÍCIOS DE MÚLTIPLA ESCOLHA

Questão 1: Alternativa B

Justificativa:

O laço for percorre os 5 elementos da lista saldos. Quando o valor é negativo (-300 e -150), a condição $\text{saldos} < 0$ é verdadeira e o comando continue ignora o restante do código da iteração, não executando o .append(). Apenas os saldos positivos (1200, 4500 e 8000) são adicionados à lista saldos_positivos. Portanto, len(saldos_positivos) retorna 3.

Questão 2: Alternativa A

Justificativa:

Ambas as asserções são verdadeiras e a II justifica perfeitamente a I. As Árvores de Decisão são consideradas modelos "caixa-branca" (interpretáveis) exatamente porque suas decisões se fundamentam no particionamento progressivo dos dados por meio de testes condicionais que buscam minimizar a impureza dos nós resultantes.

Questão 3: Alternativa A

Justificativa:

Nó A: $I_G = 1 - [(0,5)^2 + (0,5)^2] = 1 - [0,25 + 0,25] = 1 - 0,50 = 0,50$ (Máxima impureza para 2 classes).

Nó B: $I_G = 1 - [(1,0)^2 + (0,0)^2] = 1 - [1,0 + 0,0] = 1 - 1,0 = 0,00$ (Nó completamente puro).

Questão 4: Alternativa B

Justificativa:

A enorme disparidade entre o excelente desempenho no treino (100%) e o desempenho fraco no teste (62%) evidencia que a árvore memorizou excessivamente os dados de treino, caracterizando o Sobreajuste (Overfitting). Para corrigi-lo, deve-se limitar a profundidade máxima ou realizar a poda da árvore.

Questão 5: Alternativa B

Justificativa:

Iteração 1: tentativas = 1, sucesso = False. A condição do while (1 < 3 and not False) continua verdadeira.

Iteração 2: tentativas = 2. Como tentativas == 2, sucesso passa a ser True.

Verificação de Teste: Na verificação do laço, a condição not sucesso avalia para not True (→ False). O laço while é interrompido e o valor impresso para tentativas é 2.

Questão 6: Alternativa B

Justificativa:

A Entropia é a métrica fundamentada na Teoria da Informação utilizada pelos algoritmos ID3 e C4.5 para quantificar o grau de desordem, incerteza ou impureza em um conjunto de dados.

Questão 7: Alternativa D

Justificativa:

A pureza do nó folha é a proporção de observações pertencentes à classe correta em relação ao total atingido pelo nó:

$$\text{Pureza} = \frac{180 \text{ (Churn = SIM)}}{200 \text{ (Total no Nó)}} = 0,90 \quad (90\%)$$

EXERCÍCIOS PRÁTICOS EM PYTHON

EXERCÍCIO PRÁTICO 1

Resolução Comentada em Python:

```
# Exercício Prático 1 - Varredura Automática de Logs
logs_acesso = [
    {"ip": "192.168.1.10", "status": 200, "tempo_ms": 120},
    {"ip": "10.0.0.45", "status": 404, "tempo_ms": 50},
    {"ip": "172.16.0.8", "status": 200, "tempo_ms": 850},
    {"ip": "192.168.1.15", "status": 500, "tempo_ms": 1200},
    {"ip": "10.0.0.88", "status": 200, "tempo_ms": 90}
]

ips_alerta = []

print("=== INICIANDO VARREDURA DE LOGS DE REDE ===")
for log in logs_acesso:
    # Condição: Erro de Servidor/Cliente (HTTP >= 400) OU Lenteza (> 500ms)
    if log["status"] >= 400 or log["tempo_ms"] > 500:
```

```
ips_alerta.append(log["ip"])
print(f"[ALERTA] IP {log['ip']} -> Status: {log['status']} | Tempo: {log['tempo_ms']}ms")

print("\n--- RESUMO DE SEGURANÇA ---")
print(f"Total de requisições analisadas: {len(logs_acesso)}")
print(f"Total de IPs em estado de alerta: {len(ips_alerta)}")
```

EXERCÍCIO PRÁTICO 2

Resolução Comentada em Python:

```
# Exercício Prático 2 - Calculadora Nativa de Impureza de Gini

def calcular_gini(rotulos):
    n = len(rotulos)
    if n == 0:
        return 0.0

    # Identificar classes únicas
    classes_unicas = set(rotulos)
    soma_proporcoes_quadrado = 0.0

    for classe in classes_unicas:
        contagem = rotulos.count(classe)
        p_i = contagem / n
        soma_proporcoes_quadrado += p_i ** 2

    gini = 1.0 - soma_proporcoes_quadrado
    return gini

# Teste com diferentes cenários de nós
no_puro = ["Legítimo", "Legítimo", "Legítimo", "Legítimo"]
no_misto = ["Fraude", "Legítimo", "Fraude", "Legítimo"]
no_assimetrico = ["Fraude", "Fraude", "Fraude", "Legítimo"]

print("=== CALCULADORA DE IMPUREZA DE GINI ===")
print(f"Gini (Nó Puro): {calcular_gini(no_puro):.4f}")
print(f"Gini (Nó Misto 50/50): {calcular_gini(no_misto):.4f}")
print(f"Gini (Nó Assimétrico): {calcular_gini(no_assimetrico):.4f}")
```

EXERCÍCIO PRÁTICO 3

Resolução Comentada em Python:

```
# Exercício Prático 3 - Simulador de Regras de Árvore de Decisão

def classificar_credito(renda, inadimplente, score):
    # Nó Raiz: Teste de Renda
    if renda > 5000:
        # Nó Interno 1
        if not inadimplente:
            return "Aprovado - Limite Alto"
        else:
            return "Aprovado - Limite Médio"
    else:
        # Nó Interno 2
        if score > 700:
            return "Aprovado - Limite Inicial"
        else:
            return "Negado"

# Lote de clientes para teste
novos_clientes = [
    {"nome": "Lucas", "renda": 7500, "inadimplente": False, "score": 800},
    {"nome": "Mariana", "renda": 3200, "inadimplente": False, "score": 750},
    {"nome": "Roberto", "renda": 2800, "inadimplente": True, "score": 550}]

print("=== AVALIAÇÃO AUTOMÁTICA DE CRÉDITO ===")
```

```
for cliente in novos_clientes:
    resultado = classificar_credito(
        cliente["renda"],
        cliente["inadimplente"],
        cliente["score"]
    )
    print(f"Cliente: {cliente['nome']} -> Decisão da Árvore: {resultado}")
```

EXERCÍCIO PRÁTICO 4

Resolução Comentada em Python:

```
# Exercício Prático 3 - Simulador de Regras de Árvore de Decisão

def classificar_credito(renda, inadimplente, score):
    # Nó Raiz: Teste de Renda
    if renda > 5000:
        # Nó Interno 1
        if not inadimplente:
            return "Aprovado - Limite Alto"
        else:
            return "Aprovado - Limite Médio"
    else:
        # Nó Interno 2
        if score > 700:
            return "Aprovado - Limite Inicial"
        else:
            return "Negado"

# Lote de clientes para teste
novos_clientes = [
    {"nome": "Lucas", "renda": 7500, "inadimplente": False, "score": 800},
    {"nome": "Mariana", "renda": 3200, "inadimplente": False, "score": 750},
    {"nome": "Roberto", "renda": 2800, "inadimplente": True, "score": 550}
]

print("=== AVALIAÇÃO AUTOMÁTICA DE CRÉDITO ===")
for cliente in novos_clientes:
    resultado = classificar_credito(
        cliente["renda"],
        cliente["inadimplente"],
        cliente["score"]
    )
    print(f"Cliente: {cliente['nome']} -> Decisão da Árvore: {resultado}")
```

Unidade 5: Agrupamento de Dados (Clustering) com K-Means e Dendogramas

EXERCÍCIOS DE MÚLTIPLA ESCOLHA

Questão 1: Alternativa B

Justificativa:

Como a base de dados não possui rótulos ou categorias pré-definidas, trata-se de um problema clássico de Aprendizado Não Supervisionado. A técnica ideal para encontrar grupos naturais de compradores com base em seus atributos é o **Agrupamento (Clustering) via K-Means**.

Questão 2: Alternativa D

Justificativa:

A **asserção I é FALSA**: o K-Means é sensível à inicialização aleatória dos centroides e pode ficar preso em mínimos locais, não garantindo o mínimo global absoluto. A **asserção II é VERDADEIRA**: a etapa de atualização dos centroides recalcula a posição dos centros obtendo a média aritmética das coordenadas das observações de cada grupo.

Questão 3: Alternativa C

Justificativa:

Aplicando a fórmula da Distância Euclidiana entre A(10, 20) e B(13, 24):

$$d(A, B) = \sqrt{(13 - 10)^2 + (24 - 20)^2} = \sqrt{(3)^2 + (4)^2} = \sqrt{9 + 16} = \sqrt{25} = 5,0$$

Questão 4: Alternativa B

Justificativa:

O ponto $K = 2$ representa o "cotovelo" da curva, onde há uma queda acentuada na inércia (de 800 para 400). A partir de $K = 2$, os ganhos adicionais na redução da inércia tornam-se marginais, indicando que $K = 2$ é a quantidade ideal de grupos.

Questão 5: Alternativa C

Justificativa:

Realizar um corte horizontal em uma determinada altura de distância em um dendograma secciona as ramificações verticais, definindo uma partição direta da

amostra em um número fixo de grupos (clusters) correspondente ao número de ramos cortados.

Questão 6: Alternativa B

Justificativa:

Por definição, o critério de Encadeamento Máximo (Complete Linkage) mede a maior distância possível entre qualquer ponto pertencente ao primeiro grupo e qualquer ponto pertencente ao segundo grupo.

Questão 7: Alternativa B

Justificativa:

O K-Means baseia-se em métricas de distância espacial (como a Distância Euclidiana). Se as variáveis estiverem em escalas muito diferentes (ex: R\$ 80.000 contra 80 anos), a variável de maior magnitude numérica dominará completamente o cálculo da distância, distorcendo o agrupamento. A padronização equaliza a importância de todas as variáveis no modelo.

EXERCÍCIOS PRÁTICOS EM PYTHON

EXERCÍCIO PRÁTICO 1

```
# Exercício Prático 1 - Calculadora Nativa de Distância Euclidiana
import math

def calcular_distancia_euclidiana(p1, p2):
    """Calcula a Distância Euclidiana entre dois pontos em N dimensões."""
    if len(p1) != len(p2):
        raise ValueError("Os pontos devem ter o mesmo número de dimensões.")

    soma_quadrados = 0.0
    for i in range(len(p1)):
        soma_quadrados += (p1[i] - p2[i]) ** 2

    return math.sqrt(soma_quadrados)

# Dados de teste: [Renda em R$ mil, Idade, Anos de Fidelidade]
cliente_A = (10.0, 25, 2)
cliente_B = (14.0, 28, 5)
cliente_C = (10.0, 32, 2)

print("=== CÁLCULO DE DISTÂNCIA EUCLIDIANA ===")
print(f"Cliente A: {cliente_A}")
print(f"Cliente B: {cliente_B}")
print(f"Cliente C: {cliente_C}")

distancia = calcular_distancia_euclidiana(cliente_A, cliente_B)
print(f"Distância Espacial (A e B): {distancia:.4f}")

distancia = calcular_distancia_euclidiana(cliente_A, cliente_C)
print(f"Distância Espacial (A e C): {distancia:.4f}")
```

Questão 2

```
# Exercício Prático 2 - Etapa de Atribuição do K-Means
import math

# Centroides atuais: C0 e C1
centroides = {
    0: (10.0, 20.0), # Centroide do Grupo 0
    1: (50.0, 60.0) # Centroide do Grupo 1
}

# Novos clientes: (Renda em R$ mil, Gasto em R$ mil)
clientes = [
    (12.0, 22.0),
    (48.0, 58.0),
    (15.0, 18.0),
    (52.0, 65.0)
]

def encontrar_centroide_proximo(ponto, centroides_dict):
    menor_distancia = float('inf')
    grupo_designado = None

    for grupo, coord_centroide in centroides_dict.items():
        # Distância Euclidiana 2D
        dist = math.sqrt((ponto[0] - coord_centroide[0])**2 + (ponto[1] - coord_centroide[1])**2)
        if dist < menor_distancia:
            menor_distancia = dist
            grupo_designado = grupo

    return grupo_designado

print("=== ATRIBUIÇÃO AUTOMÁTICA AOS CLUSTERS ===")
for i, cliente in enumerate(clientes, start=1):
    grupo = encontrar_centroide_proximo(cliente, centroides)
    print(f"Cliente {i} {cliente} -> Alocado no Grupo: {grupo}")
```

Questão 3

```
# Exercício Prático 3 - Recálculo de Centroides

membros_grupo_0 = [
    (10.0, 20.0),
    (12.0, 22.0),
    (14.0, 18.0)
]

def recalcular_centroide(pontos):
    n = len(pontos)
    if n == 0:
        return None

    dimensoes = len(pontos[0])
    novas_coordenadas = []

    for d in range(dimensoes):
        soma_dim = sum(ponto[d] for ponto in pontos)
        media_dim = soma_dim / n
        novas_coordenadas.append(media_dim)

    return tuple(novas_coordenadas)

novo_centroide_0 = recalcular_centroide(membros_grupo_0)

print("=== ATUALIZAÇÃO DO CENTROIDE ===")
print(f"Pontos do Grupo: {membros_grupo_0}")
print(f"Nova Posição do Centroide: {novo_centroide_0}")
```

Questão 4

```
# Exercício Prático 4 - Padronizador Z-Score Nativo
import math

salarios = [2000.0, 2500.0, 3000.0, 10000.0, 15000.0]
n = len(salarios)

# 1. Média
media = sum(salarios) / n

# 2. Desvio Padrão Amostral
soma_quad_desvios = sum((x - media)**2 for x in salarios)
desvio_padrao = math.sqrt(soma_quad_desvios / (n - 1))

# 3. Conversão para Z-Score
salarios_padronizados = [(x - media) / desvio_padrao for x in salarios]

print("=== PADRONIZAÇÃO DE DADOS PARA K-MEANS ===")
print(f"Salários Originais: {salarios}")
print(f"Média: R$ {media:.2f} | Desvio Padrão: R$ {desvio_padrao:.2f}")
print("-" * 50)
print("Valores Padronizados (Z-Score):")
for orig, z in zip(salarios, salarios_padronizados):
    print(f"Original: R$ {orig:<8.2f} -> Z-Score: {z:+.4f}")
```

EXERCÍCIOS DE MÚLTIPLA ESCOLHA

Questão 1: Alternativa B

Justificativa: A eliminação de palavras funcionais que não possuem carga semântica relevante (como artigos, preposições e pronomes) é denominada Remoção de Stopwords.

Questão 2: Alternativa C

Justificativa: A métrica TF-IDF valoriza termos que são frequentes no documento individual analisado (TF alto), mas que são raros no conjunto geral de documentos (IDF alto), destacando as palavras mais informativas do texto.

Questão 3: Alternativa B

Justificativa:

- **Suporte:** $\text{Suporte}(\text{Café} \cap \text{Biscoito}) = \frac{100}{1000} = 0,10$ (10%).
- **Confiança:** $\text{Confiança}(\text{Café} \Rightarrow \text{Biscoito}) = \frac{\text{Suporte}(\text{Café} \cap \text{Biscoito})}{\text{Suporte}(\text{Café})} = \frac{100}{200} = 0,50$ ou $\frac{0,10}{0,15}$ para os totais. Recalculando: $\frac{100}{200} = 0,50$ do total de Café, o que equivale a 50% dos compradores de Café compram Biscoito. Analisando as alternativas com Suporte 10% e Confiança exata de $100/150 = 66,7\%$ para a direção inversa.

Questão 4: Alternativa A

Justificativa: Ambas as asserções são verdadeiras e a II justifica perfeitamente a I. Um $\text{Lift} > 1,0$ indica que a taxa de coocorrência é superior àquela que ocorreria de forma independente ao acaso.

Questão 5: Alternativa A

Justificativa: O algoritmo Apriori utiliza a Propriedade da Anti-monotonia (ou propriedade Apriori), que estabelece que se um conjunto de itens é infrequente, nenhum de seus superconjuntos poderá ser frequente.

Questão 6: Alternativa B

Justificativa: A Tokenização é o processo de fragmentação de um texto em uma lista de termos ou unidades menores (tokens).

Questão 7: Alternativa B

Justificativa: Quando o Lift < 1,0, significa que os itens possuem uma associação negativa (a presença do primeiro item reduz a chance de compra do segundo em relação à média geral). Portanto, a regra não deve ser utilizada para recomendação cruzada.

EXERCÍCIOS PRÁTICOS EM PYTHON

EXERCÍCIO PRÁTICO 1: Limpeza e Contador de Frequência de Palavras (Tokens)

Exercício Prático 1 - Contador de Frequência de Palavras (Tokens)

```
texto_avaliacao = "O produto é excelente, excelente qualidade e entrega rápida!"
stopwords = {"o", "é", "e"}
```

1. Normalização e remoção de pontuação

```
texto_limpo = texto_avaliacao.lower().replace(",", "").replace("!", "")
```

2. Tokenização

```
tokens = texto_limpo.split()
```

3. Filtragem de Stopwords e Contagem de Frequência

```
frequencia = {}
```

```
for token in tokens:
```

```
    if token not in stopwords:
```

```
        frequencia[token] = frequencia.get(token, 0) + 1
```

```
print("=== FREQUÊNCIA DE TOKENS NO TEXTO ===")
```

```
print(f"Texto Original: '{texto_avaliacao}'")
```

```
print("Contagem de Palavras Relevantes:")
```

```
for palavra, qtd in frequencia.items():
```

```
    print(f" - Palavra: '{palavra}': {qtd}x")
```

EXERCÍCIO PRÁTICO 2: Calculadora de TF-IDF Simplificada para 2 Documentos

Exercício Prático 2 - Cálculo Simplificado de TF-IDF

```
import math
```

```
doc1 = ["big", "data", "python"]
```

```
doc2 = ["python", "codigo", "dados"]
```

```
corpus = [doc1, doc2]
```

```
palavra_alvo = "big"
```

1. Cálculo do TF no Documento 1

```
tf_doc1 = doc1.count(palavra_alvo) / len(doc1)
```

2. Cálculo do IDF no Corpus

```
docs_com_palavra = sum(1 for doc in corpus if palavra_alvo in doc)
```

```
idf = math.log(len(corpus) / docs_com_palavra) if docs_com_palavra > 0 else 0
```

3. TF-IDF

```
tfidf = tf_doc1 * idf
```

```
print("=== CÁLCULO DE TF-IDF ===")
```

```
print(f"Palavra Analisada: '{palavra_alvo}'")
```

```
print(f"TF no Doc 1: {tf_doc1:.4f}")
```

```
print(f"IDF no Corpus: {idf:.4f}")
```

```
print(f"TF-IDF Resultante: {tfidf:.4f}")
```

EXERCÍCIO PRÁTICO 3: Verificador de Suporte de Pares de Produtos

```
# Exercício Prático 3 - Calculadora de Suporte Conjunto

compras = [
    ["pão", "leite", "manteiga"],
    ["pão", "manteiga"],
    ["leite", "café"],
    ["pão", "leite"]
]

def calcular_suporte_par(prod1, prod2, base_transacoes):
    total = len(base_transacoes)
    ocorrencias = 0
    for t in base_transacoes:
        if prod1 in t and prod2 in t:
            ocorrencias += 1
    return ocorrencias / total

suporte_pao_leite = calcular_suporte_par("pão", "leite", compras)

print("=== CÁLCULO DE SUPORTE CONJUNTO ===")
print(f"Suporte do Par ('pão', 'leite'): {suporte_pao_leite:.2f} ({suporte_pao_leite*100:.0f}%)")
```

EXERCÍCIO PRÁTICO 4: Avaliador de Regras de Associação e Cálculo do Lift

```
# Exercício Prático 4 - Avaliador de Lift

def avaliar_regra_lift(sup_x, sup_y, sup_xy):
    confianca = sup_xy / sup_x if sup_x > 0 else 0
    lift = confianca / sup_y if sup_y > 0 else 0
    return confianca, lift

# Dados de teste: Regra {Cerveja} -> {Petiscos}
suporte_cerveja = 0.20
suporte_petiscos = 0.10
suporte_conjunto = 0.08

conf, lift = avaliar_regra_lift(suporte_cerveja, suporte_petiscos, suporte_conjunto)

print("=== AVALIAÇÃO DE LIFT E AÇÃO DE NEGÓCIO ===")
print(f"Confiança: {conf*100:.1f}%")
print(f"Lift: {lift:.2f}")

if lift > 1.0:
    print("Ação recomendada: Posicionar produtos próximos no layout da loja!")
else:
    print("Ação recomendada: Não realizar promoção conjunta.")
```

Unidade 7: Manipulação de Strings

EXERCÍCIOS DE MÚLTIPLA ESCOLHA

Questão 1: Alternativa B

Justificativa: O método `.strip()` elimina espaços excedentes e quebras de linha das extremidades da string, e o método `.title()` converte a primeira letra de cada palavra para maiúscula.

Questão 2: Alternativa B

Justificativa: O fatiamento `[10:13]` acessa os caracteres dos índices 10, 11 e 12 da string "PROD-8842-SUL-2026", que correspondem exatamente à palavra "SUL".

Questão 3: Alternativa A

Justificativa: Ambas as asserções são verdadeiras e a II justifica a I. Devido à imutabilidade das strings em Python, métodos como `.replace()` geram uma nova string sem alterar o objeto original em memória.

Questão 4: Alternativa C

Justificativa: O método `.endswith(sufixo)` verifica se a string termina com a sequência informada, retornando `True` ou `False`.

Questão 5: Alternativa B

Justificativa: Em Python, o método `.join()` é chamado a partir da string delimitadora (no caso "-"), recebendo a lista de strings como argumento: `"-".join(tokens)`.

Questão 6: Alternativa C

Justificativa: Em Regex, `\d` representa qualquer dígito numérico (0-9) e o quantificador `{3}` exige a repetição exata de 3 dígitos.

Questão 7: Alternativa B

Justificativa: Para que a string "R\$ 1.250,50" possa ser convertida para float em Python, é necessário remover o símbolo da moeda, eliminar o ponto de milhar (que causaria erro no Python) e substituir a vírgula decimal por ponto.

EXERCÍCIOS PRÁTICOS EM PYTHON

EXERCÍCIO PRÁTICO 1: Limpeza e Normalização de Textos de Cadastros

```
# Exercício Prático 1 - Normalização de Nomes de Estados

estados_brutos = [" são paulo \n", " RIO DE JANEIRO ", "minas gerais\t", " paraná "]

estados_limpos = []
for estado in estados_brutos:
    # Remove espaços/quebras de linha e converte para maiúsculas
    item_formatado = estado.strip().upper()
    estados_limpos.append(item_formatado)

print("=== ESTADOS PADRONIZADOS ===")
print(estados_limpos)
```

EXERCÍCIO PRÁTICO 2: Conversor de Valores Monetários em String para Float

```
# Exercício Prático 2 - Conversor de Moeda

def converter_moeda_para_float(texto_valor):
    # 1. Remover o símbolo "R$ " e espaços
    limpo = texto_valor.replace("R$", "").strip()
    # 2. Remover pontos de milhar
    limpo = limpo.replace(".", "")
    # 3. Substituir vírgula por ponto
    limpo = limpo.replace(",", ".")
    # 4. Converter para float
    return float(limpo)

# Teste da função
preco_str = "R$ 12.990,50"
preco_num = converter_moeda_para_float(preco_str)

print("=== CONVERSÃO MONETÁRIA ===")
print(f"String Original : '{preco_str}' (Tipo: {type(preco_str).__name__})")
print(f"Float Resultante: {preco_num} (Tipo: {type(preco_num).__name__})")
```

EXERCÍCIO PRÁTICO 3: Máscara Anonimizadora de CPF para LGPD

```
# Exercício Prático 3 - Anonizador LGPD

def anonimizar_cpf(cpf):
    # Garante que o CPF tenha os 14 caracteres com pontuação
    if len(cpf) == 14:
        # Pega os 3 primeiros caracteres e os 3 últimos
        inicio = cpf[:4] # "123."
        fim = cpf[-3:] # "-00"
        return inicio + "***.***" + fim
    else:
        return "CPF inválido"

cpf_cliente = "987.654.321-99"
cpf_protegido = anonimizar_cpf(cpf_cliente)

print("=== ANONIMIZAÇÃO DE DADOS (LGPD) ===")
print(f"CPF Original: {cpf_cliente}")
print(f"CPF MASCARADO: {cpf_protegido}")
```

EXERCÍCIO PRÁTICO 4: Extrator de E-mails com Expressões Regulares (Regex)

```
# Exercício Prático 4 - Extrator de E-mails via Regex
import re

texto_suporte = """
Olá equipe, favor entrar em contato com suporte@empresa.com ou com o diretor
no e-mail carlos.silva@provedor.com.br para alinhar o projeto.
"""

# Padrão Regex para identificar e-mails
padrao_email = r'[a-zA-C0-9._%+-]+@[a-zA-C0-9.-]+\.[a-zA-C]{2,}'
emails_encontrados = re.findall(padrao_email, texto_suporte, re.IGNORECASE)

print("=== E-MAILS EXTRAÍDOS DO TEXTO ===")
for i, email in enumerate(emails_encontrados, start=1):
    print(f"E-mail {i}: {email}")
```

FMU
LEVEL UP