Programação para Ciência de Dados

Análise de Dados Multidimensionais

Arthur Casals

13 de Novembro de 2025

Avisos

- Terceira entrega quinzenal: AMANHÃ 14/11!
- Notas

Agenda

- Introdução: O que são Dados Multidimensionais?
- Bloco 1: Tensores e Operações em 3D+
- Bloco 2: Imagens como Dados Multidimensionais
- Bloco 3: Outros Tipos de Dados 3D+
- Bloco 4: Pipeline Completo e Boas Práticas

O que são Dados Multidimensionais?

Até agora trabalhamos principalmente com:

- **1D:** Vetores, listas, séries temporais
- **2D:** Matrizes, DataFrames, tabelas

Mas muitos dados têm 3+ dimensões:

- **3D:** Imagens RGB, séries temporais multivariadas
- **4D:** Vídeos, batch de imagens
- **5D+:** Dados científicos complexos

Nota Importante

Dados multidimensionais = tensores = arrays com 3+ dimensões

Exemplos do Mundo Real

Imagens e Vídeos:

- Foto colorida: altura × largura × 3 canais (RGB)
- Vídeo: frames × altura × largura × canais
- Ressonância magnética: x × y × z (volumétrico)

Dados de Negócios:

- Vendas: produtos × regiões × períodos
- Clientes: features × segmentos × trimestres
- Estoque: SKUs × lojas × semanas

Dados Científicos:

- Clima: latitude × longitude × altitude × tempo
- Experimentos: amostras × condições × medições × repetições
- Sensores: tempo × variáveis × localizações

Por que Dados Multidimensionais Importam?

São ubíquos na ciência de dados:

- **Imagens e visão computacional**
  - Análise de imagens médicas
  - Processamento de fotos e vídeos
  - Reconhecimento de padrões visuais
- **Análise de negócios**
  - Análise de vendas multi-dimensional
  - Comportamento de clientes ao longo do tempo
  - Comparações entre múltiplas variáveis
- **Pesquisa científica**
  - Simulações físicas e químicas
  - Dados de experimentos complexos
  - Análise de séries temporais multivariadas

Progressão das Nossas Aulas

Evolução natural da complexidade:

`Aulas 3-4:` NumPy básico (1D, 2D) ✓

`Aulas 5-6:` Pandas (DataFrames 2D) ✓

`Aula 11:` Séries Temporais (1D temporal) ✓

`Aula 12:` Tensores (3D+) ← **HOJE**

Nota Importante

Hoje aprendemos a trabalhar com estruturas de dados complexas que têm múltiplas dimensões!

Dataset de Hoje: MNIST

Modified National Institute of Standards and Technology

O que é:

- Dataset clássico de dígitos manuscritos (0-9)
- 1797 imagens em grayscale (versão simplificada)
- 8×8 pixels por imagem
- Perfeito para demonstrar manipulação de tensores

Estrutura dos dados:

- Shape: (1797, 8, 8) = tensor 3D
- 1797 imagens × 8 pixels altura × 8 pixels largura
- Valores: 0-16 (intensidade de pixels)

Por que MNIST?

- Demonstra arrays 3D de forma visual
- Fácil de entender (são imagens)
- Permite análises interessantes

Objetivos de Aprendizado

Ao final desta aula, você será capaz de:

1. Entender o conceito de tensores e rank
2. Criar e manipular arrays 3D, 4D, e além
3. Usar o parâmetro `axis` com confiança
4. Trabalhar com imagens como arrays multidimensionais
5. Aplicar transformações em tensores
6. Analisar dados de negócios e científicos 3D+
7. Construir pipelines de processamento multi-dimensional
8. Otimizar operações com broadcasting

Nota Importante

Foco: manipulação e análise de dados, não modelagem estatística!

Estrutura da Aula

Blocos e tempo estimado:

Bloco 1: Tensores e Dados 3D+
- Conceitos fundamentais
- Operações em múltiplas dimensões
- Parâmetro axis em profundidade

Bloco 2: Imagens
- MNIST como tensor 3D
- Análise exploratória visual
- Filtros e transformações

Blocos 3-4: Outros Dados + Pipeline
- Séries temporais multivariadas
- Dados de vendas 3D
- Boas práticas e pipeline completo

Setup: Imports e Configurações

Python

```python
# Imports necessarios
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
# Configurar matplotlib
plt.style.use('default')
plt.rcParams['figure.figsize'] = (12, 6)
plt.rcParams['font.size'] = 10
# Configurar NumPy para prints mais legíveis
np.set_printoptions(precision=2, suppress=True)
# Seed para reprodutibilidade
np.random.seed(42)
print("Setup completo!")
```

Bloco 1

Tensores e Dados 3D+

O que é um Tensor?

Tensor = generalização de arrays para N dimensões

Hierarquia de tensores:

- **Rank 0 (0D):** Escalar - um único número
  - Exemplo: temperatura = 25.3
- **Rank 1 (1D):** Vetor - lista de números
  - Exemplo: [10, 20, 30, 40]
- **Rank 2 (2D):** Matriz - tabela de números
  - Exemplo: DataFrame, imagem grayscale
- **Rank 3+ (3D+):** Tensor - estrutura multi-dimensional
  - Exemplo: imagem RGB, vídeo, dados volumétricos

Nota Importante

Rank = número de dimensões = número de índices necessários

Criar Tensores: Exemplos Básicos

Python

```python
# Rank 0: Escalar
scalar = np.array(42)
print(f"Rank 0: {scalar.ndim}D, Shape: {scalar.shape}")
# Rank 0: 0D, Shape: ()

# Rank 1: Vetor
vector = np.array([1, 2, 3, 4, 5])
print(f"Rank 1: {vector.ndim}D, Shape: {vector.shape}")
# Rank 1: 1D, Shape: (5,)
```

Criar Tensores: Exemplos Básicos (cont.)

</> Python

```python
# Rank 2: Matriz
matrix = np.array([[1, 2, 3],
                   [4, 5, 6]])
print(f"Rank 2: {matrix.ndim}D, Shape: {matrix.shape}")
# Rank 2: 2D, Shape: (2, 3)

# Rank 3: Tensor 3D
tensor_3d = np.array([[[1, 2], [3, 4]],
                      [[5, 6], [7, 8]]])
print(f"Rank 3: {tensor_3d.ndim}D, Shape: {tensor_3d.shape}")
# Rank 3: 3D, Shape: (2, 2, 2)
```

Entendendo Shape em 3D

Python

```python
# Criar tensor 3D simples
tensor = np.random.rand(3, 4, 5)

print(f"Shape: {tensor.shape}")  # (3, 4, 5)
print(f"Rank (ndim): {tensor.ndim}")  # 3
print(f"Total de elementos: {tensor.size}")  # 60

# Interpretacao do shape (3, 4, 5):
# - Primeira dimensao: 3 "camadas" ou "blocos"
# - Segunda dimensao: 4 "linhas" em cada camada
# - Terceira dimensao: 5 "colunas" em cada linha
```

Entendendo Shape em 3D

Python

```python
# Acessar elemento especifico
elemento = tensor[0, 2, 3]
print(f"Elemento [0, 2, 3]: {elemento:.2f}")

# Acessar uma camada completa (2D)
camada_0 = tensor[0, :, :]  # ou tensor[0]
print(f"Shape da camada 0: {camada_0.shape}")  # (4, 5)

# Acessar uma linha especifica
linha = tensor[0, 1, :]
print(f"Shape da linha: {linha.shape}")  # (5,)
```

Visualizando Tensores 3D

Como pensar em um tensor 3D?

Analogia 1: Pilha de matrizes

- Tensor (3, 4, 5) = 3 matrizes de tamanho (4, 5)
- Cada ”camada”é uma matriz 2D
- Primeira dimensão = índice da camada

Analogia 2: Cubo de números

- 3 blocos na profundidade
- 4 linhas na altura
- 5 colunas na largura

Analogia 3: Imagens RGB

- Shape (altura, largura, canais)
- 3 canais: Red, Green, Blue
- Cada canal é uma matriz 2D

Casos de Uso: Tensores 3D

Onde encontramos tensores 3D?

1. **Imagens coloridas (RGB):**
   - Shape: (altura, largura, 3 canais)
   - Exemplo: (256, 256, 3) = imagem 256×256 RGB

2. **Imagens em grayscale (batch):**
   - Shape: (n_imagens, altura, largura)
   - Exemplo: (1797, 8, 8) = 1797 imagens 8×8
   - Este é o caso do MNIST!

Casos de Uso: Tensores 3D

Onde encontramos tensores 3D?

3. Séries temporais multivariadas:

- Shape: (tempo, variáveis, locais)
- Exemplo: (365, 5, 10) = 365 dias, 5 sensores, 10 cidades

4. Dados de vendas:

- Shape: (produtos, regiões, períodos)
- Exemplo: (100, 5, 12) = 100 produtos, 5 regiões, 12 meses

Casos de Uso: Tensores 4D+

Onde encontramos tensores 4D ou mais?

Tensores 4D:

- **Batch de imagens RGB:** (n, altura, largura, canais)
- **Vídeo:** (frames, altura, largura, canais)
- **Dados volumétricos temporais:** (tempo, x, y, z)

Tensores 5D:

- **Batch de vídeos:** (n_vídeos, frames, altura, largura, canais)
- **Simulações científicas:** (tempo, x, y, z, variáveis)

Nota Importante

NumPy suporta arrays de qualquer dimensão! Mas 3D-4D são os mais comuns na prática.

Criar Tensor 3D: Exemplo Sintético

Python

```python
# Exemplo: dados de temperatura de sensores
# 100 dias x 5 sensores x 3 horarios (manha, tarde, noite)

np.random.seed(42)

# Criar tensor 3D
# dias, sensores, horarios
temp_data = np.random.randn(100, 5, 3) * 5 + 20

print(f"Shape: {temp_data.shape}")  # (100, 5, 3)
print(f"Rank: {temp_data.ndim}")    # 3
print(f"Tamanho: {temp_data.size}") # 1500
```

Criar Tensor 3D: Exemplo Sintético (cont.)

Python

```python
# Estatisticas basicas
print(f"\nTemperatura media geral: {temp_data.mean():.1f}°C") # 20.2°C
print(f"Temperatura minima: {temp_data.min():.1f}°C") # 3.8°C
print(f"Temperatura maxima: {temp_data.max():.1f}°C") # 39.3°C
print(f"Desvio padrao: {temp_data.std():.1f}°C") # 4.9°C

# Acessar dia especifico (retorna matriz 2D)
dia_0 = temp_data[0]  # Shape: (5, 3)
print(f"\nShape do dia 0: {dia_0.shape}")
```

Criar Tensor 4D: Exemplo

Python

```python
# Exemplo: batch de imagens RGB
# 10 imagens x 64 pixels altura x 64 pixels largura x 3 canais

np.random.seed(42)

# Criar tensor 4D
# n_imagens, altura, largura, canais
images = np.random.randint(0, 256, size=(10, 64, 64, 3))

print(f"Shape: {images.shape}")  # (10, 64, 64, 3)
print(f"Rank: {images.ndim}")    # 4
print(f"Tamanho: {images.size}") # 122,880 pixels
```

Criar Tensor 4D: Exemplo (cont.)

Python

```python
# Acessar primeira imagem (retorna tensor 3D)
img_0 = images[0]  # Shape: (64, 64, 3)
print(f"Shape da imagem 0: {img_0.shape}")

# Acessar canal vermelho da primeira imagem
red_channel = images[0, :, :, 0]  # Shape: (64, 64)
print(f"Shape do canal vermelho: {red_channel.shape}")

# Estatisticas por canal
for i, cor in enumerate(['Red', 'Green', 'Blue']):
    mean_val = images[:, :, :, i].mean()
    print(f"{cor}: media = {mean_val:.1f}")
```

Indexação em Tensores 3D

Python

```python
# Criar tensor 3D
tensor = np.arange(24).reshape(2, 3, 4)

print("Shape:", tensor.shape)  # (2, 3, 4)
print("\nTensor completo:")
print(tensor)

# Indexacao basica
print("\n=== INDEXACAO ===")

# Acessar primeira "camada" (matriz 2D)
layer_0 = tensor[0]
print(f"tensor[0] shape: {layer_0.shape}")  # (3, 4)
```

Indexação em Tensores 3D

Python

```python
# Acessar elemento especifico
elem = tensor[1, 2, 3]
print(f"tensor[1, 2, 3] = {elem}")  # 23

# Acessar linha especifica
row = tensor[0, 1]
print(f"tensor[0, 1] shape: {row.shape}")  # (4,)

# Acessar coluna em todas as camadas
col = tensor[:, :, 0]
print(f"tensor[:, :, 0] shape: {col.shape}")  # (2, 3)
```

Slicing Complexo em 3D

</> Python

```python
# Tensor do slide anterior - Shape: (2, 3, 4)

print("=== SLICING ===")

# Slice em uma dimensao
slice_1 = tensor[0, :, :]  # Primeira camada completa
print(f"tensor[0, :, :] shape: {slice_1.shape}")  # (3, 4)

# Slice em multiplas dimensoes
slice_2 = tensor[:, 1:, 2:]  # Todas camadas, linha 1+, coluna 2+
print(f"tensor[:, 1:, 2:] shape: {slice_2.shape}")  # (2, 2, 2)
```

Slicing Complexo em 3D

Python

```python
# Slice com step
slice_3 = tensor[::1, ::2, ::2]  # Pegar elementos alternados
print(f"tensor[::1, ::2, ::2] shape: {slice_3.shape}")  # (2, 2, 2)

# Combinar index e slice
slice_4 = tensor[0, :2, 1:3]  # Camada 0, primeiras 2 linhas, colunas 1-2
print(f"tensor[0, :2, 1:3] shape: {slice_4.shape}")  # (2, 2)
print(slice_4)
```

O Parâmetro ’axis’: Conceito Fundamental

axis define a direção da operação

Para um tensor 3D com shape (A, B, C):

- `axis=0` $\rightarrow$ opera ao longo da primeira dimensão (A)
- `axis=1` $\rightarrow$ opera ao longo da segunda dimensão (B)
- `axis=2` $\rightarrow$ opera ao longo da terceira dimensão (C)
- `axis=-1` $\rightarrow$ última dimensão (mesma que axis=2)
- `axis=-2` $\rightarrow$ penúltima dimensão (mesma que axis=1)

Resultado após operação com axis:

- A dimensão especificada **desaparece**
- Shape resultante: remove a dimensão do axis
- Exemplo: (2, 3, 4) com axis=1 $\rightarrow$ (2, 4)

O Parâmetro ’axis’: Conceito Fundamental

Nota Importante

Entender ’axis’ é crucial para operações em tensores!

Parâmetro 'axis': Exemplos com sum()

Python

```python
# Criar tensor 3D simples
tensor = np.arange(24).reshape(2, 3, 4)
print(f"Shape original: {tensor.shape}")  # (2, 3, 4)
print(f"Soma total: {tensor.sum()}")      # 276

# Sem axis: soma tudo (retorna escalar)
total = tensor.sum()
print(f"\nsum() sem axis: {total}, shape: {np.array(total).shape}")

# axis=0: soma ao longo da primeira dimensao
sum_axis0 = tensor.sum(axis=0)
print(f"\nsum(axis=0) shape: {sum_axis0.shape}")  # (3, 4)
# Resultado: soma as 2 camadas, restam (3, 4)
```

Parâmetro 'axis': Exemplos com sum() (cont.)

Python

```python
# axis=1: soma ao longo da segunda dimensao
sum_axis1 = tensor.sum(axis=1)
print(f"sum(axis=1) shape: {sum_axis1.shape}")  # (2, 4)
# Resultado: soma as 3 linhas, restam (2, 4)

# axis=2: soma ao longo da terceira dimensao
sum_axis2 = tensor.sum(axis=2)
print(f"sum(axis=2) shape: {sum_axis2.shape}")  # (2, 3)
# Resultado: soma as 4 colunas, restam (2, 3)
```

Parâmetro 'axis': Visualização Gráfica

Python

```python
# Exemplo visual com dados de temperatura
# Shape: (dias, sensores, horarios) = (100, 5, 3)

np.random.seed(42)
temp_data = np.random.randn(100, 5, 3) * 5 + 20

print(f"Shape original: {temp_data.shape}")  # (100, 5, 3)

# axis=0: media ao longo dos dias
# Resultado: temperatura media de cada sensor em cada horario
mean_dias = temp_data.mean(axis=0)
print(f"\nmean(axis=0) - media por dia:")
print(f"Shape: {mean_dias.shape}")  # (5, 3)
print("Interpretacao: 5 sensores × 3 horarios")
```

Parâmetro 'axis': Visualização Gráfica

Python

```python
# axis=1: media ao longo dos sensores
# Resultado: temperatura media em cada horario de cada dia
mean_sensores = temp_data.mean(axis=1)
print(f"\nmean(axis=1) - media por sensor:")
print(f"Shape: {mean_sensores.shape}")  # (100, 3)
print("Interpretacao: 100 dias × 3 horarios")

# axis=2: media ao longo dos horarios
# Resultado: temperatura media diaria de cada sensor
mean_horarios = temp_data.mean(axis=2)
print(f"\nmean(axis=2) - media por horario:")
print(f"Shape: {mean_horarios.shape}")  # (100, 5)
print("Interpretacao: 100 dias × 5 sensores")
```

Outras Operações com ’axis’

</> Python

```python
tensor = np.random.randn(10, 20, 30)
print(f"Shape: {tensor.shape}")  # (10, 20, 30)

# Diferentes operacoes com axis

# 1. Maximo ao longo de cada eixo
max_0 = tensor.max(axis=0)  # Shape: (20, 30)
max_1 = tensor.max(axis=1)  # Shape: (10, 30)
max_2 = tensor.max(axis=2)  # Shape: (10, 20)

print(f"max(axis=0): {max_0.shape}")
print(f"max(axis=1): {max_1.shape}")
print(f"max(axis=2): {max_2.shape}")
```

Outras Operações com ’axis’ (cont.)

Python

```python
# 2. Desvio padrao
std_0 = tensor.std(axis=0)  # Shape: (20, 30)
print(f"\nstd(axis=0): {std_0.shape}")

# 3. Argmax (indice do valor maximo)
argmax_0 = tensor.argmax(axis=0)  # Shape: (20, 30)
print(f"argmax(axis=0): {argmax_0.shape}")

# 4. Percentis
p95_1 = np.percentile(tensor, 95, axis=1)  # Shape: (10, 30)
print(f"percentile(95, axis=1): {p95_1.shape}")
```

Operações com Axis: keepdims em Profundidade

keepdims: preservar dimensões para broadcasting

Python

```python
# Criar tensor 3D
tensor = np.random.rand(10, 8, 8)  # Similar ao MNIST
print(f"Shape original: {tensor.shape}")  # (10, 8, 8)

# SEM keepdims
mean_no_keep = tensor.mean(axis=(1, 2))
print(f"mean(axis=(1,2)): {mean_no_keep.shape}")  # (10,)

# COM keepdims
mean_with_keep = tensor.mean(axis=(1, 2), keepdims=True)
print(f"mean(axis=(1,2), keepdims=True): {mean_with_keep.shape}")  # (10,
    1, 1)
```

Operações com Axis: keepdims em Profundidade (cont.)

Python

```python
# Por que isso importa? BROADCASTING!
# Sem keepdims: broadcasting falharia
# Com keepdims: broadcasting funciona perfeitamente

# Normalizar cada imagem pela sua propria media
tensor_normalized = tensor / mean_with_keep  # Broadcasting (10,8,8) / (10,1,1)
print(f"Normalizado: {tensor_normalized.shape}")  # (10, 8, 8)
```

keepdims: Comparação Visual

Python

```python
# Exemplo pratico: normalizar cada elemento do dataset
X = np.random.rand(10, 8, 8)
# ABORDAGEM 1: Sem keepdims (requer reshape manual)
means_no_keep = X.mean(axis=(1, 2))  # (10,)
# Precisa expandir manualmente:
means_expanded = means_no_keep[:, np.newaxis, np.newaxis]  # (10, 1, 1)
X_norm_1 = X / means_expanded
# ABORDAGEM 2: Com keepdims (direto!)
means_with_keep = X.mean(axis=(1, 2), keepdims=True)  # (10, 1, 1)
X_norm_2 = X / means_with_keep
# Verificar equivalencia
print(f"Resultados identicos: {np.allclose(X_norm_1, X_norm_2)}")

# keepdims = menos codigo, mais claro!
```

keepdims: Comparação Visual

Nota Importante

keepdims=True simplifica broadcasting - use sempre que for normalizar!

Agregações Encadeadas

Combinar múltiplas operações de agregação:

</> Python

```python
# Tensor 3D: (amostras, altura, largura)
X = np.random.rand(100, 8, 8)

# Agregacao encadeada: media -> desvio padrao
# Passo 1: Media espacial de cada amostra
spatial_means = X.mean(axis=(1, 2))  # (100,)

# Passo 2: Desvio padrao das medias
overall_std = spatial_means.std()

print(f"Passo 1 - Medias espaciais: shape {spatial_means.shape}") # (100,)
print(f"Passo 2 - Std das medias: {overall_std:.4f}") # 0.0364
```

Agregações Encadeadas (cont.)

Python

```python
# Alternativa: tudo de uma vez (menos controle)
overall_mean = X.mean()
print(f"\nMedia global direta: {overall_mean:.4f}") # 0.5005

# Diferenca: agregacao encadeada permite analise intermediaria!
```

Agregações Condicionais

Combinar máscaras booleanas com agregações:

Python

```python
# Dados de temperatura: (100 dias, 5 sensores)
temp_data = np.random.randn(100, 5) * 5 + 20

# Pergunta: Quantos dias cada sensor esteve acima de 25°C?
hot_days_mask = temp_data > 25  # (100, 5) boolean

# Contar por sensor (agregar axis=0)
hot_days_per_sensor = hot_days_mask.sum(axis=0)
print(f"Dias quentes por sensor: {hot_days_per_sensor}")
```

Agregações Condicionais (cont.)

Python

```python
# Media de temperatura APENAS em dias quentes
# Usar np.where para manter shape
temp_hot_only = np.where(hot_days_mask, temp_data, np.nan)
mean_on_hot_days = np.nanmean(temp_hot_only, axis=0)
print(f"\nMedia em dias quentes: {mean_on_hot_days}")

# Percentual de dias quentes
pct_hot = 100 * hot_days_per_sensor / len(temp_data)
print(f"\nPercentual dias quentes: {pct_hot}%")
```

Normalização: Múltiplas Estratégias

Diferentes formas de normalizar tensores 3D:

</> Python

```python
X = np.random.rand(100, 8, 8) * 10 + 5  # (100, 8, 8)

# ESTRATEGIA 1: Normalizacao global (todo dataset)
X_norm_global = (X - X.mean()) / X.std()
print(f"Global: mean={X_norm_global.mean():.2e}, std={X_norm_global.std()
    :.2f}")

# ESTRATEGIA 2: Por amostra (cada imagem independente)
means_per_sample = X.mean(axis=(1,2), keepdims=True)  # (100, 1, 1)
stds_per_sample = X.std(axis=(1,2), keepdims=True)
X_norm_per_sample = (X - means_per_sample) / stds_per_sample
print(f"Por amostra: cada imagem tem mean=0, std=1")
```

Normalização: Múltiplas Estratégias

Diferentes formas de normalizar tensores 3D:

</> Python

```python
# ESTRATEGIA 3: Por pixel (atraves do dataset)
means_per_pixel = X.mean(axis=0, keepdims=True)  # (1, 8, 8)
stds_per_pixel = X.std(axis=0, keepdims=True)
X_norm_per_pixel = (X - means_per_pixel) / stds_per_pixel
print(f"Por pixel: cada posicao espacial normalizada")

# Qual usar? Depende da aplicacao!
```

Normalização: Quando Usar Cada Estratégia

Escolher estratégia baseado no objetivo:

1. Normalização Global:

- **Usar quando:** Comparar amostras diretamente
- **Exemplo:** Machine learning com features homogêneas
- **Efeito:** Todo dataset tem média 0, std 1

2. Normalização por Amostra:

- **Usar quando:** Cada amostra tem escala diferente
- **Exemplo:** Imagens com iluminação variável
- **Efeito:** Cada amostra independente tem média 0, std 1

Normalização: Quando Usar Cada Estratégia

Escolher estratégia baseado no objetivo:

3. Normalização por Pixel/Feature:

- **Usar quando:** Features têm importâncias diferentes
- **Exemplo:** Sensores com diferentes ranges
- **Efeito:** Cada posição/feature normalizada através do dataset

Nota Importante

Não existe ”melhor”estratégia - depende do problema!

Broadcasting: Revisão e Exemplos 3D

Broadcasting permite operações entre arrays de shapes diferentes

</> Python

```python
# Tensor 3D
tensor = np.random.rand(10, 8, 8)

# Broadcasting com escalar (sempre funciona)
result_1 = tensor + 5  # (10, 8, 8) + escalar
print(f"Tensor + escalar: {result_1.shape}")

# Broadcasting com vetor 1D
vec_1d = np.array([1, 2, 3, 4, 5, 6, 7, 8])  # (8,)
result_2 = tensor + vec_1d  # (10, 8, 8) + (8,)
print(f"Tensor + vetor (8,): {result_2.shape}")
# Broadcasting: (10, 8, 8) + (1, 1, 8) -> (10, 8, 8)
```

Broadcasting: Revisão e Exemplos 3D (cont.)

Python

```python
# Broadcasting com array 2D
arr_2d = np.ones((8, 8))  # (8, 8)
result_3 = tensor + arr_2d  # (10, 8, 8) + (8, 8)
print(f"Tensor + array (8,8): {result_3.shape}")
# Broadcasting: (10, 8, 8) + (1, 8, 8) -> (10, 8, 8)
```

Broadcasting: newaxis em Diferentes Posições

np.newaxis controla onde adicionar dimensões

Python

```python
# Vetor base
vec = np.array([1, 2, 3, 4, 5])  # (5,)

# Tensor alvo
tensor = np.random.rand(10, 5, 8)  # (10, 5, 8)

# POSICAO 1: newaxis no inicio
vec_expanded_1 = vec[np.newaxis, :, np.newaxis]  # (1, 5, 1)
result_1 = tensor * vec_expanded_1  # Broadcasting funciona!
print(f"Shape 1: {vec_expanded_1.shape} -> {result_1.shape}")
```

Broadcasting: newaxis em Diferentes Posições (cont.)

Python

```python
# POSICAO 2: newaxis no final
vec_expanded_2 = vec[:, np.newaxis]  # (5, 1)
# Nao funciona diretamente com (10, 5, 8)!
# Precisa adicionar outra dimensao
vec_expanded_2b = vec[np.newaxis, :, np.newaxis]  # (1, 5, 1)
result_2 = tensor * vec_expanded_2b
print(f"Shape 2: {vec_expanded_2b.shape} -> {result_2.shape}")

# DICA: Alinhar dimensoes da direita para esquerda
```

Broadcasting: Casos Complexos

Python

```python
# Aplicar pesos diferentes em cada dimensao

# Tensor: (10 amostras, 8 linhas, 8 colunas)
tensor = np.random.rand(10, 8, 8)

# Pesos por linha (eixo vertical)
weights_row = np.linspace(1.0, 0.5, 8)  # (8,)
weights_row = weights_row[np.newaxis, :, np.newaxis]  # (1, 8, 1)

# Pesos por coluna (eixo horizontal)
weights_col = np.linspace(1.0, 0.8, 8)  # (8,)
weights_col = weights_col[np.newaxis, np.newaxis, :]  # (1, 1, 8)
```

Broadcasting: Casos Complexos (cont.)

Python

```python
# Aplicar ambos os pesos (broadcasting automatico!)
weighted = tensor * weights_row * weights_col
print(f"Shape final: {weighted.shape}")  # (10, 8, 8)

# Verificar que pesos foram aplicados
print(f"Media original: {tensor.mean():.4f}")
print(f"Media ponderada: {weighted.mean():.4f}")
```

Broadcasting: Regras Visuais

Como determinar se broadcasting vai funcionar:

Regra 1: Alinhamento à direita

- Comparar shapes da direita para esquerda
- Dimensões faltando são tratadas como 1

Regra 2: Compatibilidade dimensional

- Duas dimensões compatíveis se: (a) iguais OU (b) uma delas é 1

Broadcasting: Regras Visuais

Exemplos de broadcasting VÁLIDO:

```
(10, 8, 8)  +  (8,)
(10, 8, 8)  +  (8, 8)
(10, 8, 8)  +  (10, 1, 1)
(10, 8, 8)  +  (1, 8, 1)
```

Exemplos de broadcasting INVÁLIDO:

```
(10, 8, 8)  +  (10, 8)✗
(10, 8, 8)  +  (9, 8, 8)✗
```

Fancy Indexing: Seleções Complexas

Selecionar elementos específicos com arrays de índices:

</> Python

```python
# Dataset MNIST
digits = load_digits()
X = digits.images  # (1797, 8, 8)

# Fancy indexing: selecionar imagens especificas
indices_imgs = [0, 10, 100, 500, 1000]
selected_images = X[indices_imgs]
print(f"Imagens selecionadas: {selected_images.shape}")  # (5, 8, 8)

# Fancy indexing multi-dimensional
# Selecionar pixels especificos de multiplas imagens
indices_imgs = [0, 1, 2]
indices_rows = [3, 4, 5]
indices_cols = [2, 3, 4]
```

Fancy Indexing: Seleções Complexas (cont.)

Python

```python
pixels = X[indices_imgs, indices_rows, indices_cols]
print(f"Pixels selecionados: {pixels.shape}")  # (3,)
print(f"Valores: {pixels}")

# Combinar com slicing normal
subgrid_from_selected = X[indices_imgs, 2:6, 2:6]
print(f"Subgrids: {subgrid_from_selected.shape}")  # (3, 4, 4)
```

Fancy Indexing: Arrays Multidimensionais

Python

```python
X = digits.images  # (1797, 8, 8)

# Criar grid de indices
row_indices = np.array([[0, 1], [2, 3]])  # (2, 2)
col_indices = np.array([[0, 1], [2, 3]])  # (2, 2)

# Fancy indexing com arrays 2D
# Para primeira imagem
img_0 = X[0]  # (8, 8)
selected_grid = img_0[row_indices, col_indices]
print(f"Grid selecionado: {selected_grid.shape}")  # (2, 2)
print(selected_grid)
```

Fancy Indexing: Arrays Multidimensionais (cont.)

Python

```python
# Aplicar em multiplas imagens
# Selecionar mesma regiao de 10 imagens
imgs_subset = X[:10]  # (10, 8, 8)

# Extrair centro de cada imagem
center_rows = slice(3, 5)
center_cols = slice(3, 5)
centers = imgs_subset[:, center_rows, center_cols]
print(f"\nCentros extraidos: {centers.shape}")  # (10, 2, 2)
```

Boolean Indexing Avançado

Combinar condições complexas para seleção:

Python

```python
X = digits.images  # (1797, 8, 8)

# Condicao 1: Imagens com media alta
mean_per_img = X.mean(axis=(1, 2))  # (1797,)
bright_mask = mean_per_img > 10

# Condicao 2: Imagens com alta variabilidade
std_per_img = X.std(axis=(1, 2))
variable_mask = std_per_img > 5
```

Boolean Indexing Avançado (cont.)

Python

```python
# Combinar condicoes: AND logico
bright_and_variable = bright_mask & variable_mask
selected = X[bright_and_variable]
print(f"Imagens claras E variaveis: {selected.shape[0]}")

# Combinar condicoes: OR logico
bright_or_variable = bright_mask | variable_mask
selected_or = X[bright_or_variable]
print(f"Imagens claras OU variaveis: {selected_or.shape[0]}")

# Condicao inversa: NOT
dark = ~bright_mask
dark_images = X[dark]
print(f"Imagens escuras: {dark_images.shape[0]}")
```

Boolean Indexing: Aplicar em Subsets

Python

```python
X = digits.images
y = digits.target

# Boolean indexing em labels para filtrar dataset
# Exemplo: Apenas digitos pares
even_digits = np.isin(y, [0, 2, 4, 6, 8])
X_even = X[even_digits]
y_even = y[even_digits]
print(f"Dataset apenas pares: {X_even.shape}")
```

Boolean Indexing: Aplicar em Subsets (cont.)

Python

```python
# Boolean indexing espacial (por pixel)
# Criar mascara espacial (circular no centro)
rows, cols = np.ogrid[:8, :8]
center_row, center_col = 4, 4
radius = 3
mask_spatial = (rows - center_row)**2 + (cols - center_col)**2 <= radius**2

# Aplicar mascara espacial em todas as imagens
X_masked = X.copy()
X_masked[:, ~mask_spatial] = 0  # Zerar fora do circulo
print(f"Mascaras aplicadas: {X_masked.shape}")
```

Boolean Indexing: Aplicar em Subsets (cont.)

Python

```python
# Media apenas da regiao mascarada
mean_in_mask = X[:, mask_spatial].mean()
print(f"Media na regiao central: {mean_in_mask:.2f}")
```

Operações em Batch: Comparação de Performance

Demonstrar speedup de vetorização:

Python

```python
import time
X = digits.images  # (1797, 8, 8)

# ABORDAGEM 1: Loop Python (LENTO)
start = time.time()
X_loop = np.zeros_like(X, dtype=float)
for i in range(len(X)):
    X_loop[i] = (X[i] - X[i].mean()) / X[i].std()
time_loop = time.time() - start
```

Operações em Batch: Comparação de Performance (cont.)

Python

```python
# ABORDAGEM 2: Vetorizada com broadcasting (RAPIDO)
start = time.time()
means = X.mean(axis=(1,2), keepdims=True)
stds = X.std(axis=(1,2), keepdims=True)
X_vectorized = (X - means) / stds
time_vectorized = time.time() - start
```

Operações em Batch: Comparação de Performance (cont.)

Python

```python
print(f"Tempo com loop: {time_loop*1000:.2f} ms")
print(f"Tempo vetorizado: {time_vectorized*1000:.2f} ms")
print(f"Speedup: {time_loop/time_vectorized:.1f}x mais rapido!")

# Verificar equivalencia
print(f"Resultados identicos: {np.allclose(X_loop, X_vectorized)}")

# Tempo com loop: 499.84 ms
# Tempo vetorizado: 22.09 ms
# Speedup: 22.6x mais rapido!
# Resultados identicos: True
```

Batch Processing: Funções Customizadas

Criar funções que operam em batches inteiros:

</> Python

```python
def process_batch(batch, operation='normalize'):
    """
    Processar batch de imagens de uma vez

    Parameters:
    -----------
    batch : ndarray, shape (n, h, w)
    operation : str, tipo de processamento

    Returns:
    --------
    processed : ndarray, shape (n, h, w)
    """
```

Batch Processing: Funções Customizadas (cont.)

</> Python

```python
    if operation == 'normalize':
        # Normalizar cada imagem
        min_vals = batch.min(axis=(1,2), keepdims=True)
        max_vals = batch.max(axis=(1,2), keepdims=True)
        return (batch - min_vals) / (max_vals - min_vals + 1e-10)
    elif operation == 'standardize':
        # Padronizar cada imagem
        means = batch.mean(axis=(1,2), keepdims=True)
        stds = batch.std(axis=(1,2), keepdims=True)
        return (batch - means) / (stds + 1e-10)
    elif operation == 'binarize':
        # Binarizar com threshold adaptativo (media de cada imagem)
        thresholds = batch.mean(axis=(1,2), keepdims=True)
        return (batch > thresholds).astype(float)
    return batch
```

Batch Processing: Usar Função Customizada

Python

```python
X = digits.images[:100]  # Primeiras 100 imagens

# Aplicar diferentes operacoes
X_normalized = process_batch(X, operation='normalize')
X_standardized = process_batch(X, operation='standardize')
X_binary = process_batch(X, operation='binarize')

# Comparar resultados
print("Normalizacao:")
print(f"  Min: {X_normalized.min():.2f}, Max: {X_normalized.max():.2f}")
print(f"  Mean: {X_normalized.mean():.4f}")

print("\nPadronizacao:")
print(f"  Mean: {X_standardized.mean():.2e}")
print(f"  Std: {X_standardized.std():.4f}")
```

Batch Processing: Usar Função Customizada (cont.)

Python

```python
print("\nBinarizacao:")
print(f"  Valores unicos: {np.unique(X_binary)}")
print(f"  Pct pixels brancos: {100*X_binary.mean():.1f}%")

# Visualizar exemplos
# [codigo de visualizacao]
```

Nota Importante

Funções vetorizadas = código limpo, rápido e reutilizável!

Batch Processing: Operações Estatísticas

Python

```python
X = digits.images  # (1797, 8, 8)

# Estatisticas por imagem (batch processing automatico!)
means = X.mean(axis=(1, 2))      # (1797,)
stds = X.std(axis=(1, 2))        # (1797,)
mins = X.min(axis=(1, 2))        # (1797,)
maxs = X.max(axis=(1, 2))        # (1797,)
ranges = maxs - mins             # (1797,)
```

Batch Processing: Operações Estatísticas (cont.)

Python

```python
# Criar DataFrame de estatisticas (integrar com Pandas!)
import pandas as pd
stats_df = pd.DataFrame({
    'mean': means,
    'std': stds,
    'min': mins,
    'max': maxs,
    'range': ranges,
    'label': y
})

print(stats_df.head(10))
print(f"\nEstatisticas agregadas por digito:")
print(stats_df.groupby('label')['mean'].agg(['mean', 'std']))  #...
```

Operações em Múltiplos Eixos

Python

```python
# Criar tensor 3D
tensor = np.random.randn(4, 5, 6)
print(f"Shape original: {tensor.shape}")  # (4, 5, 6)

# Operar em multiplos eixos ao mesmo tempo

# 1. Soma em dois eixos (eixos 0 e 1)
sum_multi = tensor.sum(axis=(0, 1))
print(f"\nsum(axis=(0, 1)) shape: {sum_multi.shape}")  # (6,)
# Remove dimensoes 0 e 1, resta apenas dimensao 2

# 2. Media em dois eixos
mean_multi = tensor.mean(axis=(1, 2))
print(f"mean(axis=(1, 2)) shape: {mean_multi.shape}")  # (4,)
# Remove dimensoes 1 e 2, resta apenas dimensao 0
```

Operações em Múltiplos Eixos (cont.)

Python

```python
# 3. Max em todos os eixos exceto um
# keepdims=True mantem as dimensoes
max_keepdims = tensor.max(axis=(0, 2), keepdims=True)
print(f"max(axis=(0,2), keepdims=True): {max_keepdims.shape}")  # (1, 5, 1)

# 4. Flatten vs axis
flatten_all = tensor.sum()  # Soma tudo
print(f"\nsum() total: {flatten_all:.2f}")
```

Reshape em Tensores

Mudar a forma sem mudar os dados

Regra fundamental:

- Produto das dimensões deve ser igual
- Exemplo: (2, 3, 4) = 24 elementos
- Pode reshapear para: (4, 6), (8, 3), (2, 12), (24,), etc.

Casos de uso comuns:

- **Flatten:** (A, B, C) $\rightarrow$ (A, B*C) ou (A*B*C,)
- **Adicionar batch:** (H, W, C) $\rightarrow$ (1, H, W, C)
- **Separar dimensões:** (100, 8, 8) $\rightarrow$ (100, 64)
- **Reorganizar:** (N, H*W) $\rightarrow$ (N, H, W)

Dica importante:

- Use -1 para inferir uma dimensão automaticamente
- Exemplo: reshape(10, -1) $\rightarrow$ NumPy calcula a segunda dimensão

Reshape: Exemplos Práticos

Python

```python
# Tensor 3D original
tensor = np.arange(24).reshape(2, 3, 4)
print(f"Original shape: {tensor.shape}")  # (2, 3, 4)

# 1. Flatten completo
flat = tensor.reshape(-1)
print(f"\nFlatten: {flat.shape}")  # (24,)

# 2. Matriz 2D
mat = tensor.reshape(6, 4)
print(f"Reshape para 2D: {mat.shape}")  # (6, 4)

# 3. Outra configuracao 2D
mat2 = tensor.reshape(2, -1)  # -1 infere automaticamente
print(f"Reshape(2, -1): {mat2.shape}")  # (2, 12)
```

Reshape: Exemplos Práticos (cont.)

Python

```python
# 4. Tensor 4D (adicionar dimensao batch)
tensor_4d = tensor.reshape(1, 2, 3, 4)
print(f"Adicionar batch: {tensor_4d.shape}")  # (1, 2, 3, 4)

# 5. Reorganizar dimensoes
reorg = tensor.reshape(4, 2, 3)
print(f"Reorganizado: {reorg.shape}")  # (4, 2, 3)

# IMPORTANTE: reshape nao muda os dados, apenas a forma!
print(f"\nPrimeiro elemento original: {tensor.flat[0]}")
print(f"Primeiro elemento reorganizado: {reorg.flat[0]}")
```

Transpose em Tensores 3D+

Python

```python
tensor = np.random.rand(2, 3, 4)
print(f"Original shape: {tensor.shape}")  # (2, 3, 4)

# Transpose sem argumentos: inverte todas as dimensoes
transposed = tensor.transpose()
print(f"\ntranspose(): {transposed.shape}")  # (4, 3, 2)
# Equivalente a transpose(2, 1, 0)

# Transpose com ordem especifica
# Ordem: (nova_dim_0, nova_dim_1, nova_dim_2)

# Trocar primeira e ultima dimensao
t1 = tensor.transpose(2, 1, 0)
print(f"transpose(2,1,0): {t1.shape}")  # (4, 3, 2)
```

Transpose em Tensores 3D+ (cont.)

Python

```python
# Trocar primeira e segunda dimensao
t2 = tensor.transpose(1, 0, 2)
print(f"transpose(1,0,2): {t2.shape}")  # (3, 2, 4)

# Mover ultima dimensao para primeira
t3 = tensor.transpose(2, 0, 1)
print(f"transpose(2,0,1): {t3.shape}")  # (4, 2, 3)

# Exemplo pratico: imagem (H, W, C) -> (C, H, W)
img = np.random.rand(256, 256, 3)
img_channels_first = img.transpose(2, 0, 1)
print(f"\nImagem (H,W,C): {img.shape}") #(256, 256, 3)
print(f"Imagem (C,H,W): {img_channels_first.shape}")  # (3, 256, 256)
```

Stack e Concatenate em 3D

Python

```python
# Criar dois tensores 2D
tensor_a = np.ones((3, 4))
tensor_b = np.zeros((3, 4))

print(f"Shape de cada tensor: {tensor_a.shape}")  # (3, 4)

# 1. STACK: cria nova dimensao
stacked_0 = np.stack([tensor_a, tensor_b], axis=0)
print(f"\nstack(axis=0): {stacked_0.shape}")  # (2, 3, 4)
# Nova dimensao na posicao 0

stacked_1 = np.stack([tensor_a, tensor_b], axis=1)
print(f"stack(axis=1): {stacked_1.shape}")  # (3, 2, 4)
# Nova dimensao na posicao 1
```

Stack e Concatenate em 3D (cont.)

Python

```python
stacked_2 = np.stack([tensor_a, tensor_b], axis=2)
print(f"stack(axis=2): {stacked_2.shape}")  # (3, 4, 2)
# Nova dimensao na posicao 2

# 2. CONCATENATE: junta ao longo de dimensao existente
concat_0 = np.concatenate([tensor_a, tensor_b], axis=0)
print(f"\nconcatenate(axis=0): {concat_0.shape}")  # (6, 4)
# Junta linhas

concat_1 = np.concatenate([tensor_a, tensor_b], axis=1)
print(f"concatenate(axis=1): {concat_1.shape}")  # (3, 8)
# Junta colunas
```

Broadcasting em 3D

Python

```python
# Tensor 3D
tensor = np.random.rand(4, 5, 6)
print(f"Tensor shape: {tensor.shape}")  # (4, 5, 6)

# 1. Broadcast escalar (sempre funciona)
result_1 = tensor + 10
print(f"\ntensor + escalar: {result_1.shape}")  # (4, 5, 6)

# 2. Broadcast vetor 1D
# Vetor com shape (6,) -> broadcasts em dimensoes 0 e 1
vec = np.array([1, 2, 3, 4, 5, 6])
result_2 = tensor + vec
print(f"tensor + vec(6,): {result_2.shape}")  # (4, 5, 6)
```

Broadcasting em 3D (cont.)

</> Python

```python
# 3. Broadcast array 2D
# Array (5, 6) -> broadcasts na dimensao 0
arr_2d = np.ones((5, 6))
result_3 = tensor + arr_2d
print(f"tensor + arr(5,6): {result_3.shape}")  # (4, 5, 6)

# 4. Broadcast com newaxis
# Adicionar dimensoes para broadcasting
vec_col = vec[np.newaxis, np.newaxis, :]  # Shape: (1, 1, 6) - preciso?
result_4 = tensor + vec_col
print(f"tensor + vec(6,1,1): {result_4.shape}")  # (4, 5, 6)
```

Broadcasting em 3D (cont.)

Python

```python
# 5. Exemplo pratico: normalizar por canal
# Subtrair media de cada "camada"
mean_per_layer = tensor.mean(axis=(1, 2), keepdims=True)  # (4, 1, 1)
normalized = tensor - mean_per_layer
print(f"\nNormalizado shape: {normalized.shape}")  # (4, 5, 6)
```

Broadcasting: Regras Simplificadas

Como NumPy faz broadcasting entre arrays?

Regra 1: Alinhamento à direita

- Shapes são comparados da direita para esquerda
- Dimensões faltando são tratadas como 1

Regra 2: Compatibilidade

- Duas dimensões são compatíveis se:
- (a) São iguais, OU
- (b) Uma delas é 1

Broadcasting: Regras Simplificadas

Como NumPy faz broadcasting entre arrays?

Exemplos de broadcasting válido:

- (4, 5, 6) + (6,) $\rightarrow$ (4, 5, 6) [vetor broadcast em dim 0 e 1]
- (4, 5, 6) + (1, 5, 6) $\rightarrow$ (4, 5, 6)
- (4, 5, 6) + (4, 1, 1) $\rightarrow$ (4, 5, 6)
- (4, 5, 6) + (5, 1) $\rightarrow$ (4, 5, 6)

Exemplos de broadcasting INVÁLIDO:

- (4, 5, 6) + (5, 6, 4) $\rightarrow$ ✗ Shapes incompatíveis
- (4, 5, 6) + (4, 6) $\rightarrow$ ✗ Segunda dim. não compatível

Resumo do Bloco 1

Conceitos fundamentais:

- Tensores = arrays N-dimensionais
- Rank = número de dimensões
- Shape = tupla com tamanho de cada dimensão

Operações essenciais:

- Indexação e slicing em 3D+
- Parâmetro `axis` para agregações
- Reshape para mudar forma
- Transpose para reordenar dimensões
- Stack/concatenate para combinar tensores

Broadcasting:

- Permite operações entre tensores de shapes diferentes
- Segue regras específicas de compatibilidade
- Essencial para código vetorizado eficiente

Transição para Bloco 2

Aprendemos a teoria de tensores...

- Criar e manipular arrays 3D+
- Usar axis para operações
- Aplicar transformações

Próximo: Aplicação prática com imagens!

- MNIST como tensor 3D real
- Análise exploratória visual
- Manipulação de pixels
- Filtros e transformações
- Operações em batch

Tudo que aprendemos sobre tensores será aplicado em imagens!

Bloco 2

Imagens como Dados Multidimensionais

Imagens são Tensores Naturais

Por que imagens são perfeitas para estudar tensores?

1. Estrutura multi-dimensional natural:
- Pixels organizados em altura × largura
- Múltiplos canais de cor (RGB)
- Coleções de imagens formam batches 4D

2. Fácil de visualizar:
- Podemos VER os dados
- Transformações são intuitivas
- Resultados são imediatamente compreensíveis

3. Aplicações práticas ubíquas:
- Medicina: raios-X, ressonâncias
- Satélites: imagens geográficas
- Segurança: câmeras, reconhecimento
- Arte: processamento de fotos

Estrutura de Imagens: Grayscale vs RGB

Imagem Grayscale (2D):

- Shape: (altura, largura)
- Cada pixel: um número (intensidade)
- Valores: 0 (preto) até 255 (branco)
- Exemplo: (256, 256) = imagem 256×256 pixels

Imagem RGB (3D):

- Shape: (altura, largura, 3 canais)
- Cada pixel: três números (R, G, B)
- Canais: Red, Green, Blue
- Exemplo: (256, 256, 3) = imagem colorida 256×256

MNIST é grayscale:

- Uma imagem: (8, 8) = 64 pixels
- Múltiplas imagens: (1797, 8, 8) = tensor 3D
- Valores: 0-16 (escala reduzida)

Batch de Imagens: Tensor 4D

Quando trabalhamos com múltiplas imagens:

Grayscale em batch (3D):

- Shape: (n_imagens, altura, largura)
- Exemplo: (1797, 8, 8) □ MNIST!
- Primeira dimensão = índice da imagem

RGB em batch (4D):

- Shape: (n_imagens, altura, largura, 3)
- Exemplo: (1000, 256, 256, 3)
- Quatro dimensões: batch, altura, largura, canais

Nota Importante

MNIST como (1797, 8, 8) é um exemplo perfeito de tensor 3D real!

Carregar MNIST Dataset

Python

```python
from sklearn.datasets import load_digits
import numpy as np
import matplotlib.pyplot as plt

# Carregar dataset MNIST simplificado
digits = load_digits()

# Componentes do dataset
X = digits.images  # Imagens como arrays 3D
y = digits.target  # Labels (digitos 0-9)
X_flat = digits.data  # Imagens achatadas (para ref.)
```

Carregar MNIST Dataset (cont.)

Python

```python
print("=== ESTRUTURA DO MNIST ===")
print(f"Imagens (X):         shape = {X.shape}")       # (1797, 8, 8)
print(f"Labels (y):          shape = {y.shape}")       # (1797,)
print(f"Achatado (X_flat):   shape = {X_flat.shape}") # (1797, 64)

print(f"\nTotal de imagens: {X.shape[0]}")
print(f"Tamanho de cada imagem: {X.shape[1]}x{X.shape[2]} pixels")
print(f"Total de pixels por imagem: {X.shape[1] * X.shape[2]}")

print(f"\nRange de valores: [{X.min():.1f}, {X.max():.1f}]")
print(f"Tipo de dados: {X.dtype}")

```

Explorar Estrutura do Dataset

Python

```python
# Informacoes detalhadas sobre as imagens

print("=== DIMENSOES ===")
print(f"Rank (ndim): {X.ndim}")           # 3
print(f"Shape: {X.shape}")                # (1797, 8, 8)
print(f"Total elementos: {X.size}")       # 115,008

print("\n=== DISTRIBUICAO DE LABELS ===")
unique, counts = np.unique(y, return_counts=True)
for digit, count in zip(unique, counts):
    print(f"Digito {digit}: {count} imagens")

# Verificar balanceamento
print(f"\nDataset balanceado? {counts.max() - counts.min() <= 10}")
```

Explorar Estrutura do Dataset (cont.)

Python

```python
print("\n=== PRIMEIRA IMAGEM ===")
print(f"Shape: {X[0].shape}")     # (8, 8)
print(f"Label: {y[0]}")           # Ex: 0
print(f"Valores min/max: [{X[0].min()}, {X[0].max()}]")

# Visualizar matriz de pixels da primeira imagem
print("\nPrimeiras 3 linhas da imagem 0:")
print(X[0][:3, :])
```

Visualizar Dígitos Individuais

Python

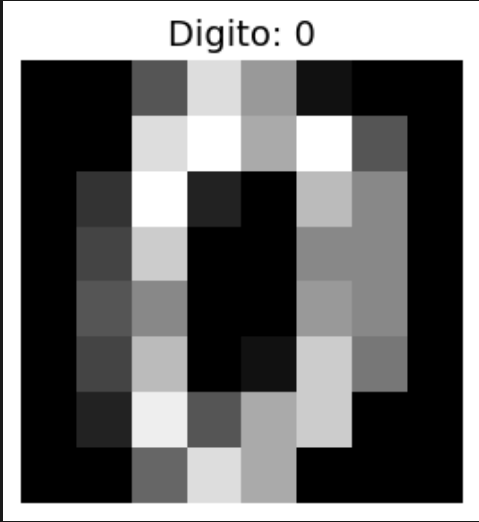
```python
# Visualizar uma unica imagem

def plot_digit(image, label, ax=None):
    """Plota um digito individual"""
    if ax is None:
        fig, ax = plt.subplots(figsize=(3, 3))

    ax.imshow(image, cmap='gray')
    ax.set_title(f'Digito: {label}', fontsize=14)
    ax.axis('off')

    return ax
```

Visualizar Dígitos Individuais (cont.)

Python

```python
# Plotar primeira imagem
fig, ax = plt.subplots(figsize=(3, 3))
plot_digit(X[0], y[0], ax)
plt.tight_layout()
plt.show()
```

Visualizar Dígitos Individuais (cont.)

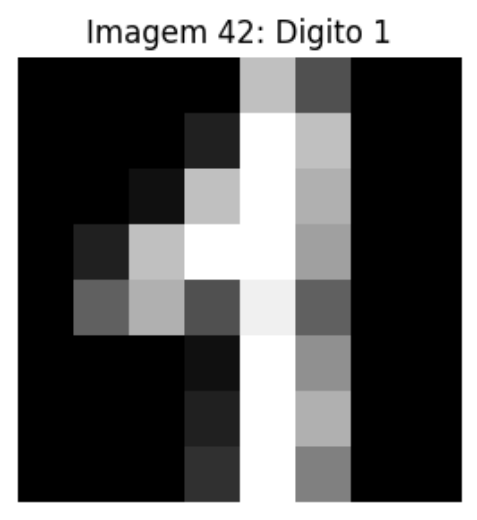


Visualizar Dígitos Individuais (cont.)

Python

```python
# Plotar imagem especifica
idx = 42
fig, ax = plt.subplots(figsize=(3, 3))
plot_digit(X[idx], y[idx], ax)
plt.title(f'Imagem {idx}: Digito {y[idx]}')
plt.tight_layout()
plt.show()
```

Visualizar Dígitos Individuais (cont.)



Visualizar Múltiplos Dígitos

Python

```python
# Grid de multiplas imagens

def plot_digits_grid(images, labels, n_rows=2, n_cols=5):
    """Plota grid de digitos"""
    fig, axes = plt.subplots(n_rows, n_cols, figsize=(12, 5))
    axes = axes.ravel()

    for i, ax in enumerate(axes):
        if i < len(images):
            ax.imshow(images[i], cmap='gray')
            ax.set_title(f'{labels[i]}', fontsize=12)
        ax.axis('off')

    plt.tight_layout()
    return fig
```

Visualizar Múltiplos Dígitos (cont.)

Python

```python
# Plotar primeiras 10 imagens
fig = plot_digits_grid(X[:10], y[:10])
plt.suptitle('Primeiras 10 Imagens do MNIST', y=1.02, fontsize=14)
plt.show()

# Plotar imagens aleatorias
np.random.seed(42)
indices = np.random.choice(len(X), size=10, replace=False)
fig = plot_digits_grid(X[indices], y[indices])
plt.suptitle('10 Imagens Aleatorias', y=1.02, fontsize=14)
plt.show()

```

Análise: Estatísticas Globais

Python

```python
# Estatisticas de todo o dataset

print("=== ESTATISTICAS GLOBAIS ===")
print(f"Media geral: {X.mean():.2f}")
print(f"Desvio padrao: {X.std():.2f}")
print(f"Minimo: {X.min():.2f}")
print(f"Maximo: {X.max():.2f}")
print(f"Mediana: {np.median(X):.2f}")

# Percentis
p25, p50, p75 = np.percentile(X, [25, 50, 75])
print(f"\nPercentis:")
print(f"  25%: {p25:.2f}")
print(f"  50%: {p50:.2f}")
print(f"  75%: {p75:.2f}")
```

Análise: Estatísticas Globais (cont.)

</> Python

```python
# Distribuicao de valores
print(f"\n% de pixels pretos (0): {(X == 0).sum() / X.size * 100:.1f}%")
print(f"% de pixels brancos (16): {(X == 16).sum() / X.size * 100:.1f}%")

# Estatisticas por imagem
mean_per_image = X.mean(axis=(1, 2))  # Media de cada imagem
print(f"\nMedia por imagem:")
print(f"  Min: {mean_per_image.min():.2f}")
print(f"  Max: {mean_per_image.max():.2f}")
print(f"  Std: {mean_per_image.std():.2f}")
```

Análise: Estatísticas por Dígito

Python

```python
# Analise separada por digito (0-9)

print("=== ESTATISTICAS POR DIGITO ===\n")

for digit in range(10):
    # Selecionar todas as imagens deste digito
    mask = (y == digit)
    images_digit = X[mask]

    # Calcular estatisticas
    n_images = images_digit.shape[0]
    mean_intensity = images_digit.mean()
    std_intensity = images_digit.std()
```

Análise: Estatísticas por Dígito (cont.)

Python

```python
    print(f"Digito {digit}:")
    print(f"  N imagens: {n_images}")
    print(f"  Intensidade media: {mean_intensity:.2f}")
    print(f"  Desvio padrao: {std_intensity:.2f}")
    print()
# Qual digito tem maior intensidade media?
mean_per_digit = []
for digit in range(10):
    mean_per_digit.append(X[y == digit].mean())
max_digit = np.argmax(mean_per_digit)
min_digit = np.argmin(mean_per_digit)

print(f"Digito mais 'claro': {max_digit} (media: {mean_per_digit[max_digit]:.2f})")
print(f"Digito mais 'escuro': {min_digit} (media: {mean_per_digit[min_digit]:.2f})")
```

Visualizar Estatísticas por Dígito

Python

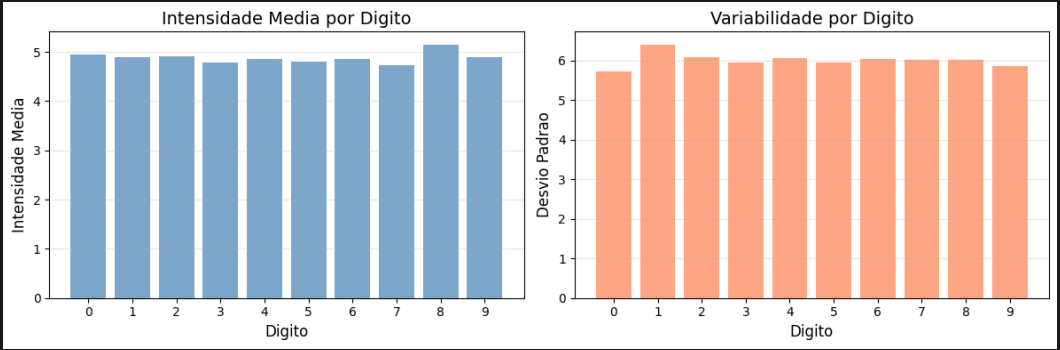
```python
# Plotar estatisticas por digito

# Calcular media e std por digito
digits_list = range(10)
means = [X[y == d].mean() for d in digits_list]
stds = [X[y == d].std() for d in digits_list]

# Plotar
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
```

Visualizar Estatísticas por Dígito (cont.)

Python

```python
# Grafico 1: Intensidade media
ax1.bar(digits_list, means, color='steelblue', alpha=0.7)
ax1.set_xlabel('Digito', fontsize=12)
ax1.set_ylabel('Intensidade Media', fontsize=12)
ax1.set_title('Intensidade Media por Digito', fontsize=14)
ax1.set_xticks(digits_list)
ax1.grid(axis='y', alpha=0.3)
# Grafico 2: Desvio padrao
ax2.bar(digits_list, stds, color='coral', alpha=0.7)
ax2.set_xlabel('Digito', fontsize=12)
ax2.set_ylabel('Desvio Padrao', fontsize=12)
ax2.set_title('Variabilidade por Digito', fontsize=14)
ax2.set_xticks(digits_list)
ax2.grid(axis='y', alpha=0.3)
plt.tight_layout()
plt.show()
```

Visualizar Estatísticas por Dígito (cont.)



Histogramas de Intensidade

Python

```python
# Distribuicao de intensidades de pixels

fig, axes = plt.subplots(2, 2, figsize=(12, 8))

# 1. Histograma global (todos os pixels)
axes[0, 0].hist(X.ravel(), bins=17, color='steelblue',
                alpha=0.7, edgecolor='black')
axes[0, 0].set_xlabel('Intensidade', fontsize=11)
axes[0, 0].set_ylabel('Frequencia', fontsize=11)
axes[0, 0].set_title('Distribuicao Global de Pixels', fontsize=12)
axes[0, 0].grid(axis='y', alpha=0.3)
```

Histogramas de Intensidade (cont.)

Python

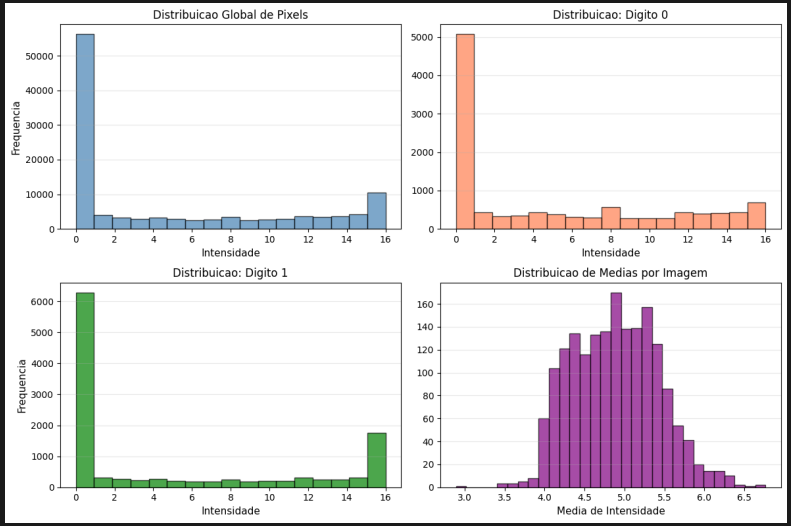
```python
# 2. Histograma para digito 0
axes[0, 1].hist(X[y == 0].ravel(), bins=17, color='coral',
                alpha=0.7, edgecolor='black')
axes[0, 1].set_xlabel('Intensidade', fontsize=11)
axes[0, 1].set_title('Distribuicao: Digito 0', fontsize=12)
axes[0, 1].grid(axis='y', alpha=0.3)

# 3. Histograma para digito 1
axes[1, 0].hist(X[y == 1].ravel(), bins=17, color='green',
                alpha=0.7, edgecolor='black')
axes[1, 0].set_xlabel('Intensidade', fontsize=11)
axes[1, 0].set_ylabel('Frequencia', fontsize=11)
axes[1, 0].set_title('Distribuicao: Digito 1', fontsize=12)
axes[1, 0].grid(axis='y', alpha=0.3)
```

Histogramas de Intensidade (cont.)

Python

```python
# 4. Comparacao de medias por imagem
mean_per_img = X.mean(axis=(1, 2))
axes[1, 1].hist(mean_per_img, bins=30, color='purple',
                alpha=0.7, edgecolor='black')
axes[1, 1].set_xlabel('Media de Intensidade', fontsize=11)
axes[1, 1].set_title('Distribuicao de Medias por Imagem', fontsize=12)
axes[1, 1].grid(axis='y', alpha=0.3)

plt.tight_layout()
plt.show()
```

Histogramas de Intensidade (cont.)



Calcular Média por Dígito

Python

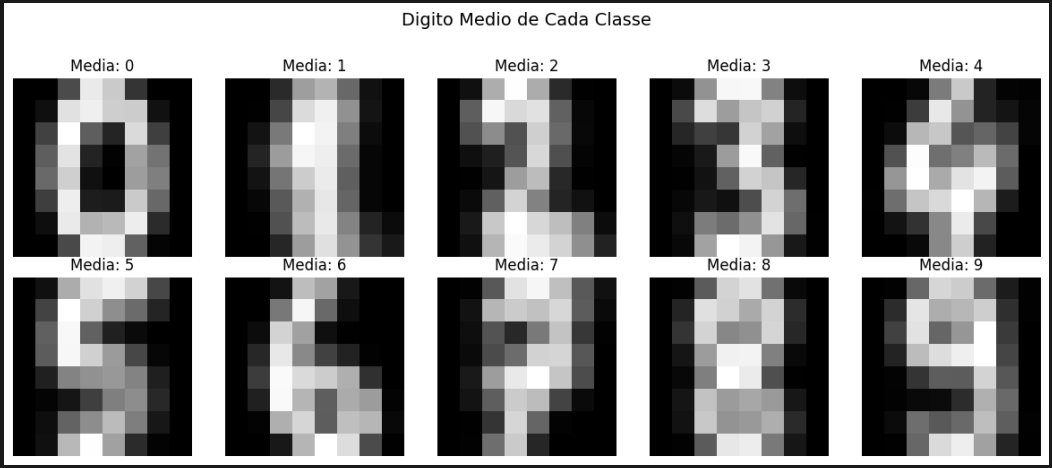
```python
# Criar "digito medio" para cada classe

# Array para armazenar medias
mean_images = np.zeros((10, 8, 8))

for digit in range(10):
    # Selecionar todas as imagens deste digito
    images_digit = X[y == digit]

    # Calcular media ao longo do eixo 0 (das imagens)
    # Esta operação calcula a **média pixel a pixel** através de todas as imagens de um dígito específico.
    mean_images[digit] = images_digit.mean(axis=0)

print(f"Shape das imagens medias: {mean_images.shape}")  # (10, 8, 8)
```

Calcular Média por Dígito (cont.)

Python

```python
# Visualizar digitos medios
fig, axes = plt.subplots(2, 5, figsize=(12, 5))
axes = axes.ravel()

for digit in range(10):
    axes[digit].imshow(mean_images[digit], cmap='gray')
    axes[digit].set_title(f'Media: {digit}', fontsize=12)
    axes[digit].axis('off')

plt.suptitle('Digito Medio de Cada Classe', fontsize=14, y=1.02)
plt.tight_layout()
plt.show()

# Insight: podemos ver o "digito tipico" de cada classe!

```

Calcular Média por Dígito (cont.)



Variância por Dígito

Python

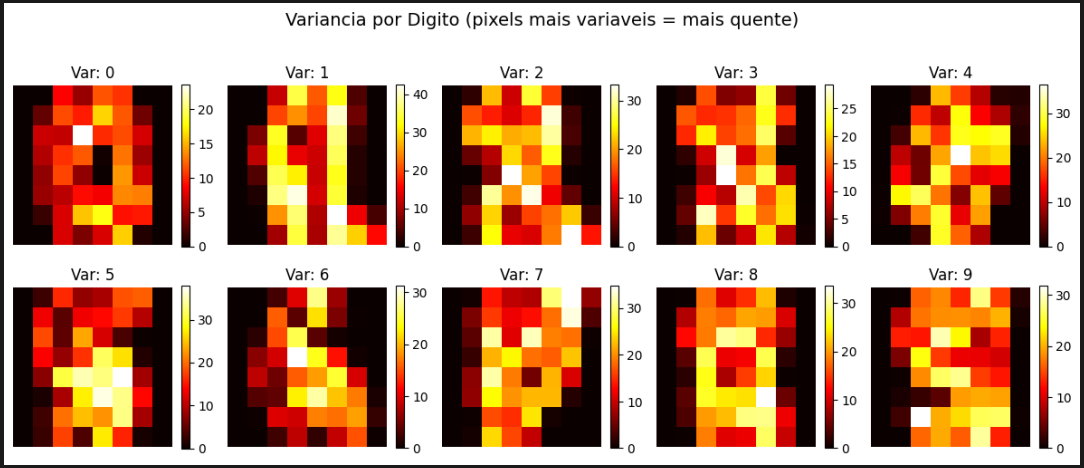
```python
# Calcular variancia por digito (variabilidade)

# Array para armazenar variancias
var_images = np.zeros((10, 8, 8))

for digit in range(10):
    images_digit = X[y == digit]
    # Variancia ao longo do eixo 0
    var_images[digit] = images_digit.var(axis=0)

# Visualizar variancias
fig, axes = plt.subplots(2, 5, figsize=(12, 5))
axes = axes.ravel()
```

Variância por Dígito (cont.)

Python

```python
for digit in range(10):
    im = axes[digit].imshow(var_images[digit], cmap='hot')
    axes[digit].set_title(f'Var: {digit}', fontsize=12)
    axes[digit].axis('off')
    plt.colorbar(im, ax=axes[digit], fraction=0.046)

plt.suptitle('Variancia por Digito (pixels mais variaveis = mais quente)',
             fontsize=14, y=1.02)
plt.tight_layout()
plt.show()

# Insight: areas de alta variancia mostram onde os digitos
# variam mais entre diferentes escritas
```

Variância por Dígito (cont.)



Dígitos Mais e Menos Variáveis

Python

```python
# Identificar digitos com maior/menor variabilidade

# Calcular variancia total por digito
total_var_per_digit = []
for digit in range(10):
    images_digit = X[y == digit]
    # Variancia total = soma das variancias de cada pixel
    total_var = images_digit.var(axis=0).sum()
    total_var_per_digit.append(total_var)

# Encontrar digitos mais/menos variaveis
most_var_digit = np.argmax(total_var_per_digit)
least_var_digit = np.argmin(total_var_per_digit)
```

Dígitos Mais e Menos Variáveis (cont.)

Python

```python
print("=== VARIABILIDADE POR DIGITO ===\n")
for digit in range(10):
    print(f"Digito {digit}: variancia total = {total_var_per_digit[digit
    ]:.2f}")

print(f"\nDigito MAIS variavel: {most_var_digit}") # 1
print(f"Digito MENOS variavel: {least_var_digit}") # 0

# Insight: digitos mais variaveis tem mais estilos de escrita diferentes
# Digitos menos variaveis sao escritos de forma mais consistente
```

Comparar Exemplos de um Dígito

Python

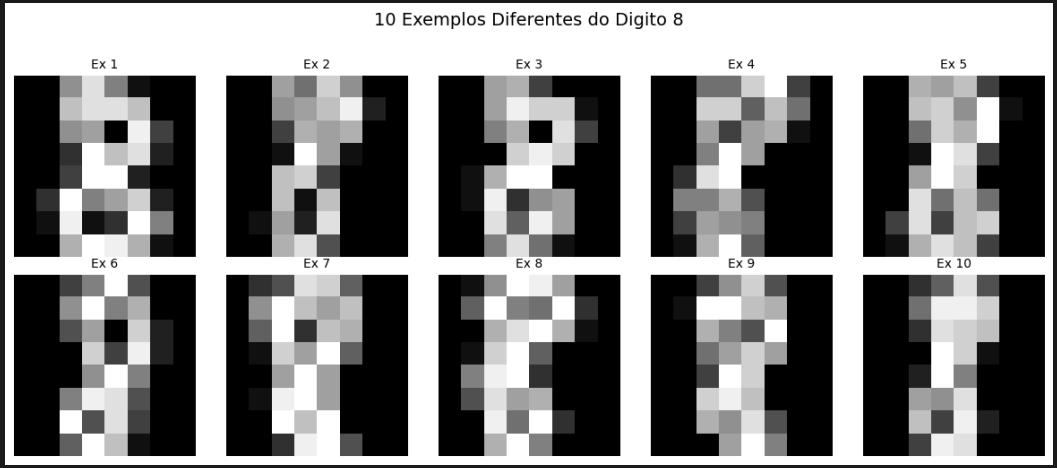
```python
# Visualizar multiplos exemplos do mesmo digito

digit_to_show = 8  # Escolher um digito

# Selecionar todas as imagens deste digito
indices = np.where(y == digit_to_show)[0]

# Pegar primeiras 10
indices_sample = indices[:10]

# Plotar
fig, axes = plt.subplots(2, 5, figsize=(12, 5))
axes = axes.ravel()
```

Comparar Exemplos de um Dígito (cont.)

Python

```python
for i, idx in enumerate(indices_sample):
    axes[i].imshow(X[idx], cmap='gray')
    axes[i].set_title(f'Ex {i+1}', fontsize=10)
    axes[i].axis('off')

plt.suptitle(f'10 Exemplos Diferentes do Digito {digit_to_show}',
             fontsize=14, y=1.02)
plt.tight_layout()
plt.show()

# Insight: mesmo digito pode ser escrito de formas muito diferentes!
```

Comparar Exemplos de um Dígito (cont.)



Identificar Padrões Visuais

Python

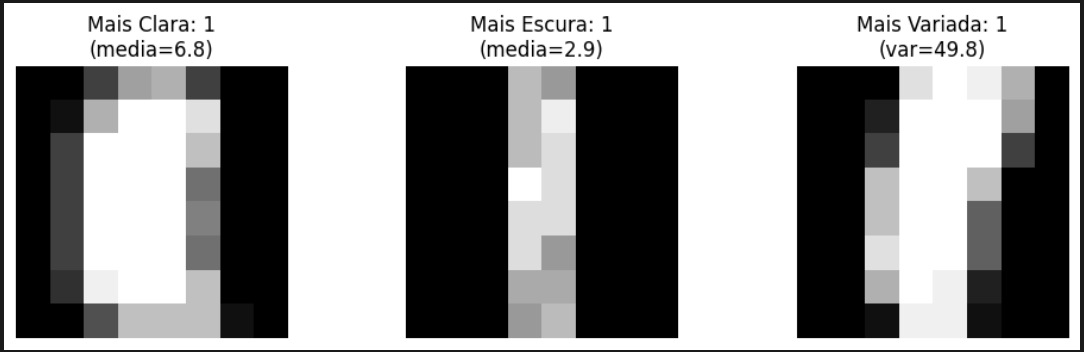
```python
# Analise visual: encontrar imagens "extremas"

# 1. Imagem mais clara (maior intensidade media)
mean_per_image = X.mean(axis=(1, 2))
brightest_idx = mean_per_image.argmax()

# 2. Imagem mais escura (menor intensidade media)
darkest_idx = mean_per_image.argmin()

# 3. Imagem com maior variancia (mais "espalhada")
var_per_image = X.var(axis=(1, 2))
most_varied_idx = var_per_image.argmax()
```

Identificar Padrões Visuais (cont.)

Python

```python
# Visualizar
fig, axes = plt.subplots(1, 3, figsize=(10, 3))

axes[0].imshow(X[brightest_idx], cmap='gray')
axes[0].set_title(f'Mais Clara: {y[brightest_idx]}\n(media={mean_per_image[brightest_idx]:.1f})')
axes[0].axis('off')

axes[1].imshow(X[darkest_idx], cmap='gray')
axes[1].set_title(f'Mais Escura: {y[darkest_idx]}\n(media={mean_per_image[darkest_idx]:.1f})')
axes[1].axis('off')

axes[2].imshow(X[most_varied_idx], cmap='gray')
axes[2].set_title(f'Mais Variada: {y[most_varied_idx]}\n(var={var_per_image[most_varied_idx]:.1f})')
```

Identificar Padrões Visuais (cont.)



Operações Básicas: Acessar Pixels

Python

```python
# Acessar e modificar pixels individuais

# Selecionar primeira imagem
img = X[0].copy()  # Copy para nao modificar original
print(f"Shape da imagem: {img.shape}")  # (8, 8)
print(f"Label: {y[0]}")

# Acessar pixel especifico (linha, coluna)
pixel_value = img[3, 4]
print(f"\nPixel [3, 4]: {pixel_value}")

# Acessar linha completa
row = img[2, :]
print(f"\nLinha 2: {row}")
print(f"Shape: {row.shape}")  # (8,)
```

Operações Básicas: Acessar Pixels (cont.)

Python

```python
# Acessar coluna completa
col = img[:, 5]
print(f"\nColuna 5: {col}")
print(f"Shape: {col.shape}")  # (8,)

# Acessar regiao (slice 2D)
region = img[2:5, 3:6]
print(f"\nRegiao [2:5, 3:6]:")
print(region)
print(f"Shape: {region.shape}")  # (3, 3)
```

Operações Básicas: Modificar Pixels

Python

```python
# Modificar valores de pixels

# Criar copia da primeira imagem
img_modified = X[0].copy()

# 1. Modificar pixel unico
img_modified[3, 4] = 16  # Colocar branco
print("Pixel [3, 4] modificado para 16")

# 2. Modificar linha inteira
img_modified[0, :] = 0  # Primeira linha preta
print("Primeira linha definida como 0")
```

Operações Básicas: Modificar Pixels (cont.)

Python

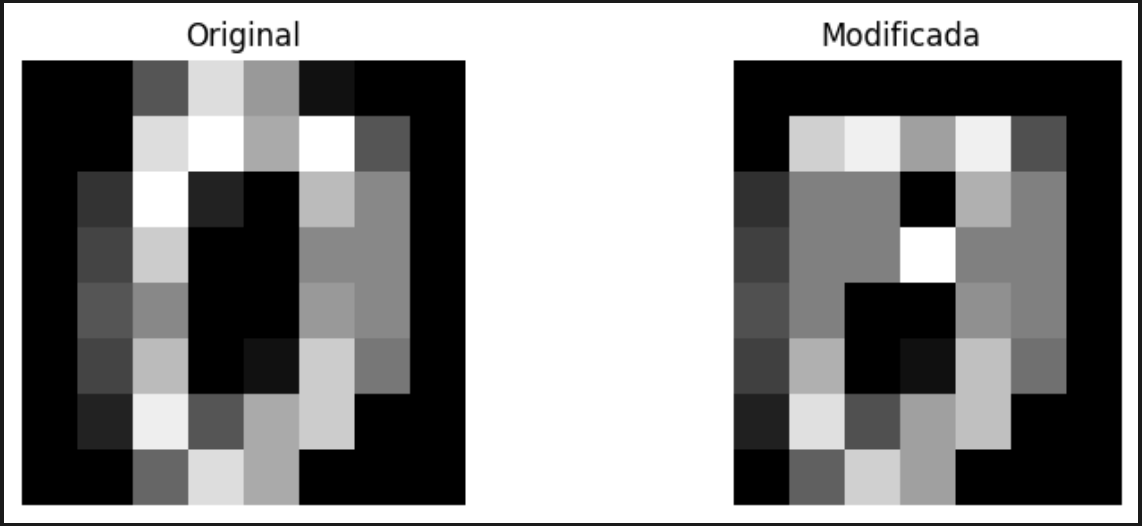
```python
# 3. Modificar coluna inteira
img_modified[:, 0] = 16  # Primeira coluna branca
print("Primeira coluna definida como 16")

# 4. Modificar regiao
img_modified[2:4, 2:4] = 8  # Quadrado cinza no centro
print("Regiao [2:4, 2:4] definida como 8")
```

Operações Básicas: Modificar Pixels (cont.)

Python

```python
# Visualizar comparacao
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 3))

ax1.imshow(X[0], cmap='gray')
ax1.set_title('Original')
ax1.axis('off')

ax2.imshow(img_modified, cmap='gray')
ax2.set_title('Modificada')
ax2.axis('off')

plt.tight_layout()
plt.show()
```

Operações Básicas: Modificar Pixels (cont.)



Operações Aritméticas em Imagens

Operações aritméticas aplicam-se a QUALQUER tensor

Operações básicas (broadcasting automático):

- **Soma/Subtração:** Ajustar nível (ex: offset)
- **Multiplicação/Divisão:** Escalar valores
- **Exponenciação:** Transformações não-lineares
- **Inversão:** Complemento de valores

Exemplo com imagens (generaliza para qualquer dado):

Operações Aritméticas em Imagens

Python

```python
# Operacoes matematicas em pixels

img = X[0].copy()

# 1. Adicionar constante (aumentar brilho)
img_brighter = img + 5
img_brighter = np.clip(img_brighter, 0, 16)  # Manter no range

# 2. Multiplicar por escalar (ajustar contraste)
img_contrast = img * 1.5
img_contrast = np.clip(img_contrast, 0, 16)

# 3. Inverter (negativo)
img_inverted = 16 - img
```

Operações Aritméticas em Imagens (cont.)

Python

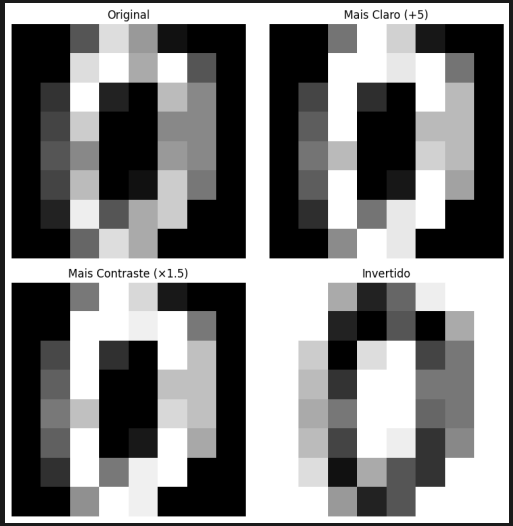
```python
# 4. Normalizar (0-1)
img_normalized = (img - img.min()) / (img.max() - img.min())

# Visualizar
fig, axes = plt.subplots(2, 2, figsize=(8, 8))

axes[0, 0].imshow(img, cmap='gray')
axes[0, 0].set_title('Original')

axes[0, 1].imshow(img_brighter, cmap='gray')
axes[0, 1].set_title('Mais Claro (+5)')

axes[1, 0].imshow(img_contrast, cmap='gray')
axes[1, 0].set_title('Mais Contraste (×1.5)')
```

Operações Aritméticas em Imagens (cont.)

Python

```python
axes[1, 1].imshow(img_inverted, cmap='gray')
axes[1, 1].set_title('Invertido')

for ax in axes.ravel():
    ax.axis('off')

plt.tight_layout()
plt.show()

# Estas operacoes funcionam IDENTICAMENTE em:
# - Series temporais (100, 5)
# - Dados de vendas (10, 5, 12)
# - Qualquer tensor numerico!
```

Operações Aritméticas em Imagens (cont.)



Máscaras Booleanas em Imagens

</> Python

```python
# Usar condicoes para selecionar pixels

img = X[0].copy()

# 1. Criar mascara: pixels escuros
mask_dark = img < 5
print(f"Pixels escuros (< 5): {mask_dark.sum()}")

# 2. Criar mascara: pixels claros
mask_bright = img > 10
print(f"Pixels claros (> 10): {mask_bright.sum()}")
```

Máscaras Booleanas em Imagens (cont.)

Python

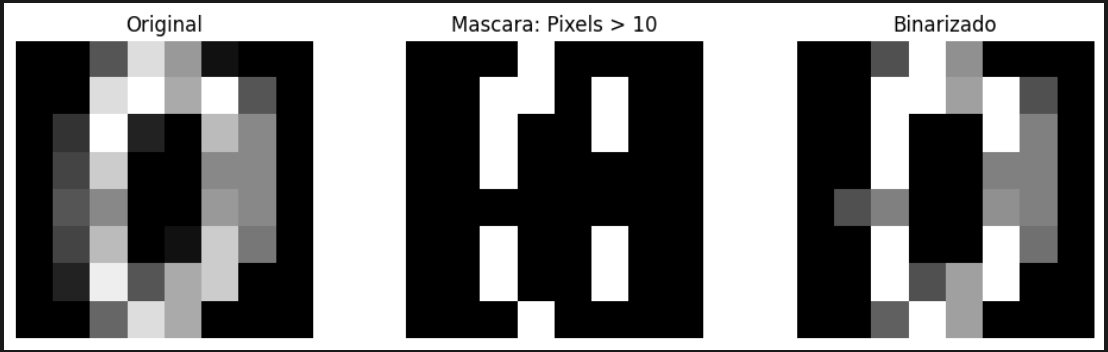
```python
# 3. Usar mascara para modificar
img_masked = img.copy()
img_masked[mask_dark] = 0      # Pixels escuros viram preto
img_masked[mask_bright] = 16   # Pixels claros viram branco

# 4. Extrair apenas pixels que atendem condicao
bright_pixels = img[mask_bright]
print(f"\nPixels claros extraidos: {bright_pixels}")
print(f"Media dos pixels claros: {bright_pixels.mean():.2f}")
```

Máscaras Booleanas em Imagens (cont.)

Python

```python
# Visualizar mascaras
fig, axes = plt.subplots(1, 3, figsize=(10, 3))
axes[0].imshow(img, cmap='gray')
axes[0].set_title('Original')

axes[1].imshow(mask_bright, cmap='gray')
axes[1].set_title('Mascara: Pixels > 10')

axes[2].imshow(img_masked, cmap='gray')
axes[2].set_title('Binarizado')

for ax in axes:
    ax.axis('off')
plt.tight_layout()
plt.show()
```

Máscaras Booleanas em Imagens (cont.)



Comparações entre Amostras de Tensor

Comparar elementos do tensor entre si

Métricas comuns:

- Diferença absoluta: $|a - b|$
- Diferença quadrática: $(a - b)^2$
- Distância euclidiana: $\sqrt{\sum (a - b)^2}$
- Correlação: covariância normalizada

Comparações entre Imagens

Python

```python
# Comparar duas imagens elemento-wise

# Selecionar dois digitos '8' diferentes
indices_8 = np.where(y == 8)[0]
img1 = X[indices_8[0]]
img2 = X[indices_8[5]]

# 1. Diferenca absoluta
diff = np.abs(img1 - img2)
print(f"Diferenca media: {diff.mean():.2f}")
print(f"Diferenca maxima: {diff.max():.2f}")
```

Comparações entre Imagens (cont.)

Python

```python
# 2. Similaridade (correlacao)
# Flatten para calcular correlacao
corr = np.corrcoef(img1.ravel(), img2.ravel())[0, 1]
print(f"Correlacao: {corr:.3f}")

# 3. Distancia euclidiana
dist = np.sqrt(((img1 - img2) ** 2).sum())
print(f"Distancia euclidiana: {dist:.2f}")

# Visualizar comparacao
fig, axes = plt.subplots(1, 4, figsize=(12, 3))
```

Comparações entre Imagens (cont.)

Python

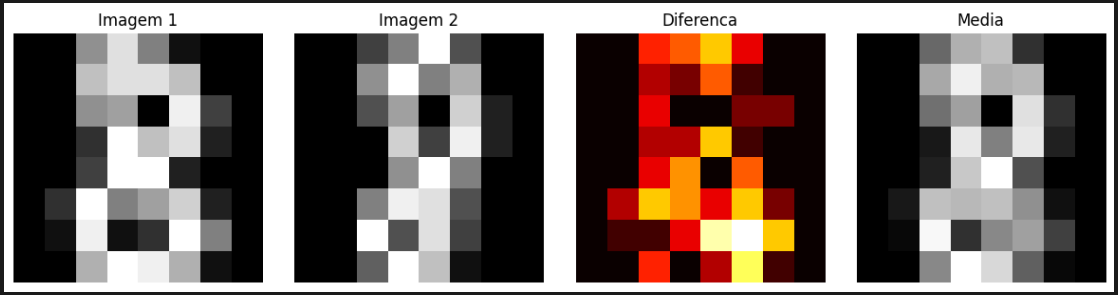
```python
axes[0].imshow(img1, cmap='gray')
axes[0].set_title('Imagem 1')
axes[1].imshow(img2, cmap='gray')
axes[1].set_title('Imagem 2')
axes[2].imshow(diff, cmap='hot')
axes[2].set_title('Diferenca')
axes[3].imshow((img1 + img2) / 2, cmap='gray')
axes[3].set_title('Media')
for ax in axes:
    ax.axis('off')
plt.tight_layout()
plt.show()
```

Comparações entre Imagens (cont.)

Python

```python
# Generalização: comparar quaisquer amostras de tensor
# Serie temporal: comparar padrões de dois dias
dia1 = temp_data[10]  # (5 sensores, 3 horários)
dia2 = temp_data[20]
diff_temp = np.abs(dia1 - dia2)
dist_temp = np.sqrt(((dia1 - dia2) ** 2).sum())
print(f"Distância entre dias: {dist_temp:.2f}")
# Vendas: comparar performance de dois produtos
prod1 = sales[0]  # (5 regiões, 12 meses)
prod2 = sales[1]
corr_prod = np.corrcoef(prod1.ravel(), prod2.ravel())[0, 1]
print(f"Correlação entre produtos: {corr_prod:.3f}")
# PRINCÍPIO: operações elemento-wise permitem comparações diretas
```

Máscaras Booleanas em Imagens (cont.)



Operações em Batch: Processar Múltiplas

Python

```python
# Aplicar operacao em todas as imagens de uma vez

# 1. Normalizar TODAS as imagens (vetorizado)
X_normalized = (X - X.min()) / (X.max() - X.min())
print(f"Range normalizado: [{X_normalized.min():.2f}, {X_normalized.max()
    :.2f}]")

# 2. Aumentar brilho de todas
X_brighter = X + 3
X_brighter = np.clip(X_brighter, 0, 16)
print(f"Brilho aumentado em todas: shape {X_brighter.shape}")
```

Operações em Batch: Processar Múltiplas (cont.)

Python

```python
# 3. Calcular media por imagem (axis=(1,2))
mean_per_image = X.mean(axis=(1, 2))
print(f"\nMedias por imagem: shape {mean_per_image.shape}")  # (1797,)
print(f"Primeiras 5 medias: {mean_per_image[:5]}")

# 4. Centralizar cada imagem (subtrair sua propria media)
# Broadcasting: (1797, 8, 8) - (1797, 1, 1)
X_centered = X - mean_per_image[:, np.newaxis, np.newaxis]
print(f"\nCentralizado: shape {X_centered.shape}")
print(f"Nova media global: {X_centered.mean():.10f}")  # ~0

# Verificar que media de cada imagem eh 0
print(f"Medias apos centralizacao: {X_centered.mean(axis=(1,2))[:5]}")
```

Comparações entre Amostras em Batch

Computar similaridades/distâncias entre múltiplas amostras:

Python

```python
X = digits.images  # (1797, 8, 8)

# Selecionar subset para comparação
X_subset = X[:10]  # Primeiras 10 imagens

# Flatten para calcular distancias
X_flat = X_subset.reshape(10, -1)  # (10, 64)

# Matriz de distancias euclidanas (broadcasting!)
# Distancia entre i e j: sqrt(sum((X[i] - X[j])^2))
```

Comparações entre Amostras em Batch (cont.)

Python

```python
# Abordagem vetorizada (sem loops!)
# Expandir dimensoes para broadcasting
X_expanded_i = X_flat[:, np.newaxis, :]    # (10, 1, 64)
X_expanded_j = X_flat[np.newaxis, :, :]    # (1, 10, 64)
# Calcular diferencas
diffs = X_expanded_i - X_expanded_j  # (10, 10, 64)
# Distancias euclidanas
distances = np.sqrt((diffs ** 2).sum(axis=2))  # (10, 10)
print(f"Matriz de distancias: {distances.shape}")
print(f"Distancia entre img 0 e img 1: {distances[0, 1]:.2f}") # 59.56
# Imagens mais similares (distância mínima, exceto diagonal)
np.fill_diagonal(distances, np.inf)
most_similar = np.unravel_index(distances.argmin(), distances.shape)
print(f"Imagens mais similares: {most_similar[0]} e {most_similar[1]}")
# Imagens mais similares: 5 e 9
```

Correlações em Batch

</> Python

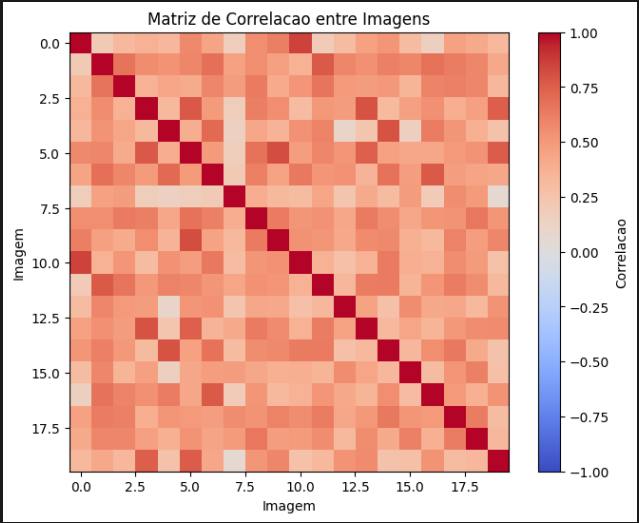
```python
X = digits.images[:20]  # 20 imagens

# Flatten
X_flat = X.reshape(20, -1)  # (20, 64)

# Matriz de correlacao (20 x 20)
# np.corrcoef calcula correlações entre linhas
corr_matrix = np.corrcoef(X_flat)

print(f"Matriz de correlacao: {corr_matrix.shape}")  # (20, 20)
print(f"Correlacao entre img 0 e img 1: {corr_matrix[0, 1]:.3f}")
```

Correlações em Batch (cont.)

Python

```python
# Encontrar pares com alta correlação
threshold = 0.8
high_corr_mask = (corr_matrix > threshold) & (corr_matrix < 1.0)
high_corr_pairs = np.argwhere(high_corr_mask)

print(f"\nPares com correlacao > {threshold}:")
for i, j in high_corr_pairs[:5]:  # Mostrar apenas primeiros 5
    if i < j:  # Evitar duplicatas
        print(f"  Imgs {i} e {j}: r={corr_matrix[i, j]:.3f}")

# Visualizar matriz de correlacao como heatmap
import matplotlib.pyplot as plt
plt.figure(figsize=(8, 6))
plt.imshow(corr_matrix, cmap='coolwarm', vmin=-1, vmax=1)
plt.colorbar(label='Correlacao')
plt.title('Matriz de Correlacao entre Imagens')
```

Correlações em Batch (cont.)



Resumo: Imagens como Tensores

Estrutura:

- Imagens grayscale: 2D (altura × largura)
- Imagens RGB: 3D (altura × largura × 3)
- Batch de imagens: 3D ou 4D
- MNIST: (1797, 8, 8) - tensor 3D real

Análise exploratória:

- Estatísticas globais e por dígito
- Imagem média e variância por classe
- Histogramas de intensidade
- Identificar padrões e outliers

Operações básicas:

- Acessar e modificar pixels
- Operações aritméticas
- Máscaras booleanas
- Processamento em batch (vetorizado)

Filtros: Binarização (Threshold)

Aplicar operação condicional em tensor (exemplo: binarização)

Conceito geral: Transformar valores baseado em condição

Aplicável a qualquer tensor:

- Imagens: pixels acima/abaixo threshold
- Séries temporais: valores acima/abaixo limite
- Vendas: performance acima/abaixo meta

Nota Importante

Operação condicional + casting = padrão comum em análise de dados!

Filtros: Binarização (Threshold)

Python

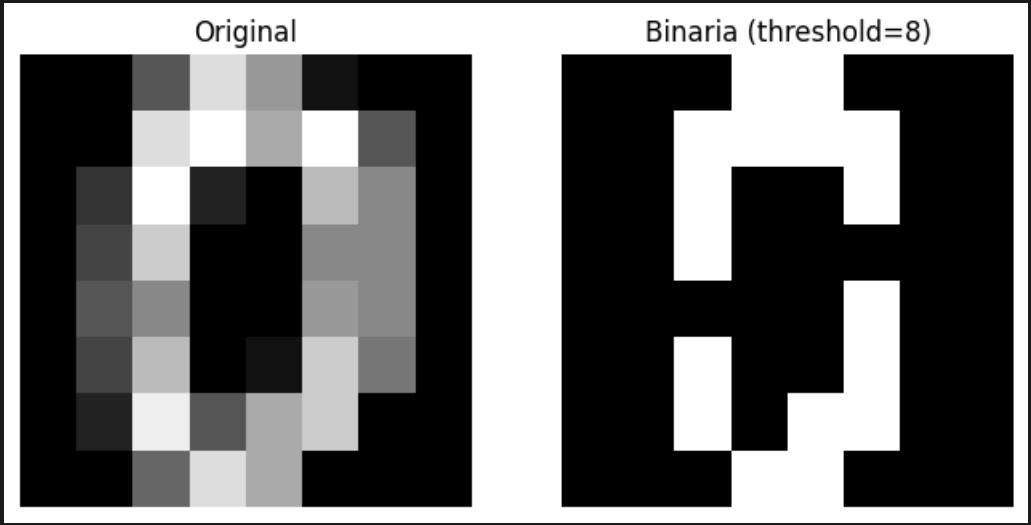
```python
# Binarizacao: pixels acima do threshold -> branco, abaixo -> preto

# Escolher uma imagem
img = X[0]  # Primeiro digito do dataset

# Definir threshold (limiar)
threshold = 8  # Valor medio da escala 0-16

# Aplicar threshold
img_binary = (img > threshold).astype(int) * 16
# True -> 1 -> 16 (branco), False -> 0 -> 0 (preto)
```

Filtros: Binarização (Threshold) (cont.)

Python

```python
print(f"Imagem original: min={img.min()}, max={img.max()}")
print(f"Imagem binaria: min={img_binary.min()}, max={img_binary.max()}")
print(f"Valores unicos: {np.unique(img_binary)}")  # [0, 16]

# Visualizar
fig, axes = plt.subplots(1, 2, figsize=(8, 4))
axes[0].imshow(img, cmap='gray')
axes[0].set_title('Original')
axes[1].imshow(img_binary, cmap='gray')
axes[1].set_title(f'Binaria (threshold={threshold})')
for ax in axes:
    ax.axis('off')
plt.show()
```

Filtros: Binarização (Threshold) (cont.)



Filtros: Binarização (Threshold) (cont.)

Python

```python
# GENERALIZANDO: mesma operacao em outros tensores

# Serie temporal: marcar dias quentes
temp_data = np.random.randn(100, 5) * 5 + 20
hot_days = (temp_data > 25).astype(int)
print(f"Dias quentes: {hot_days.sum()} de {hot_days.size}")

# Vendas: identificar meses acima da meta
sales = np.random.randint(100, 1000, (10, 5, 12))
meta = 500
above_target = sales > meta
print(f"Meses acima da meta: {above_target.sum()}")
```

Reordenação de Elementos em Tensores

Modificar a posição de elementos sem alterar valores:

Operações básicas de reordenação:

- **Flip:** Inverter ordem ao longo de um eixo
- **Transpose:** Trocar posição de dimensões
- **Rotação:** Caso especial de transpose + flip

Aplicações gerais (não só imagens!):

- **Imagens:** Flip horizontal/vertical, rotações
- **Séries temporais:** Reverter ordem temporal
- **Dados de vendas:** Reorganizar perspectivas
- **Qualquer tensor:** Mudar ordem de observações

Nota Importante

Estas são operações em índices - não modificam valores!

Transformação: Flip Horizontal

</> Python

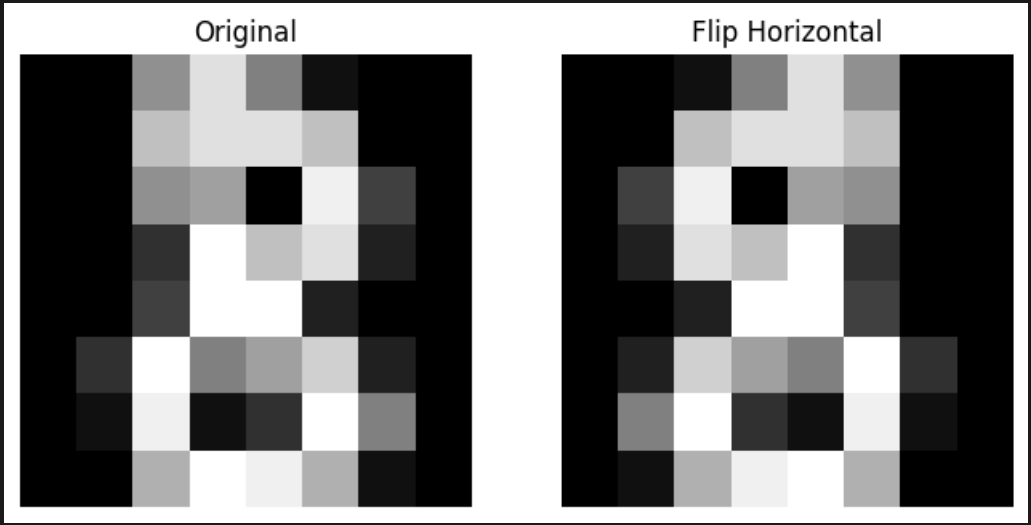
```python
# Espelhar imagem horizontalmente (esquerda <-> direita)

img = X[8]  # Escolher um digito

# Flip horizontal: inverter colunas
img_flip_h = img[:, ::-1]
# ::-1 inverte a ordem dos elementos

print(f"Original: shape {img.shape}")
print(f"Flip horizontal: shape {img_flip_h.shape}")

# Verificar que eh uma view (mesma memoria)
print(f"Original e flip compartilham memoria: {np.shares_memory(img,
    img_flip_h)}")
```

Transformação: Flip Horizontal (cont.)

Python

```python
# Para criar copia independente
img_flip_h_copy = img[:, ::-1].copy()
print(f"Copia independente: {np.shares_memory(img, img_flip_h_copy)}")

# Visualizar
fig, axes = plt.subplots(1, 2, figsize=(8, 4))
axes[0].imshow(img, cmap='gray')
axes[0].set_title('Original')
axes[1].imshow(img_flip_h, cmap='gray')
axes[1].set_title('Flip Horizontal')
for ax in axes:
    ax.axis('off')
plt.show()
```

Transformação: Flip Horizontal (cont.)



Transformação: Flip Horizontal (cont.)

Python

```python
# GENERALIZANDO: Flip funciona em qualquer tensor

# Serie temporal: reverter ordem dos dias
temp_data = np.random.rand(100, 5)  # (dias, sensores)
temp_reversed = temp_data[::-1, :]   # Reverter dias
print(f"Temporal reversed: {temp_reversed.shape}")

# Vendas: reverter ordem dos meses
sales = np.random.rand(10, 5, 12)   # (produtos, regioes, meses)
sales_reversed = sales[:, :, ::-1]  # Reverter meses
print(f"Sales reversed: {sales_reversed.shape}")

# IMPORTANTE: valores nao mudam, apenas a ordem!
```

Transformação: Flip Horizontal (cont.)

Nota Importante

Flip = reordenação de índices - funciona em qualquer dimensão de qualquer tensor!

Transformação: Flip Vertical

Python

```python
# Flip vertical: inverter linhas
img = X[12]
img_flip_v = img[::-1, :]

# Generalizando para outros tensores
# Tensor 3D genérico (A, B, C)
tensor = np.random.rand(10, 8, 6)

# Flip na dimensão 0
flip_dim0 = tensor[::-1, :, :]

# Flip na dimensão 1
flip_dim1 = tensor[:, ::-1, :]
```

Transformação: Flip Vertical (cont.)

Python

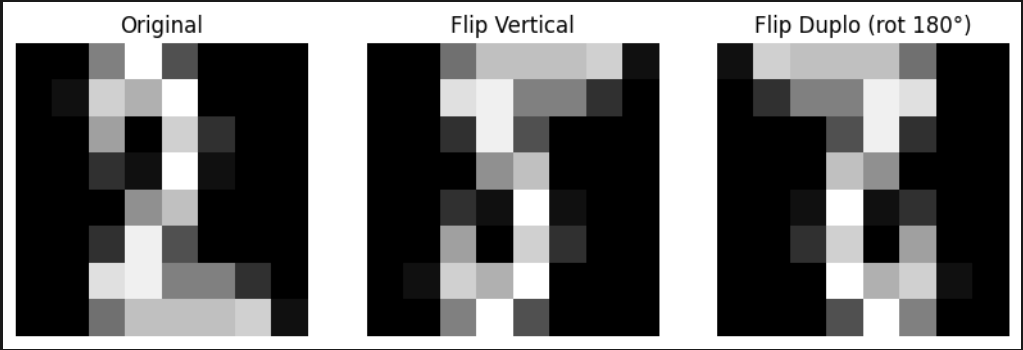
```python
# Flip na dimensão 2
flip_dim2 = tensor[:, :, ::-1]

# Flip múltiplo (duas dimensões)
flip_multi = tensor[::-1, ::-1, :]

print("Flip funciona em qualquer dimensão de qualquer tensor!")
print(f"Original: {tensor.shape}")
print(f"Flip dim0: {flip_dim0.shape}")  # Shape não muda
```

Transformação: Flip Vertical (cont.)

Python

```python
# Visualização compacta
fig, axes = plt.subplots(1, 3, figsize=(10, 3))
axes[0].imshow(img, cmap='gray')
axes[0].set_title('Original')
axes[1].imshow(img_flip_v, cmap='gray')
axes[1].set_title('Flip Vertical')
axes[2].imshow(img[::-1, ::-1], cmap='gray')
axes[2].set_title('Flip Duplo (rot 180°)')
for ax in axes:
    ax.axis('off')
plt.show()
```

Transformação: Flip Vertical (cont.)



Rotação: Caso Especial de Transpose

Rotação 90° = Transpose + Flip (para dados 2D)

</> Python

```python
img = X[13]  # (8, 8)

# NumPy fornece rot90 para conveniência
img_rot90 = np.rot90(img, k=1)    # 90° anti-horário
img_rot180 = np.rot90(img, k=2)   # 180°
img_rot270 = np.rot90(img, k=3)   # 270° (ou -90°)

# Equivalentes manuais:
# rot90 = transpose + flip
img_rot90_manual = img.T[::-1, :]

# rot180 = flip duplo
img_rot180_manual = img[::-1, ::-1]
```

Rotação: Caso Especial de Transpose (cont.)

Python

```python
print(f"rot90 igual manual: {np.array_equal(img_rot90, img_rot90_manual)}")

# IMPORTANTE para dados 3D+:
# rot90 opera em 2D - especificar axes
tensor_3d = np.random.rand(10, 8, 8)
# Rotar cada "imagem" no tensor
rotated_3d = np.rot90(tensor_3d, k=1, axes=(1, 2))
print(f"Tensor 3D rotado: {rotated_3d.shape}")  # (10, 8, 8)
```

Rotação: Visualização Compacta

Python

```python
# Visualizar rotações
fig, axes = plt.subplots(1, 4, figsize=(12, 3))

rotations = [
    (img, '0°'),
    (img_rot90, '90°'),
    (img_rot180, '180°'),
    (img_rot270, '270°')
]

for ax, (img_rot, angle) in zip(axes, rotations):
    ax.imshow(img_rot, cmap='gray')
    ax.set_title(angle)
    ax.axis('off')
```

Rotação: Visualização Compacta (cont.)

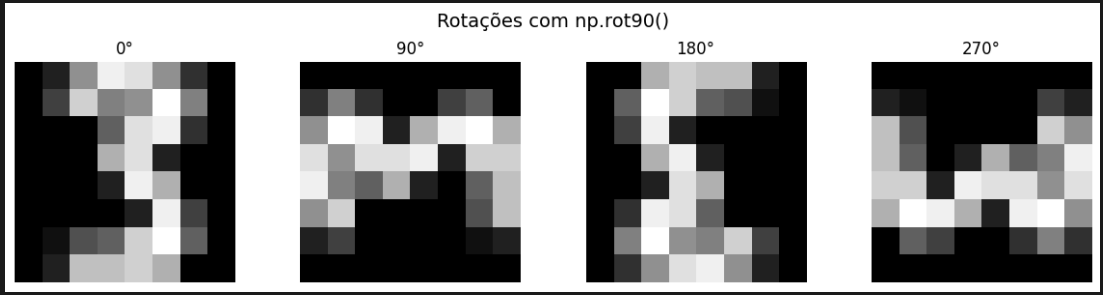
Python

```python
plt.suptitle('Rotações com np.rot90()', fontsize=14)
plt.tight_layout()
plt.show()

# Nota: Para rotações arbitrárias (não múltiplos de 90°),
# usar bibliotecas especializadas (scipy, opencv)
```

Nota Importante

np.rot90 é útil mas limitado a múltiplos de 90° - suficiente para muitos casos!

Rotação: Visualização Compacta (cont.)



Combinar Tensores: Concatenação Horizontal

Concatenar tensores ao longo de um eixo específico

Conceito geral: Juntar múltiplos tensores em uma dimensão

Aplicações:

- Imagens: criar mosaicos visuais
- Séries temporais: juntar períodos
- Dados tabulares: combinar features

Combinar Imagens: Mosaico Horizontal

</> Python

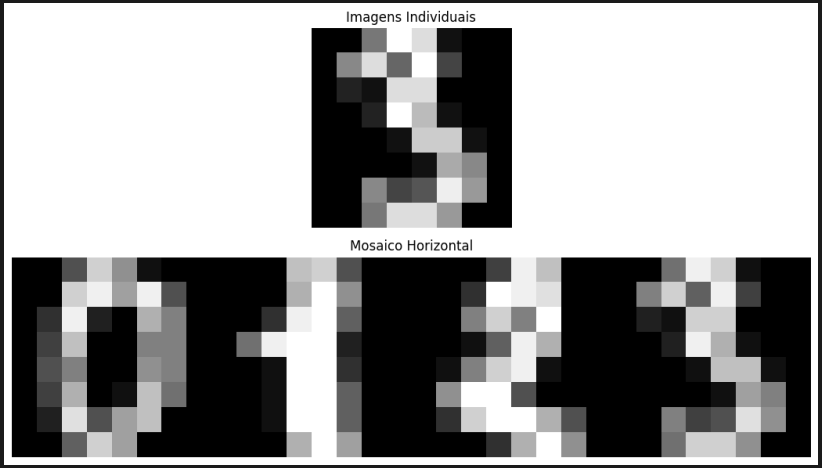
```python
# Concatenar imagens horizontalmente (lado a lado)

# Selecionar 4 digitos diferentes
indices = [0, 1, 2, 3]
images = [X[i] for i in indices]

# Concatenar horizontalmente
mosaic_h = np.hstack(images)
# Ou: mosaic_h = np.concatenate(images, axis=1)

print(f"Imagens individuais: {images[0].shape}")
print(f"Mosaico horizontal: {mosaic_h.shape}")  # (8, 32)
```

Combinar Imagens: Mosaico Horizontal (cont.)

</> Python

```python
# Visualizar
fig, axes = plt.subplots(2, 1, figsize=(12, 6))
# Imagens individuais
for i, idx in enumerate(indices):
    axes[0].imshow(X[idx], cmap='gray',
                   extent=[i*8, (i+1)*8, 0, 8])
axes[0].set_title('Imagens Individuais')
axes[0].axis('off')
# Mosaico
axes[1].imshow(mosaic_h, cmap='gray')
axes[1].set_title('Mosaico Horizontal')
axes[1].axis('off')
plt.tight_layout()
plt.show()
```

Combinar Imagens: Mosaico Horizontal (cont.)



Combinar Imagens: Mosaico Horizontal (cont.)

</> Python

```python
# Generalização: concatenar qualquer tensor
# Serie temporal: juntar diferentes periodos
periodo1 = np.random.rand(30, 5)  # 30 dias
periodo2 = np.random.rand(40, 5)  # 40 dias
periodos_combined = np.concatenate([periodo1, periodo2], axis=0)
print(f"Períodos combinados: {periodos_combined.shape}")  # (70, 5)

# Vendas: juntar regiões
vendas_regiao1 = np.random.rand(10, 3, 12)  # 3 subregions
vendas_regiao2 = np.random.rand(10, 2, 12)  # 2 subregions
vendas_combined = np.concatenate([vendas_regiao1, vendas_regiao2], axis=1)
print(f"Regiões combinadas: {vendas_combined.shape}")  # (10, 5, 12)
```

Combinar Tensores: Concatenação Vertical

Python

```python
# Concatenar verticalmente
indices = [10, 20, 30]
images = [X[i] for i in indices]

# np.vstack = concatenate em axis=0
mosaic_v = np.vstack(images)
# Equivalente a:
mosaic_v = np.concatenate(images, axis=0)

print(f"Vertical stack: {mosaic_v.shape}")  # (24, 8)
```

Combinar Tensores: Concatenação Vertical (cont.)

Python

```python
# Exemplo genérico
arr1 = np.random.rand(5, 10)
arr2 = np.random.rand(3, 10)
combined = np.vstack([arr1, arr2])  # (8, 10)

# REGRA: todas as dimensões devem ser compatíveis exceto a de concatenação
```

Nota Importante

vstack = concatenate(axis=0), hstack = concatenate(axis=1)

Combinar Tensores: Estruturas Complexas

Construir estruturas complexas com múltiplas concatenações:

Python

```python
# Exemplo: Grid 4x5 de imagens
n_rows, n_cols = 4, 5

# Criar cada linha
rows = []
for i in range(n_rows):
    # Imagens desta linha
    start_idx = i * n_cols
    row_images = [X[j] for j in range(start_idx, start_idx + n_cols)]
    # Concatenar horizontalmente
    row_mosaic = np.hstack(row_images)
    rows.append(row_mosaic)
```

Combinar Tensores: Estruturas Complexas (cont.)

</> Python

```python
# Concatenar linhas verticalmente
grid = np.vstack(rows)
print(f"Grid final: {grid.shape}")  # (32, 40) = 4*8 × 5*8

# PRINCÍPIO GERAL:
# Concatenações aninhadas permitem estruturas arbitrárias
```

Stack: Criar Nova Dimensão

Diferença entre concatenate e stack:

</> Python

```python
# Dois arrays 2D
arr1 = np.random.rand(8, 8)
arr2 = np.random.rand(8, 8)

# CONCATENATE: junta em dimensão existente
concat_result = np.concatenate([arr1, arr2], axis=0)
print(f"Concatenate: {concat_result.shape}")  # (16, 8)

# STACK: cria nova dimensão
stack_result = np.stack([arr1, arr2], axis=0)
print(f"Stack: {stack_result.shape}")  # (2, 8, 8)
```

Stack: Criar Nova Dimensão (cont.)

Python

```python
# Stack pode criar dimensão em qualquer posição
stack_axis1 = np.stack([arr1, arr2], axis=1)
print(f"Stack axis=1: {stack_axis1.shape}")  # (8, 2, 8)

stack_axis2 = np.stack([arr1, arr2], axis=2)
print(f"Stack axis=2: {stack_axis2.shape}")  # (8, 8, 2)

# Uso comum: criar batch de amostras individuais
# Múltiplas imagens (8,8) -> batch (n, 8, 8)
```

O Conceito Central: Operações Genéricas

TODAS as operações que fizemos aplicam-se a QUALQUER TENSOR 3D+

Não importa se é:

- **Imagens:** (n_imagens, altura, largura)
- **Séries temporais:** (tempo, variáveis, locais)
- **Vendas:** (produtos, regiões, períodos)
- **Dados volumétricos:** (x, y, z)
- **Qualquer outra estrutura 3D+**

O Conceito Central: Operações Genéricas

As ferramentas são as mesmas:

- Indexação e slicing
- Broadcasting
- Agregações com axis
- Máscaras booleanas
- Transformações vetorizadas

Nota Importante

Apenas a INTERPRETAÇÃO muda!

Exemplo Comparativo: Mesma Operação, Diferentes Dados

Operação: Calcular média ao longo do primeiro eixo

</> Python

```python
# CASO 1: Imagens (1797, 8, 8)
X_images = digits.images
mean_images = X_images.mean(axis=0)  # (8, 8)
# INTERPRETACAO: pixel medio atraves de todas as imagens

# CASO 2: Series temporais (100, 5, 3) [dias, sensores, horarios]
temp_data = np.random.randn(100, 5, 3) * 5 + 20
mean_temporal = temp_data.mean(axis=0)  # (5, 3)
# INTERPRETACAO: leitura media de cada sensor em cada horario
```

Exemplo Comparativo: Mesma Operação, Diferentes Dados

Operação: Calcular média ao longo do primeiro eixo

</> Python

```python
# CASO 3: Vendas (10, 5, 12) [produtos, regioes, meses]
sales = np.random.randint(100, 1000, (10, 5, 12))
mean_sales = sales.mean(axis=0)  # (5, 12)
# INTERPRETACAO: venda media por regiao em cada mes

print("MESMA OPERACAO (mean(axis=0)), INTERPRETACOES DIFERENTES:")
print(f"Imagens: {mean_images.shape} - pixel medio")
print(f"Temporal: {mean_temporal.shape} - sensor-horario medio")
print(f"Vendas: {mean_sales.shape} - regiao-mes medio")
```

Tabela Comparativa: Operações em Diferentes Domínios

Como cada operação se traduz em diferentes tipos de dados:

Operação	Imagens	Temporal	Vendas
`mean(axis=0)`	Pixel médio	Sensor médio	Região média
`sum(axis=1)`	Soma por linha	Soma por sensor	Soma por região
`max(axis=2)`	Max por coluna	Max por horário	Max por mês
`std(axis=(1,2))`	Std espacial	Std sensor-hora	Std região-mês
Broadcasting	Normalizar pixels	Normalizar sensores	Normalizar regiões
Boolean mask	Pixels > thresh	Temp > limite	Vendas > meta
Reshape	Flatten imagem	Reorganizar series	Mudar perspectiva
Transpose	Trocar H↔W	Trocar tempo↔sensor	Trocar prod↔região

Nota Importante

Princípio fundamental: A mecânica é idêntica, só a semântica muda!

Broadcasting: Aplicação Universal

Broadcasting funciona EXATAMENTE igual em qualquer tensor:

</> Python

```python
# IMAGENS: Normalizar cada imagem pela sua media
X_images = np.random.rand(100, 8, 8)
means_img = X_images.mean(axis=(1,2), keepdims=True)  # (100, 1, 1)
X_norm_img = X_images / means_img  # Broadcasting

# TEMPORAL: Normalizar cada dia pela media dos sensores
temp_data = np.random.rand(100, 5, 3)
means_temp = temp_data.mean(axis=1, keepdims=True)  # (100, 1, 3)
temp_norm = temp_data / means_temp  # Broadcasting
```

Broadcasting: Aplicação Universal (cont.)

Python

```python
# VENDAS: Normalizar cada produto pela media das regioes
sales = np.random.rand(10, 5, 12)
means_sales = sales.mean(axis=1, keepdims=True)  # (10, 1, 12)
sales_norm = sales / means_sales  # Broadcasting

print("Broadcasting funciona IDENTICAMENTE em todos!")
print(f"Imagens: {X_images.shape} / {means_img.shape} = {X_norm_img.shape}
    ")
print(f"Temporal: {temp_data.shape} / {means_temp.shape} = {temp_norm.
    shape}")
print(f"Vendas: {sales.shape} / {means_sales.shape} = {sales_norm.shape}")
```

Indexação: Aplicação Universal

Slicing e indexação trabalham identicamente:

</> Python

```python
# IMAGENS (1797, 8, 8)
X = digits.images
primeira_imagem = X[0]           # (8, 8) - uma imagem
centro_todas = X[:, 3:5, 3:5]    # (1797, 2, 2) - centro de todas

# TEMPORAL (100, 5, 3) [dias, sensores, horarios]
temp_data = np.random.rand(100, 5, 3)
primeiro_dia = temp_data[0]           # (5, 3) - um dia
manha_todos = temp_data[:, :, 0]      # (100, 5) - manhã de todos os dias
```

Indexação: Aplicação Universal (cont.)

Python

```python
# VENDAS (10, 5, 12) [produtos, regioes, meses]
sales = np.random.rand(10, 5, 12)
primeiro_produto = sales[0]           # (5, 12) - um produto
trimestre1_todos = sales[:, :, 0:3]   # (10, 5, 3) - Q1 de todos

print("Indexacao: MESMA sintaxe, significados diferentes")
print(f"X[0] = primeira imagem")
print(f"temp_data[0] = primeiro dia")
print(f"sales[0] = primeiro produto")
```

Máscaras Booleanas: Aplicação Universal

Boolean indexing funciona em qualquer contexto:

</> Python

```python
# IMAGENS: Filtrar imagens claras
X = digits.images
mean_brightness = X.mean(axis=(1,2))
bright_mask = mean_brightness > 10
bright_images = X[bright_mask]
print(f"Imagens claras: {bright_images.shape[0]}/{len(X)}")

# TEMPORAL: Filtrar dias quentes
temp_data = np.random.randn(100, 5, 3) * 5 + 20
mean_daily_temp = temp_data.mean(axis=(1,2))
hot_days_mask = mean_daily_temp > 25
hot_days_data = temp_data[hot_days_mask]
print(f"Dias quentes: {hot_days_data.shape[0]}/100")
```

Máscaras Booleanas: Aplicação Universal (cont.)

Python

```python
# VENDAS: Filtrar produtos de sucesso
sales = np.random.randint(100, 1000, (10, 5, 12))
total_sales = sales.sum(axis=(1,2))
successful_mask = total_sales > 50000
successful_products = sales[successful_mask]
print(f"Produtos de sucesso: {successful_products.shape[0]}/10")

# MESMO PADRÃO: condicao -> mascara -> filtrar
```

Transformações de Shape: Aplicação Universal

Reshape, transpose, stack - mesmas regras:

</> Python

```python
# RESHAPE funciona em qualquer tensor
X_img = np.random.rand(10, 8, 8)      # Imagens
X_flat = X_img.reshape(10, -1)         # (10, 64) - flatten espacial

temp = np.random.rand(100, 5, 3)       # Temporal
temp_flat = temp.reshape(100, -1)      # (100, 15) - flatten sensores

sales = np.random.rand(10, 5, 12)      # Vendas
sales_flat = sales.reshape(10, -1)     # (10, 60) - flatten regioes-meses
```

Transformações de Shape: Aplicação Universal (cont.)

Python

```python
# TRANSPOSE muda perspectiva
X_T = X_img.transpose(0, 2, 1)        # (10, 8, 8) -> (10, 8, 8)
temp_T = temp.transpose(1, 0, 2)      # (100, 5, 3) -> (5, 100, 3)
sales_T = sales.transpose(1, 0, 2)    # (10, 5, 12) -> (5, 10, 12)

print("Transformacoes: MESMA mecanica, interpretacoes diferentes")
print(f"Img transpose: trocar  alturalargura")
print(f"Temporal transpose: sensores como dimensao principal")
print(f"Sales transpose: regioes como dimensao principal")
```

Preparação Mental para Bloco 3

Você já sabe TUDO que precisa!

No Bloco 3, veremos:

- Séries temporais multivariadas
- Dados de vendas 3D
- Dados volumétricos espaciais

MAS... você já aprendeu:

- ✓ Como criar e manipular tensores 3D
- ✓ Como usar axis para agregar
- ✓ Como aplicar broadcasting
- ✓ Como indexar e fatiar
- ✓ Como transformar shapes
- ✓ Como usar máscaras
- ✓ Como vetorizar operações

Preparação Mental para Bloco 3

Diferença no Bloco 3:

- Não serão imagens
- Interpretação contextual diferente
- Mas a mecânica é **EXATAMENTE A MESMA!**

Nota Importante

Usamos imagens para aprender porque são visuais - agora aplicamos em tudo!

Checklist de Conceitos Aprendidos no Bloco 2

Antes de prosseguir, você deve conseguir:

Estrutura e Exploração:

- ☐ Entender imagens como tensores 3D
- ☐ Calcular estatísticas globais e por grupo
- ☐ Criar visualizações apropriadas

Operações Fundamentais:

- ☐ Usar indexação e slicing avançado em 3D
- ☐ Aplicar fancy indexing e boolean indexing
- ☐ Dominar axis e keepdims
- ☐ Utilizar broadcasting eficientemente
- ☐ Criar e aplicar máscaras booleanas

Checklist de Conceitos Aprendidos no Bloco 2

Antes de prosseguir, você deve conseguir:

Transformações:

- ☐ Aplicar operações aritméticas e condicionais
- ☐ Normalizar dados (global, por amostra, por feature)
- ☐ Transformar shapes (reshape, transpose)
- ☐ Combinar arrays (stack, concatenate)

Performance:

- ☐ Vetorizar operações em batch
- ☐ Comparar performance: loops vs vetorização

Bloco 3

Outros Tipos de Dados Multidimensionais

Diversidade de Dados Multidimensionais

Imagens são apenas um exemplo de tensores!

Outros tipos comuns de dados 3D+:

- **Séries temporais multivariadas:** múltiplas variáveis ao longo do tempo
  - Sensores, estações meteorológicas, sinais biomédicos
- **Dados de negócios 3D:** produtos/clientes × regiões × períodos
  - Vendas, estoque, métricas de performance
- **Dados volumétricos:** grids espaciais 3D
  - Temperatura, pressão, densidade em espaço físico

Conexão com Bloco 2:

- Bloco 2: Aprendemos operações usando IMAGENS como exemplo
- Bloco 3: Aplicamos as MESMAS operações em outros dados
- Ferramentas: axis, broadcasting, indexação, máscaras - IDÊNTICAS!

Diversidade de Dados Multidimensionais

Nota Importante

As técnicas de manipulação do Bloco 2 aplicam-se DIRETAMENTE aqui!

Caso 1: Séries Temporais Multivariadas

Conexão com Aula 11 (Séries Temporais):

- **Aula 11:** 1 variável ao longo do tempo (1D ou 2D)
- **Hoje:** múltiplas variáveis ao longo do tempo (3D)

Estrutura de dados 3D:

- **Dimensão 0:** tempo (dias, horas, minutos...)
- **Dimensão 1:** variáveis/sensores/features
- **Shape:** (n_timesteps, n_variables)

Exemplo de hoje:

- 5 sensores de temperatura
- Medições diárias durante 100 dias
- Shape: (100, 5) - tensor 2D (mas conceito se estende a 3D+)

Caso 1: Séries Temporais Multivariadas

Operações do Bloco 2 que usaremos:

- `mean(axis=0)`: Média temporal (ao longo dos dias)
- `mean(axis=1)`: Média por dia (ao longo dos sensores)
- Broadcasting: Normalizar cada sensor independentemente
- Boolean indexing: Identificar dias com anomalias
- Correlação: Similaridade entre sensores

Séries Temporais: Criar Dados Sintéticos

Python

```python
import numpy as np
import matplotlib.pyplot as plt

# Configurar seed para reprodutibilidade
np.random.seed(42)

# Parametros
n_days = 100      # 100 dias de medicoes
n_sensors = 5     # 5 sensores diferentes

# Temperatura base (20 graus Celsius)
temp_base = 20

# Gerar dados com ruido aleatorio
temp_data = np.random.randn(n_days, n_sensors) * 5 + temp_base
```

Séries Temporais: Criar Dados Sintéticos (cont.)

Python

```python
# Adicionar tendencia temporal (aquecimento gradual)
trend = np.linspace(0, 10, n_days)  # Aumento de 0 a 10 graus
temp_data = temp_data + trend[:, np.newaxis]  # Broadcasting

print(f"Shape dos dados: {temp_data.shape}")  # (100, 5)
print(f"Temperatura media global: {temp_data.mean():.2f}°C")
print(f"Temperatura min: {temp_data.min():.2f}°C")
print(f"Temperatura max: {temp_data.max():.2f}°C")
```

Séries Temporais: Explorar Estrutura

Python

```python
# Entender a estrutura dos dados

print(f"Shape: {temp_data.shape}")
print(f"Dimensoes (rank): {temp_data.ndim}")
print(f"Total de elementos: {temp_data.size}")
print(f"Dtype: {temp_data.dtype}")

# Primeiros 5 dias, todos os sensores
print(f"\nPrimeiros 5 dias:")
print(temp_data[:5])
# Cada linha = 1 dia, cada coluna = 1 sensor

# Sensor 0, primeiros 10 dias
print(f"\nSensor 0, primeiros 10 dias:")
print(temp_data[:10, 0])
```

Séries Temporais: Explorar Estrutura (cont.)

Python

```python
# Dia 50, todos os sensores
print(f"\nDia 50, todos os sensores:")
print(temp_data[50, :])

# Ultimo dia
print(f"\nUltimo dia:")
print(temp_data[-1])
```

Séries Temporais: Explorar Estrutura (cont.)

Python

```python
# Shape: (100, 5)
# Dimensoes (rank): 2
# Total de elementos: 500
# Dtype: float64
# Primeiros 5 dias:
# [[22.48 19.31 23.24 27.62 18.83]
# [18.93 28.   23.94 17.75 22.81]
#  [17.88 17.87 21.41 10.64 11.58]
#  [17.49 15.24 21.87 15.76 13.24]
#  [27.73 19.28 20.74 13.28 17.68]]
# Sensor 0, primeiros 10 dias:
# [22.48 18.93 17.88 17.49 27.73 21.06 17.6  14.6  24.5  17.31]
# Dia 50, todos os sensores:
# [18.75 29.64 35.66 30.21 17.45]
# Ultimo dia:
# [32.69 24.81 29.05 25.62 23.09]
```

Séries Temporais: Operações por Eixo Temporal

Python

```python
# Analise ao longo do tempo (axis=0)
# Temperatura media de cada sensor ao longo de todo o periodo
temp_mean_per_sensor = temp_data.mean(axis=0)
print(f"Media por sensor (100 dias): {temp_mean_per_sensor}")
print(f"Shape: {temp_mean_per_sensor.shape}")  # (5,)

# Desvio padrao de cada sensor
temp_std_per_sensor = temp_data.std(axis=0)
print(f"\nDesvio padrao por sensor: {temp_std_per_sensor}")

# Min e max de cada sensor
temp_min_per_sensor = temp_data.min(axis=0)
temp_max_per_sensor = temp_data.max(axis=0)
print(f"\nMin por sensor: {temp_min_per_sensor}")
print(f"Max por sensor: {temp_max_per_sensor}")
```

Séries Temporais: Operações por Eixo Temporal (cont.)

Python

```python
# Identificar sensor mais estavel (menor std)
sensor_mais_estavel = temp_std_per_sensor.argmin()
print(f"\nSensor mais estavel: Sensor {sensor_mais_estavel}")
print(f"Std: {temp_std_per_sensor[sensor_mais_estavel]:.2f}")

# Identificar sensor mais variavel
sensor_mais_variavel = temp_std_per_sensor.argmax()
print(f"Sensor mais variavel: Sensor {sensor_mais_variavel}")
print(f"Std: {temp_std_per_sensor[sensor_mais_variavel]:.2f}")
```

Séries Temporais: Operações por Eixo Temporal (cont.)

Python

```
# Media por sensor (100 dias): [24.78 25.38 24.53 25.64 24.84]
# Shape: (5,)

# Desvio padrao por sensor: [5.21 5.58 5.26 6.19 6.37]

# Min por sensor: [12.63 14.62  9.05 10.64  8.32]
# Max por sensor: [39.12 37.3  37.16 44.99 43.41]

# Sensor mais estavel: Sensor 0
# Std: 5.21
# Sensor mais variavel: Sensor 4
# Std: 6.37
```

Séries Temporais: Operações por Variável

Python

```python
# Analise ao longo dos sensores (axis=1)

# Temperatura media de cada dia (media dos 5 sensores)
temp_mean_per_day = temp_data.mean(axis=1)
print(f"Media por dia: shape {temp_mean_per_day.shape}")  # (100,)
print(f"Primeiros 10 dias: {temp_mean_per_day[:10]}")

# Variabilidade entre sensores em cada dia
temp_std_per_day = temp_data.std(axis=1)
print(f"\nStd por dia: shape {temp_std_per_day.shape}")
```

Séries Temporais: Operações por Variável (cont.)

</> Python

```python
# Identificar dia com maior variabilidade entre sensores
dia_maior_variabilidade = temp_std_per_day.argmax()
print(f"\nDia com maior variabilidade: Dia {dia_maior_variabilidade}")
print(f"Std: {temp_std_per_day[dia_maior_variabilidade]:.2f}")
print(f"Temperaturas nesse dia: {temp_data[dia_maior_variabilidade]}")

# Range de temperatura em cada dia
temp_range_per_day = temp_data.max(axis=1) - temp_data.min(axis=1)
print(f"\nRange medio entre sensores: {temp_range_per_day.mean():.2f}°C")
```

Séries Temporais: Operações por Variável (cont.)

Python

```
# Media por dia: shape (100,)
# Primeiros 10 dias: [22.3  22.29 15.88 16.72 19.74 18.95 21.61 16.6
    19.82 19.37]

# Std por dia: shape (100,)

# Dia com maior variabilidade: Dia 41
# Std: 7.92
# Temperaturas nesse dia: [19.45 26.72 26.71 26.72 43.41]

# Range medio entre sensores: 11.43°C
```

Séries Temporais: Agregações Multi-dimensionais

</> Python

```python
# Combinando operacoes em multiplas dimensoes

# Media movel de 7 dias para cada sensor
window = 7
temp_moving_avg = np.zeros_like(temp_data)

for i in range(n_sensors):
    for j in range(window, n_days):
        temp_moving_avg[j, i] = temp_data[j-window:j, i].mean()

print(f"Media movel calculada: shape {temp_moving_avg.shape}")

# Temperatura maxima global
temp_max_global = temp_data.max()
print(f"\nTemperatura maxima global: {temp_max_global:.2f}°C")
```

Séries Temporais: Agregações Multi-dimensionais (cont.)

</> Python

```python
# Encontrar onde ocorreu o maximo
max_day, max_sensor = np.unravel_index(temp_data.argmax(), temp_data.shape)
print(f"Ocorreu no dia {max_day}, sensor {max_sensor}")

# Temperatura minima global
temp_min_global = temp_data.min()
min_day, min_sensor = np.unravel_index(temp_data.argmin(), temp_data.shape)
print(f"\nTemperatura minima: {temp_min_global:.2f}°C")
print(f"Ocorreu no dia {min_day}, sensor {min_sensor}")
```

Séries Temporais: Agregações Multi-dimensionais (cont.)

Python

```python
# Media movel calculada: shape (100, 5)

# Temperatura maxima global: 44.99°C
# Ocorreu no dia 95, sensor 3

# Temperatura minima: 8.32°C
# Ocorreu no dia 14, sensor 4
```

Séries Temporais: Visualizações

Python

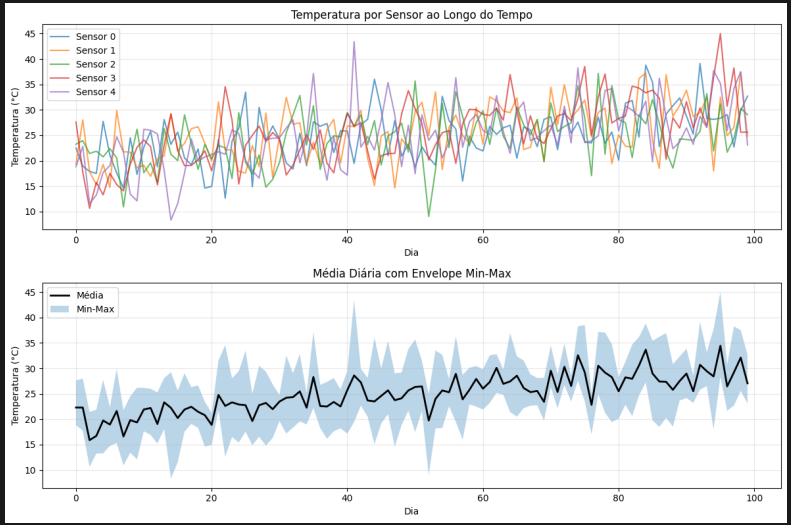
```python
# Visualizar series temporais

fig, axes = plt.subplots(2, 1, figsize=(12, 8))

# Plot 1: Todas as series temporais
for i in range(n_sensors):
    axes[0].plot(temp_data[:, i], label=f'Sensor {i}', alpha=0.7)

axes[0].set_xlabel('Dia')
axes[0].set_ylabel('Temperatura (°C)')
axes[0].set_title('Temperatura por Sensor ao Longo do Tempo')
axes[0].legend()
axes[0].grid(True, alpha=0.3)
```

Séries Temporais: Visualizações (cont.)

Python

```python
# Plot 2: Media e envelope (min/max)
mean_daily = temp_data.mean(axis=1)
min_daily = temp_data.min(axis=1)
max_daily = temp_data.max(axis=1)
axes[1].plot(mean_daily, label='Média', linewidth=2, color='black')
axes[1].fill_between(range(n_days), min_daily, max_daily,
                     alpha=0.3, label='Min-Max')
axes[1].set_xlabel('Dia')
axes[1].set_ylabel('Temperatura (°C)')
axes[1].set_title('Média Diária com Envelope Min-Max')
axes[1].legend()
axes[1].grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
```

Séries Temporais: Visualizações (cont.)



Séries Temporais: Reshape e Transformações

Python

```python
# Diferentes representacoes dos mesmos dados

# Formato atual: (100, 5) - wide format
print(f"Wide format: {temp_data.shape}")

# Converter para long format (1D)
temp_long = temp_data.flatten()  # ou temp_data.ravel()
print(f"Long format (1D): {temp_long.shape}")  # (500,)

# Reconstruir de 1D para 2D
temp_reconstructed = temp_long.reshape(n_days, n_sensors)
print(f"Reconstruido: {temp_reconstructed.shape}")

# Verificar igualdade
print(f"Dados iguais? {np.allclose(temp_data, temp_reconstructed)}")
```

Séries Temporais: Reshape e Transformações (cont.)

Python

```python
# Transpor: sensores x dias (inverte perspectiva)
temp_transposed = temp_data.T
print(f"\nTransposto: {temp_transposed.shape}")  # (5, 100)
# Agora cada linha eh um sensor, cada coluna eh um dia

# Estatisticas no transposto (axis muda de significado!)
mean_per_sensor_v2 = temp_transposed.mean(axis=1)  # Media ao longo dos dias
print(f"Media por sensor (via transpose): {mean_per_sensor_v2}")
```

Séries Temporais: Reshape e Transformações (cont.)

Python

```
# Wide format: (100, 5)
# Long format (1D): (500,)
# Reconstruido: (100, 5)
# Dados iguais? True

# Transposto: (5, 100)
# Media por sensor (via transpose): [24.78 25.38 24.53 25.64 24.84]
```

Caso 2: Dados de Vendas 3D

Estrutura de dados de negócios em 3 dimensões:

- **Dimensão 0:** Produtos (10 produtos diferentes)
- **Dimensão 1:** Regiões (5 regiões geográficas)
- **Dimensão 2:** Tempo (12 meses do ano)
- **Shape:** (10, 5, 12)

Perguntas que podemos responder:

- Qual produto vende mais?
- Qual região tem melhores resultados?
- Há sazonalidade nas vendas?
- Qual produto-região tem melhor performance?
- Como evoluem as vendas ao longo do ano?

Caso 2: Dados de Vendas 3D

Nota Importante

Tensor 3D permite análise em múltiplas perspectivas simultaneamente!

Vendas 3D: Criar Dados Sintéticos

Python

```python
# Dimensoes
n_products = 10    # 10 produtos
n_regions = 5      # 5 regioes
n_months = 12      # 12 meses

# Gerar vendas base (entre 100 e 1000 unidades)
sales = np.random.randint(100, 1000, size=(n_products, n_regions, n_months
    ))

# Adicionar sazonalidade (maior vendas no final do ano)
seasonal_factor = np.array([0.8, 0.9, 1.0, 1.1, 1.2, 1.3,
                            1.2, 1.1, 1.0, 0.9, 1.1, 1.4])

# Aplicar sazonalidade (broadcasting)
sales = (sales * seasonal_factor[np.newaxis, np.newaxis, :]).astype(int)
```

Vendas 3D: Criar Dados Sintéticos (cont.)

Python

```python
print(f"Shape dos dados de vendas: {sales.shape}")  # (10, 5, 12)
print(f"Total de vendas no ano: {sales.sum():,} unidades")
print(f"Media mensal: {sales.mean():.2f} unidades")
print(f"Vendas min: {sales.min()}, max: {sales.max()}")
```

Vendas 3D: Análise por Produto

Python

```python
# Analisar vendas agregando regioes e meses (axis=(1, 2))

# Total de vendas por produto (somando todas regioes e todos meses)
sales_per_product = sales.sum(axis=(1, 2))
print(f"Vendas por produto: {sales_per_product}")
print(f"Shape: {sales_per_product.shape}")  # (10,)

# Identificar top 3 produtos
top3_indices = sales_per_product.argsort()[-3:][::-1]
print(f"\nTop 3 produtos:")
for rank, idx in enumerate(top3_indices, 1):
    print(f"  {rank}. Produto {idx}: {sales_per_product[idx]:,} unidades")
```

Vendas 3D: Análise por Produto (cont.)

Python

```python
# Identificar bottom 3 produtos
bottom3_indices = sales_per_product.argsort()[:3]
print(f"\nBottom 3 produtos:")
for rank, idx in enumerate(bottom3_indices, 1):
    print(f"  {rank}. Produto {idx}: {sales_per_product[idx]:,} unidades")

# Produto com maior venda media mensal
avg_per_product = sales.mean(axis=(1, 2))
print(f"\nProduto com maior media mensal: Produto {avg_per_product.argmax()}")
print(f"Media: {avg_per_product.max():.2f} unidades/mes")
```

Vendas 3D: Análise por Região

Python

```python
# Analisar vendas agregando produtos e meses (axis=(0, 2))

# Total de vendas por regiao
sales_per_region = sales.sum(axis=(0, 2))
print(f"Vendas por regiao: {sales_per_region}")
print(f"Shape: {sales_per_region.shape}")  # (5,)

# Ranking de regioes
regions_ranked = sales_per_region.argsort()[::-1]
print(f"\nRanking de regioes:")
for rank, idx in enumerate(regions_ranked, 1):
    print(f"  {rank}. Regiao {idx}: {sales_per_region[idx]:,} unidades")
```

Vendas 3D: Análise por Região (cont.)

Python

```python
# Regiao com maior crescimento entre primeiro e ultimo mes
# Vendas totais por regiao em cada mes: (5, 12)
sales_per_region_month = sales.sum(axis=0)

growth_per_region = sales_per_region_month[:, -1] - sales_per_region_month
    [:, 0]
print(f"\nCrescimento (mes 12 - mes 1) por regiao:")
print(growth_per_region)

best_growth_region = growth_per_region.argmax()
print(f"Regiao com maior crescimento: Regiao {best_growth_region}")
print(f"Crescimento: {growth_per_region[best_growth_region]:,} unidades")
```

Vendas 3D: Análise Temporal

Python

```python
# Analisar vendas agregando produtos e regioes (axis=(0, 1))

# Total de vendas por mes
sales_per_month = sales.sum(axis=(0, 1))
print(f"Vendas por mes: {sales_per_month}")
print(f"Shape: {sales_per_month.shape}")  # (12,)

# Identificar melhor e pior mes
best_month = sales_per_month.argmax()
worst_month = sales_per_month.argmin()
print(f"\nMelhor mes: Mes {best_month + 1} ({sales_per_month[best_month]:,} unidades)")
print(f"Pior mes: Mes {worst_month + 1} ({sales_per_month[worst_month]:,} unidades)")
```

Vendas 3D: Análise Temporal (cont.)

Python

```python
# Calcular crescimento mes a mes
growth_month_to_month = np.diff(sales_per_month)
print(f"\nCrescimento mes a mes: {growth_month_to_month}")

# Mes com maior crescimento
max_growth_month = growth_month_to_month.argmax()
print(f"Maior crescimento: entre mes {max_growth_month + 1} e {
    max_growth_month + 2}")
print(f"Crescimento: {growth_month_to_month[max_growth_month]:,} unidades"
    )

# Verificar sazonalidade (correlacao com fator sazonal)
print(f"\nPadrao sazonal aplicado: {seasonal_factor}")
```

Vendas 3D: Agregações Complexas

Python

```python
# Combinar multiplas perspectivas

# Melhor combinacao produto-regiao (ignorando tempo)
sales_product_region = sales.sum(axis=2)  # Shape: (10, 5)
print(f"Vendas por produto-regiao: shape {sales_product_region.shape}")

# Encontrar melhor combinacao
best_prod, best_reg = np.unravel_index(sales_product_region.argmax(),
                                       sales_product_region.shape)
print(f"\nMelhor combinacao: Produto {best_prod}, Regiao {best_reg}")
print(f"Total: {sales_product_region[best_prod, best_reg]:,} unidades")
```

Vendas 3D: Agregações Complexas (cont.)

Python

```python
# Pior combinacao
worst_prod, worst_reg = np.unravel_index(sales_product_region.argmin(),
                                         sales_product_region.shape)
print(f"Pior combinacao: Produto {worst_prod}, Regiao {worst_reg}")
print(f"Total: {sales_product_region[worst_prod, worst_reg]:,} unidades")

# Media de vendas por dimensao
print(f"\nMedia por produto: {sales.mean(axis=(1, 2)).mean():.2f}")
print(f"Media por regiao: {sales.mean(axis=(0, 2)).mean():.2f}")
print(f"Media por mes: {sales.mean(axis=(0, 1)).mean():.2f}")

```

Vendas 3D: Agregações Complexas (cont.)

Python

```python
# Vendas por produto-regiao: shape (10, 5)

# Melhor combinacao: Produto 7, Regiao 0
# Total: 8,916 unidades
# Pior combinacao: Produto 5, Regiao 3
# Total: 4,930 unidades

# Media por produto: 589.66
# Media por regiao: 589.66
# Media por mes: 589.66
```

Vendas 3D: Encontrar Top Performers

Python

```python
# Identificar top performers em diferentes dimensoes

# Top 3 produto-regiao
sales_pr_flat = sales_product_region.flatten()
top3_pr_indices = sales_pr_flat.argsort()[-3:][::-1]

print("Top 3 combinacoes Produto-Regiao:")
for rank, idx in enumerate(top3_pr_indices, 1):
    prod_idx, reg_idx = np.unravel_index(idx, sales_product_region.shape)
    total = sales_product_region[prod_idx, reg_idx]
    print(f"  {rank}. P{prod_idx}-R{reg_idx}: {total:,} unidades")
```

Vendas 3D: Encontrar Top Performers (cont.)

Python

```python
# Melhor mes para cada produto
best_month_per_product = sales.sum(axis=1).argmax(axis=1)
print(f"\nMelhor mes para cada produto:")
for prod in range(5):  # Mostrar apenas primeiros 5
    mes = best_month_per_product[prod]
    vendas = sales[prod, :, mes].sum()
    print(f"  Produto {prod}: Mes {mes + 1} ({vendas:,} unidades)")

# Produtos que tiveram crescimento consistente
# (mais vendas no Q4 do que no Q1)
q1_sales = sales[:, :, 0:3].sum(axis=(1, 2))
q4_sales = sales[:, :, 9:12].sum(axis=(1, 2))
growth = q4_sales - q1_sales
growing_products = growth > 0
print(f"\nProdutos com crescimento Q1->Q4: {growing_products.sum()}/{
    n_products}")
```

Vendas 3D: Visualização como Heatmap

Python

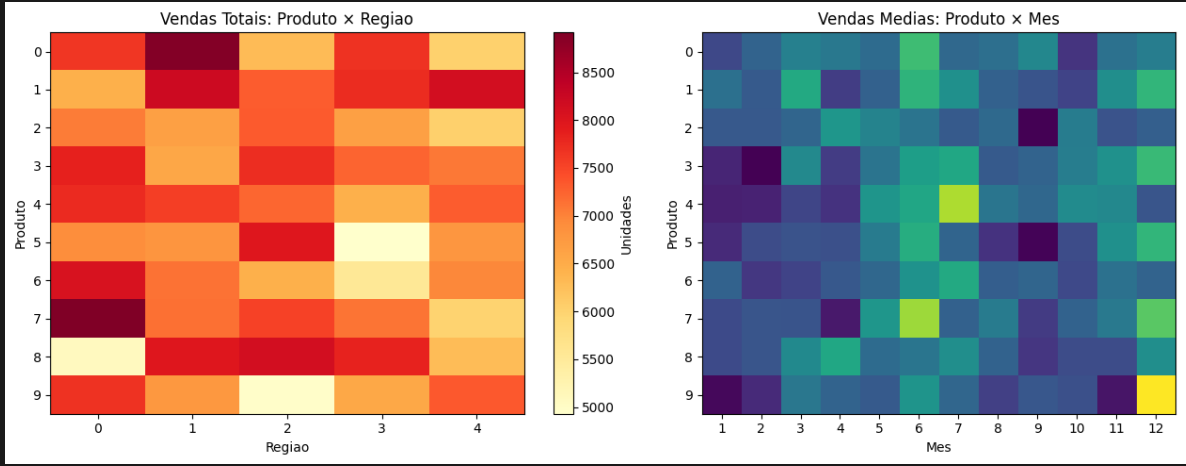
```python
# Visualizar vendas como heatmap (produto x regiao)

fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Heatmap 1: Total anual por produto-regiao
im1 = axes[0].imshow(sales_product_region, cmap='YlOrRd', aspect='auto')
axes[0].set_xlabel('Regiao')
axes[0].set_ylabel('Produto')
axes[0].set_title('Vendas Totais: Produto × Regiao')
axes[0].set_xticks(range(n_regions))
axes[0].set_yticks(range(n_products))
plt.colorbar(im1, ax=axes[0], label='Unidades')
```

Vendas 3D: Visualização como Heatmap (cont.)

Python

```python
# Heatmap 2: Vendas mensais (media por produto)
sales_product_month = sales.mean(axis=1)  # Media das regioes
im2 = axes[1].imshow(sales_product_month, cmap='viridis', aspect='auto')
axes[1].set_xlabel('Mes')
axes[1].set_ylabel('Produto')
axes[1].set_title('Vendas Medias: Produto × Mes')
axes[1].set_xticks(range(n_months))
axes[1].set_xticklabels(range(1, 13))
axes[1].set_yticks(range(n_products))
plt.colorbar(im2, ax=axes[1], label='Unidades')
plt.tight_layout()
plt.show()
```

Vendas 3D: Visualização como Heatmap (cont.)



Caso 3: Dados Volumétricos (Grid 3D)

Dados espaciais em 3 dimensões:

- **Exemplo:** Temperatura em um cubo espacial
- **Dimensão 0:** Eixo X (largura)
- **Dimensão 1:** Eixo Y (profundidade)
- **Dimensão 2:** Eixo Z (altura)
- **Shape:** (20, 20, 20) - 8000 pontos no espaço

Aplicações reais:

- Temperatura, pressão, umidade atmosférica
- Ressonância magnética (MRI) - voxels 3D
- Simulações de fluidos (CFD)
- Modelos climáticos e oceanográficos
- Geologia (densidade, porosidade subsolo)

Caso 3: Dados Volumétricos (Grid 3D)

Nota Importante

Visualizar dados 3D é desafiador - usamos slices 2D!

Grid 3D: Criar Dados Sintéticos

</> Python

```python
# Dimensoes do grid espacial
n_x, n_y, n_z = 20, 20, 20
# Gerar temperaturas com ruido
temp_grid = np.random.randn(n_x, n_y, n_z) * 3
# Adicionar gradiente vertical (mais frio conforme sobe)
z_gradient = np.linspace(30, 15, n_z)  # De 30°C (base) a 15°C (topo)
temp_grid = temp_grid + z_gradient[np.newaxis, np.newaxis, :]

print(f"Shape do grid 3D: {temp_grid.shape}")  # (20, 20, 20)
print(f"Total de pontos: {temp_grid.size}")  # 8000
print(f"Temperatura media: {temp_grid.mean():.2f}°C")
print(f"Temperatura min: {temp_grid.min():.2f}°C")
print(f"Temperatura max: {temp_grid.max():.2f}°C")
print(f"Std: {temp_grid.std():.2f}°C")
```

Grid 3D: Slices 2D de Dados 3D

Python

```python
# Extrair fatias 2D do volume 3D

# Slice no meio do eixo Z (altura = 10)
slice_z = temp_grid[:, :, 10]
print(f"Slice Z (altura=10): shape {slice_z.shape}")  # (20, 20)

# Slice no meio do eixo Y (profundidade = 10)
slice_y = temp_grid[:, 10, :]
print(f"Slice Y (prof=10): shape {slice_y.shape}")  # (20, 20)

# Slice no meio do eixo X (largura = 10)
slice_x = temp_grid[10, :, :]
print(f"Slice X (larg=10): shape {slice_x.shape}")  # (20, 20)
```

Grid 3D: Slices 2D de Dados 3D (cont.)

Python

```python
# Estatisticas por slice
print(f"\nTemperatura media no slice Z=10: {slice_z.mean():.2f}°C")
print(f"Temperatura media no slice Y=10: {slice_y.mean():.2f}°C")
print(f"Temperatura media no slice X=10: {slice_x.mean():.2f}°C")

# Temperatura em um ponto especifico (x=10, y=10, z=10)
temp_centro = temp_grid[10, 10, 10]
print(f"\nTemperatura no centro do cubo: {temp_centro:.2f}°C")
```

Grid 3D: Slices 2D de Dados 3D (cont.)

Python

```python
# Slice Z (altura=10): shape (20, 20)
# Slice Y (prof=10): shape (20, 20)
# Slice X (larg=10): shape (20, 20)

# Temperatura media no slice Z=10: 22.15°C
# Temperatura media no slice Y=10: 22.50°C
# Temperatura media no slice X=10: 22.19°C

# Temperatura no centro do cubo: 21.34°C
```

Grid 3D: Visualizar Slices

Python

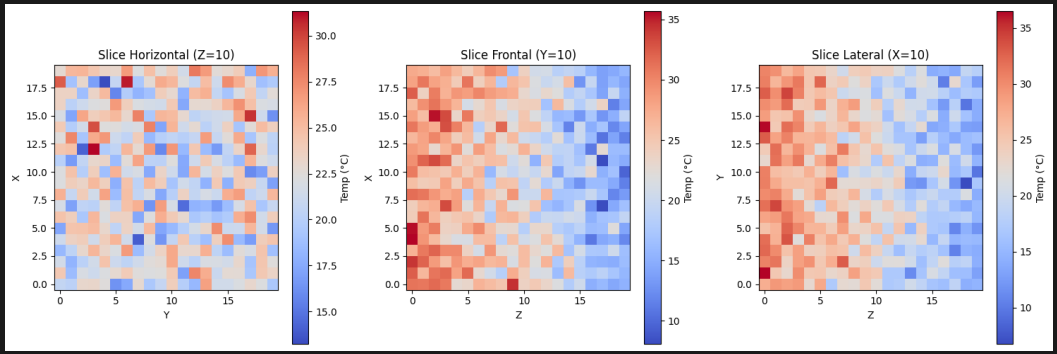
```python
# Visualizar diferentes slices 2D do volume 3D
fig, axes = plt.subplots(1, 3, figsize=(15, 5))
# Slice horizontal (z=10)
im0 = axes[0].imshow(slice_z, cmap='coolwarm', origin='lower')
axes[0].set_title('Slice Horizontal (Z=10)')
axes[0].set_xlabel('Y')
axes[0].set_ylabel('X')
plt.colorbar(im0, ax=axes[0], label='Temp (°C)')
# Slice frontal (y=10)
im1 = axes[1].imshow(slice_y, cmap='coolwarm', origin='lower')
axes[1].set_title('Slice Frontal (Y=10)')
axes[1].set_xlabel('Z')
axes[1].set_ylabel('X')
plt.colorbar(im1, ax=axes[1], label='Temp (°C)')
```

Grid 3D: Visualizar Slices (cont.)

Python

```python
# Slice lateral (x=10)
im2 = axes[2].imshow(slice_x, cmap='coolwarm', origin='lower')
axes[2].set_title('Slice Lateral (X=10)')
axes[2].set_xlabel('Z')
axes[2].set_ylabel('Y')
plt.colorbar(im2, ax=axes[2], label='Temp (°C)')

plt.tight_layout()
plt.show()
```

Grid 3D: Visualizar Slices (cont.)



Grid 3D: Operações Espaciais

Python

```python
# Operacoes ao longo de diferentes dimensoes espaciais

# Temperatura media em cada altura (agregar X e Y)
temp_per_height = temp_grid.mean(axis=(0, 1))
print(f"Temp por altura: shape {temp_per_height.shape}")  # (20,)
print(f"Temperaturas por altura (Z):\n{temp_per_height}")

# Verificar gradiente vertical
print(f"\nGradiente esperado: mais frio no topo")
print(f"Base (Z=0): {temp_per_height[0]:.2f}°C")
print(f"Meio (Z=10): {temp_per_height[10]:.2f}°C")
print(f"Topo (Z=19): {temp_per_height[19]:.2f}°C")
```

Grid 3D: Operações Espaciais (cont.)

Python

```python
# Volume com temperatura acima de 25°C
hot_volume = (temp_grid > 25).sum()
total_volume = temp_grid.size
print(f"\nVolume com temp > 25°C: {hot_volume}/{total_volume} pontos")
print(f"Percentual: {100*hot_volume/total_volume:.1f}%")

# Encontrar ponto mais quente
max_idx = np.unravel_index(temp_grid.argmax(), temp_grid.shape)
print(f"\nPonto mais quente: {max_idx}")
print(f"Temperatura: {temp_grid[max_idx]:.2f}°C")
```

Grid 3D: Agregações por Dimensão

Python

```python
# Diferentes agregacoes espaciais

# Projecao no plano XY (media ao longo de Z)
projection_xy = temp_grid.mean(axis=2)
print(f"Projecao XY: shape {projection_xy.shape}")  # (20, 20)

# Projecao no plano XZ (media ao longo de Y)
projection_xz = temp_grid.mean(axis=1)
print(f"Projecao XZ: shape {projection_xz.shape}")  # (20, 20)

# Projecao no plano YZ (media ao longo de X)
projection_yz = temp_grid.mean(axis=0)
print(f"Projecao YZ: shape {projection_yz.shape}")  # (20, 20)
```

Grid 3D: Agregações por Dimensão (cont.)

Python

```python
# Temperatura maxima em cada altura
max_per_height = temp_grid.max(axis=(0, 1))
print(f"\nMax por altura: {max_per_height}")

# Desvio padrao em cada altura (variabilidade espacial)
std_per_height = temp_grid.std(axis=(0, 1))
print(f"\nStd por altura:")
print(f"Base (Z=0): {std_per_height[0]:.2f}°C")
print(f"Topo (Z=19): {std_per_height[19]:.2f}°C")
```

Grid 3D: Visualizar Projeções

Python

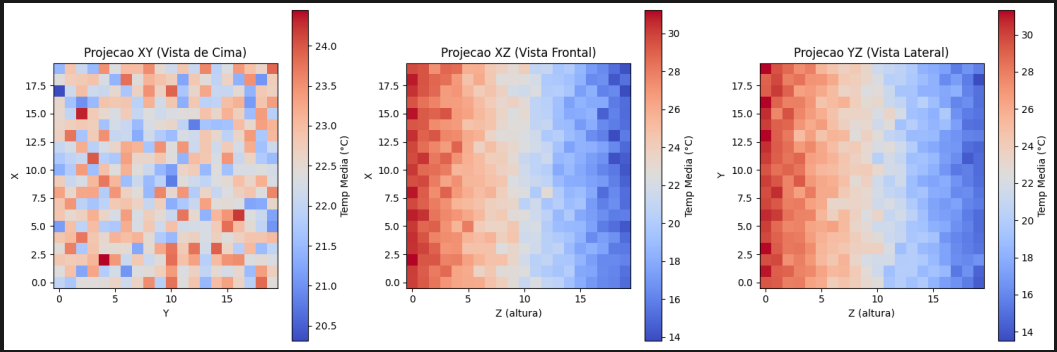
```python
# Visualizar projecoes em diferentes planos
fig, axes = plt.subplots(1, 3, figsize=(15, 5))
# Projecao XY (vista de cima)
im0 = axes[0].imshow(projection_xy, cmap='coolwarm', origin='lower')
axes[0].set_title('Projecao XY (Vista de Cima)')
axes[0].set_xlabel('Y')
axes[0].set_ylabel('X')
plt.colorbar(im0, ax=axes[0], label='Temp Media (°C)')

# Projecao XZ (vista frontal)
im1 = axes[1].imshow(projection_xz, cmap='coolwarm', origin='lower')
axes[1].set_title('Projecao XZ (Vista Frontal)')
axes[1].set_xlabel('Z (altura)')
axes[1].set_ylabel('X')
plt.colorbar(im1, ax=axes[1], label='Temp Media (°C)')
```

Grid 3D: Visualizar Projeções (cont.)

Python

```python
# Projecao YZ (vista lateral)
im2 = axes[2].imshow(projection_yz, cmap='coolwarm', origin='lower')
axes[2].set_title('Projecao YZ (Vista Lateral)')
axes[2].set_xlabel('Z (altura)')
axes[2].set_ylabel('Y')
plt.colorbar(im2, ax=axes[2], label='Temp Media (°C)')

plt.tight_layout()
plt.show()
```

Grid 3D: Visualizar Projeções (cont.)



Integração NumPy ↔ Pandas

Recap dos Blocos 2 e 3:

- Blocos 2-3: Trabalhamos com tensores puros (NumPy)
- Bloco 4: Pipeline completo - onde Pandas entra?

Integração NumPy ↔ Pandas

Quando usar cada ferramenta?

- **NumPy:** ideal para dados numéricos multidimensionais (3D+)
  - Operações vetorizadas rápidas
  - Broadcasting eficiente
  - Manipulação de shapes complexos
- **Pandas:** ideal para dados tabulares (2D) com labels
  - Colunas com nomes
  - Índices com significado
  - Operações por grupos
  - Facilidade de visualização

Solução: Combinar ambos!

- Pandas para leitura e exploração inicial
- NumPy para operações numéricas intensivas
- Pandas para apresentação final

DataFrame → Array 3D

Python

```python
# Criar DataFrame com dados de vendas (formato tabular)
# Cada linha: produto-regiao-mes
data_list = []
for prod in range(n_products):
    for reg in range(n_regions):
        for month in range(n_months):
            data_list.append({
                'produto': prod,
                'regiao': reg,
                'mes': month + 1,
                'vendas': sales[prod, reg, month]
            })
```

DataFrame → Array 3D (cont.)

Python

```python
df = pd.DataFrame(data_list)
print(f"DataFrame shape: {df.shape}")  # (600, 4)
print(df.head(10))

# Converter DataFrame para array 3D
sales_reconstructed = np.zeros((n_products, n_regions, n_months), dtype=
    int)
for _, row in df.iterrows():
    p, r, m = int(row['produto']), int(row['regiao']), int(row['mes']) - 1
    sales_reconstructed[p, r, m] = row['vendas']

print(f"\nArray 3D reconstruido: {sales_reconstructed.shape}")
print(f"Dados identicos? {np.array_equal(sales, sales_reconstructed)}")
```

Array 3D → DataFrame

Python

```python
# Converter array 3D para DataFrame (formato long)

# Criar listas para cada coluna
produtos_list = []
regioes_list = []
meses_list = []
vendas_list = []

for prod in range(n_products):
    for reg in range(n_regions):
        for month in range(n_months):
            produtos_list.append(prod)
            regioes_list.append(reg)
            meses_list.append(month + 1)
            vendas_list.append(sales[prod, reg, month])
```

Array 3D → DataFrame (cont.)

</> Python

```python
# Criar DataFrame
df_from_array = pd.DataFrame({
    'produto': produtos_list,
    'regiao': regioes_list,
    'mes': meses_list,
    'vendas': vendas_list
})

print(f"DataFrame criado: {df_from_array.shape}")
print(df_from_array.head())
print(f"\nTotal vendas (DataFrame): {df_from_array['vendas'].sum()}")
print(f"Total vendas (Array): {sales.sum()}")
```

Workflow Híbrido: Análise com Ambos

Python

```python
# Combinar pandas e numpy para analise eficiente

# 1. Usar Pandas para analise agregada facil
print("Analise com Pandas:")
print(df_from_array.groupby('produto')['vendas'].sum())

# 2. Usar NumPy para operacoes matematicas rapidas
print("\nAnalise com NumPy:")
sales_per_product_np = sales.sum(axis=(1, 2))
print(sales_per_product_np)
```

Workflow Híbrido: Análise com Ambos (cont.)

Python

```python
# 3. Combinar: NumPy para calculo, Pandas para apresentacao
# Calcular crescimento com NumPy
q1_np = sales[:, :, 0:3].sum(axis=(1, 2))
q4_np = sales[:, :, 9:12].sum(axis=(1, 2))
growth_np = q4_np - q1_np

# Apresentar com Pandas
df_summary = pd.DataFrame({
    'produto': range(n_products),
    'total_anual': sales_per_product_np,
    'q1': q1_np,
    'q4': q4_np,
    'crescimento': growth_np
})
print("\nSummary (NumPy + Pandas):")
print(df_summary)
```

Quando Usar Cada Ferramenta

USE NUMPY quando:

- Dados são puramente numéricos
- Operações matemáticas intensivas
- Broadcasting e vetorização complexa
- Dados têm 3+ dimensões
- Performance é crítica

Exemplo: média móvel 3D

</> Python

```python
window_3d = sales[:, :, -3:].mean(axis=2)  # NumPy rapido
```

Quando Usar Cada Ferramenta

USE PANDAS quando:

- Dados têm labels/nomes importantes
- Precisa de groupby, pivot, merge
- Visualização com plot()
- Dados tabulares (2D)
- Leitura/escrita de arquivos (CSV, Excel)

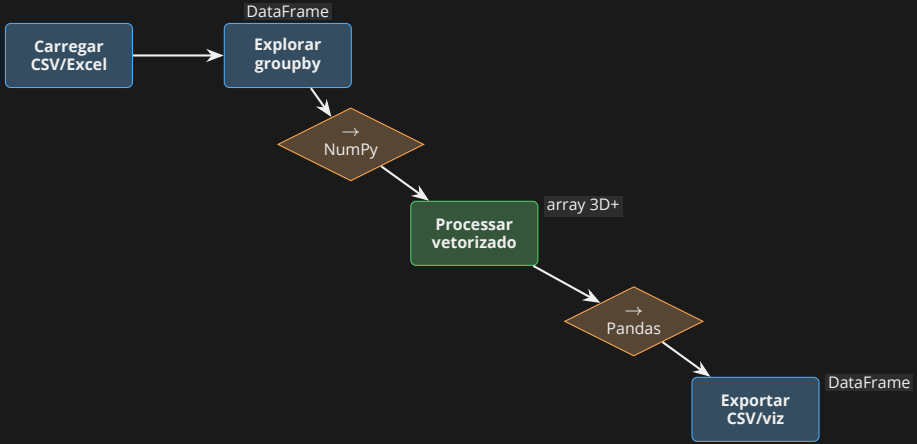
Exemplo: agrupar por categoria

</> Python

```python
df_grouped = df_from_array.groupby(['produto', 'regiao']).agg({
    'vendas': ['sum', 'mean', 'std']
})
```

Workflow Híbrido: NumPy + Pandas

Estratégia profissional para análise multi-dimensional:



Workflow Híbrido: NumPy + Pandas

Estratégia profissional para análise multi-dimensional:

Pipeline típico:

1. **Carregar:** Pandas (CSV, Excel) $\rightarrow$ DataFrame
2. **Explorar:** Pandas (groupby, describe)
3. **Converter:** DataFrame $\rightarrow$ NumPy array (tensores)
4. **Processar:** NumPy (operações vetorizadas rápidas)
5. **Resultados:** NumPy array $\rightarrow$ DataFrame (apresentação)
6. **Exportar:** Pandas (CSV, Excel, visualizações)

Vantagens:

- Pandas: I/O, labels, groupby, joins
- NumPy: performance em cálculos intensivos
- Melhor dos dois mundos!

Exercício Complementar: Análise Multi-dimensional

Para cada um dos datasets dos exemplos anteriores:

Sensores de Temperatura:

1. Identifique o dia mais quente e mais frio (média dos sensores)
2. Calcule correlação entre sensores diferentes
3. Crie visualização comparando todos os sensores
4. Implemente detecção de anomalias (valores 2σ acima da média)

Vendas 3D:

1. Identifique top 5 combinações produto-região-mês
2. Calcule taxa de crescimento trimestral por produto
3. Crie heatmap mostrando sazonalidade por produto
4. Converta para DataFrame e faça análise com groupby

Exercício Complementar: Análise Multi-dimensional

Para cada um dos datasets dos exemplos anteriores:

Grid 3D:

1. Encontre região (sub-cubo 5×5×5) mais quente
2. Calcule gradiente de temperatura em cada direção
3. Visualize múltiplos slices em diferentes alturas
4. Identifique pontos com temperatura extrema (outliers)

Resumo do Bloco 3

Exploramos diversos tipos de dados multidimensionais:

1. Séries Temporais Multivariadas:
- Estrutura: tempo × variáveis (sensores)
- Operações por eixo temporal e por variável
- Conexão com Aula 11 (séries 1D $\rightarrow$ séries multivariadas)

2. Dados de Vendas 3D:
- Estrutura: produtos × regiões × tempo
- Análise em múltiplas perspectivas simultaneamente
- Agregações complexas e identificação de padrões

3. Dados Volumétricos (Grid 3D):
- Estrutura: x × y × z (dados espaciais)
- Visualização via slices 2D
- Projeções em diferentes planos

Transição para Bloco 4

O que aprendemos:

- **Bloco 2:** Operações fundamentais com exemplo visual (imagens)
- **Bloco 3:** Mesmas operações em diferentes domínios

Tipos de dados trabalhados:

- ✓ Imagens (1797, 8, 8)
- ✓ Séries temporais multivariadas (100, 5, 3)
- ✓ Dados de vendas 3D (10, 5, 12)
- ✓ Dados volumétricos espaciais (20, 20, 20)

Ferramentas dominadas:

- Indexação, slicing, fancy/boolean indexing
- Broadcasting e newaxis
- Agregações com axis e keepdims
- Transformações de shape
- Operações vetorizadas

Transição para Bloco 4 (cont.)

Próximo: Bloco 4 - Integração e Aplicações Avançadas

1. **Pipeline End-to-End:**
   - Unir todos os conceitos em workflow completo
   - MNIST: da exploração até resultados finais
   - Código profissional e reutilizável
2. **Redução de Dimensionalidade (SVD):**
   - Compressão de dados multidimensionais
   - Trade-off qualidade vs tamanho
   - Aplicação visual em imagens
   - Comparação com PCA

Transição para Bloco 4 (cont.)

Próximo: Bloco 4 - Integração e Aplicações Avançadas

3. Performance e Otimização:

- Vetorização vs loops
- Broadcasting eficiente
- Memória e dtype

4. Boas Práticas:

- Validação, documentação, debugging
- Código limpo e profissional

Bloco 4

Pipeline Completo e Boas Práticas

Pipeline de Análise Multi-dimensional

Etapas de um pipeline completo:

1. **Carregar dados:** Importar e verificar estrutura
2. **Explorar shape:** Entender dimensões e significado
3. **Análise exploratória:** Estatísticas, padrões, outliers
4. **Transformações:** Normalizar, filtrar, agregar
5. **Visualizações:** Representar dados apropriadamente
6. **Exportar resultados:** Salvar insights e processamentos

Princípios importantes:

- Sempre verificar shapes após cada operação
- Visualizar dados intermediários
- Documentar o significado de cada dimensão
- Testar com subsets pequenos primeiro
- Validar resultados fazem sentido

Caso Prático: Pipeline MNIST Completo (1/6)

Etapa 1: Carregar Dados

Python

```python
from sklearn.datasets import load_digits
import numpy as np
import matplotlib.pyplot as plt
digits = load_digits()
X = digits.images  # (1797, 8, 8)
y = digits.target  # (1797,)
print("=== ETAPA 1: CARREGAR DADOS ===")
print(f"Images shape: {X.shape}")
print(f"Labels shape: {y.shape}")
print(f"Num classes: {len(np.unique(y))}")
print(f"Dtype: {X.dtype}")
print(f"Range: [{X.min()}, {X.max()}]")
```

Caso Prático: Pipeline MNIST Completo (2/6)

Etapa 2: Explorar estrutura

</> Python

```python
# 2. EXPLORAR ESTRUTURA
print(f"\n=== ETAPA 2: EXPLORAR ESTRUTURA ===")
print(f"Total imagens: {X.shape[0]}")
print(f"Dimensao imagem: {X.shape[1]} x {X.shape[2]}")
print(f"Total pixels por imagem: {X.shape[1] * X.shape[2]}")
print(f"Distribuicao de classes:")
for digit in range(10):
    count = (y == digit).sum()
    print(f"  Digito {digit}: {count} imagens")
```

Etapa 3: Análise Exploratória

Python

```python
# 3. ANALISE EXPLORATORIA
print("=== ETAPA 3: ANALISE EXPLORATORIA ===")

# Estatisticas globais
mean_global = X.mean()
std_global = X.std()
print(f"Media global: {mean_global:.2f}")
print(f"Std global: {std_global:.2f}")
```

Etapa 3: Análise Exploratória

Python

```python
# Estatisticas por digito
print(f"\nEstatisticas por digito:")
for digit in range(10):
    mask = y == digit
    X_digit = X[mask]
    mean_digit = X_digit.mean()
    std_digit = X_digit.std()
    brightness = X_digit.sum(axis=(1,2)).mean()  # Soma de pixels
    print(f"  Digito {digit}: mean={mean_digit:.2f}, std={std_digit:.2f},
    "
          f"brightness={brightness:.1f}")
```

Etapa 3: Análise Exploratória

Python

```python
# Identificar digito mais e menos brilhante
brightness_per_digit = np.array([X[y == d].sum(axis=(1,2)).mean()
                                 for d in range(10)])
brightest = brightness_per_digit.argmax()
darkest = brightness_per_digit.argmin()
print(f"\nDigito mais brilhante: {brightest}")
print(f"Digito mais escuro: {darkest}")
```

Caso Prático: Pipeline MNIST Completo (4/6)

Etapa 4: Transformações

</> Python

```python
# 4. TRANSFORMACOES
print("=== ETAPA 4: TRANSFORMACOES ===")

# 4.1 Normalizar para [0, 1]
X_normalized = (X - X.min()) / (X.max() - X.min())
print(f"Normalizado: range [{X_normalized.min():.2f}, {X_normalized.max()
    :.2f}]")

# 4.2 Centralizar (subtrair media)
X_centered = X - X.mean()
print(f"Centralizado: mean={X_centered.mean():.10f}")
```

Etapa 4: Transformações

Python

```python
# 4.3 Padronizar (z-score)
X_standardized = (X - X.mean()) / X.std()
print(f"Padronizado: mean={X_standardized.mean():.10f}, "
      f"std={X_standardized.std():.2f}")

# 4.4 Binarizar
threshold = 8
X_binary = (X > threshold).astype(int) * 16
n_white_pixels = (X_binary == 16).sum()
print(f"Binarizado (T={threshold}): {n_white_pixels} pixels brancos")
```

Etapa 4: Transformações

Python

```python
# 4.5 Criar imagens medias por digito
X_mean_per_digit = np.array([X[y == d].mean(axis=0) for d in range(10)])
print(f"Imagens medias: shape {X_mean_per_digit.shape}")  # (10, 8, 8)
```

Caso Prático: Pipeline MNIST Completo (5/6)

Etapa 5: Visualizações

Python

```python
# 5. VISUALIZACOES
print("=== ETAPA 5: VISUALIZACOES ===")

# 5.1 Grid de imagens originais
fig, axes = plt.subplots(2, 5, figsize=(12, 6))
for i in range(10):
    ax = axes[i // 5, i % 5]
    ax.imshow(X[i], cmap='gray')
    ax.set_title(f'Label: {y[i]}')
    ax.axis('off')
plt.suptitle('Primeiras 10 Imagens')
plt.tight_layout()
plt.show()
```

Caso Prático: Pipeline MNIST Completo (5/6) (cont.)

Etapa 5: Visualizações

Python

```python
# 5.2 Imagens medias por digito
fig, axes = plt.subplots(2, 5, figsize=(12, 6))
for digit in range(10):
    ax = axes[digit // 5, digit % 5]
    ax.imshow(X_mean_per_digit[digit], cmap='gray')
    ax.set_title(f'Media: {digit}')
    ax.axis('off')
plt.suptitle('Imagem Media por Digito')
plt.tight_layout()
plt.show()
```

Caso Prático: Pipeline MNIST Completo (6/6)

Etapa 6: Função de Processamento Customizada

Python

```python
# Criar funcao reutilizavel para processar imagens
def process_mnist_image(img, normalize=True, binarize=False,
                        threshold=8, flip=False, rotate=0):
    """
    Processar imagem MNIST com multiplas opcoes

    Parametros:
    - img: array 2D (8, 8)
    - normalize: normalizar para [0, 1]
    - binarize: aplicar threshold
    - threshold: valor do threshold
    - flip: flip horizontal
    - rotate: rotacao (0, 90, 180, 270)
    """
```

Caso Prático: Pipeline MNIST Completo (6/6) (cont.)

Etapa 6: Função de Processamento Customizada

Python

```python
    img_processed = img.copy()
    if normalize:
        img_processed = (img_processed - img_processed.min()) / \
                        (img_processed.max() - img_processed.min())
    if binarize:
        img_processed = (img_processed > threshold/16).astype(float)
    if flip:
        img_processed = img_processed[:, ::-1]
    if rotate in [90, 180, 270]:
        k = rotate // 90
        img_processed = np.rot90(img_processed, k=k)
    return img_processed
```

Caso Prático: Pipeline MNIST Completo (6/6) (cont.)

Usar função customizada:

Python

```python
# Testar funcao de processamento
img_test = X[42]

# Diferentes processamentos
processed_versions = [
    ("Original", img_test),
    ("Normalizado", process_mnist_image(img_test, normalize=True)),
    ("Binarizado", process_mnist_image(img_test, binarize=True) * 16),
    ("Flip", process_mnist_image(img_test, flip=True)),
    ("Rot 90", process_mnist_image(img_test, rotate=90)),
    ("Norm+Bin+Flip", process_mnist_image(img_test, normalize=True,
                                          binarize=True, flip=True) * 16)
]
```

Caso Prático: Pipeline MNIST Completo (6/6) (cont.)

Usar função customizada:

Python

```python
# Visualizar
fig, axes = plt.subplots(2, 3, figsize=(12, 8))
for ax, (title, img) in zip(axes.flat, processed_versions):
    ax.imshow(img, cmap='gray')
    ax.set_title(title)
    ax.axis('off')
plt.tight_layout()
plt.show()

print("Pipeline completo executado com sucesso!")
```

Performance e Otimização

Princípios para código eficiente:

1. Vetorização:
- Evitar loops Python quando possível
- Usar operações NumPy nativas
- Broadcasting automático

2. Broadcasting:
- Deixar NumPy expandir dimensões automaticamente
- Mais eficiente que loops explícitos
- Entender regras de broadcasting

3. Views vs Copies:
- Views: referência à memória original (rápido)
- Copies: nova alocação (lento)
- Usar .copy() apenas quando necessário

Performance: Vetorização vs Loops

</> Python

```python
import time

# Criar array de teste
test_array = np.random.rand(1000, 100, 100)

# LENTO: Loop Python
start = time.time()
result_loop = np.zeros_like(test_array)
for i in range(test_array.shape[0]):
    result_loop[i] = test_array[i] + 10
time_loop = time.time() - start
```

Performance: Vetorização vs Loops (cont.)

Python

```python
# RAPIDO: Vetorizado
start = time.time()
result_vectorized = test_array + 10
time_vectorized = time.time() - start

print(f"Tempo com loop: {time_loop:.4f}s")
print(f"Tempo vetorizado: {time_vectorized:.4f}s")
print(f"Speedup: {time_loop/time_vectorized:.1f}x mais rapido")

# Verificar igualdade
print(f"Resultados identicos: {np.allclose(result_loop, result_vectorized)
    }")
```

Performance: Vetorização vs Loops (cont.)

Nota Importante

Vetorização pode ser 10-100x mais rápida que loops!

Performance: Broadcasting Eficiente

Python

```python
# Broadcasting é muito mais eficiente que loops

# Exemplo: Subtrair media de cada imagem

# LENTO: Loop explícito
start = time.time()
X_centered_loop = np.zeros_like(X, dtype=float)
for i in range(X.shape[0]):
    X_centered_loop[i] = X[i] - X[i].mean()
time_loop = time.time() - start
```

Performance: Broadcasting Eficiente (cont.)

Python

```python
# RAPIDO: Broadcasting
start = time.time()
means = X.mean(axis=(1, 2))  # (1797,)
X_centered_broadcast = X - means[:, np.newaxis, np.newaxis]  # Broadcasting
time_broadcast = time.time() - start

print(f"Tempo com loop: {time_loop:.4f}s")
print(f"Tempo com broadcasting: {time_broadcast:.4f}s")
print(f"Speedup: {time_loop/time_broadcast:.1f}x")

# Verificar corretude
print(f"Resultados identicos: {np.allclose(X_centered_loop, X_centered_broadcast)}")
print(f"Media apos centralizacao: {X_centered_broadcast.mean():.10f}")  # ~0
```

Performance: Views vs Copies

Python

```python
# Entender quando NumPy cria views vs copies

# VIEW: Slicing basico
arr = np.arange(100).reshape(10, 10)
arr_slice = arr[2:5, 3:7]
print(f"Slice eh view? {np.shares_memory(arr, arr_slice)}")  # True

# VIEW: Transpose, reshape (quando possivel)
arr_T = arr.T
print(f"Transpose eh view? {np.shares_memory(arr, arr_T)}")  # True
```

Performance: Views vs Copies (cont.)

</> Python

```python
# COPY: Fancy indexing
indices = [1, 3, 5]
arr_fancy = arr[indices]
print(f"Fancy indexing eh view? {np.shares_memory(arr, arr_fancy)}")  # False

# COPY: Boolean indexing
mask = arr > 50
arr_masked = arr[mask]
print(f"Boolean indexing eh view? {np.shares_memory(arr, arr_masked)}")  # False
```

Performance: Views vs Copies (cont.)

Python

```python
# Criar copy explicitamente quando necessario
arr_copy = arr.copy()
print(f"Copy explicita eh view? {np.shares_memory(arr, arr_copy)}")  # False

print("\nViews sao rapidas mas modificam o original!")
```

Performance: Memória e dtype

</> Python

```python
# Escolher dtype apropriado pode economizar muita memoria

# Exemplo: imagens MNIST (range 0-16)

# Float64: 8 bytes por elemento
X_float64 = X.astype(np.float64)
size_float64 = X_float64.nbytes / (1024**2)  # MB

# Float32: 4 bytes por elemento (suficiente!)
X_float32 = X.astype(np.float32)
size_float32 = X_float32.nbytes / (1024**2)
```

Performance: Memória e dtype (cont.)

Python

```python
# Int8: 1 byte por elemento (range -128 to 127, suficiente para 0-16)
X_int8 = X.astype(np.int8)
size_int8 = X_int8.nbytes / (1024**2)

print(f"Memoria float64: {size_float64:.2f} MB")
print(f"Memoria float32: {size_float32:.2f} MB (economiza {100*(1-
    size_float32/size_float64):.0f}%)")
print(f"Memoria int8: {size_int8:.2f} MB (economiza {100*(1-size_int8/
    size_float64):.0f}%)")

# Verificar dados nao foram perdidos
print(f"\nDados preservados em float32? {np.allclose(X, X_float32)}")
print(f"Dados preservados em int8? {np.array_equal(X, X_int8)}")
```

Performance: Benchmark de Operações

Python

```python
# Comparar performance de diferentes abordagens

import time

def benchmark(func, *args, n_runs=100):
    """Medir tempo medio de execucao"""
    times = []
    for _ in range(n_runs):
        start = time.time()
        func(*args)
        times.append(time.time() - start)
    return np.mean(times)
```

Performance: Benchmark de Operações (cont.)

Python

```python
# Funcoes para testar
def sum_loop(arr):
    total = 0
    for i in range(arr.shape[0]):
        for j in range(arr.shape[1]):
            total += arr[i, j]
    return total

def sum_vectorized(arr):
    return arr.sum()
```

Performance: Benchmark de Operações (cont.)

Python

```python
# Benchmark
arr_test = np.random.rand(100, 100)
time_loop = benchmark(sum_loop, arr_test, n_runs=10)
time_vec = benchmark(sum_vectorized, arr_test, n_runs=100)

print(f"Loop Python: {time_loop*1000:.2f} ms")
print(f"NumPy sum(): {time_vec*1000:.2f} ms")
print(f"Speedup: {time_loop/time_vec:.0f}x")
```

Boas Práticas em Arrays Multidimensionais

1. Sempre verificar shapes:

- Use print(array.shape) frequentemente
- Adicione asserts para validar: assert X.shape == (n, m, k)
- Confira dimensões após cada operação importante

2. Documentar dimensões:

- Comente o significado de cada dimensão
- Exemplo: `# Shape: (batch, height, width, channels)`
- Use nomes descritivos: n_samples, n_features

Boas Práticas em Arrays Multidimensionais

3. Usar axis explicitamente:

- arr.mean(axis=0) é mais claro que arr.mean()
- Evita ambiguidade em operações
- Facilita manutenção e debugging

4. Validar shapes em funções:

- Verificar inputs no início das funções
- Lançar erros descritivos se shape incorreto
- Documentar shapes esperados

Boas Práticas: Exemplo de Código Bem Documentado

</> Python

```python
def process_image_batch(images, normalize=True, target_shape=(8, 8)):
    """
    Processar batch de imagens com validacao

    Parametros:
    -----------
    images : ndarray
        Array 3D com shape (n_images, height, width)
    normalize : bool
        Se True, normaliza para [0, 1]
    target_shape : tuple
        Shape esperado de cada imagem (height, width)
    """
```

Boas Práticas: Exemplo de Código Bem Documentado (cont.)

Python

```
    """
    Retorna:
    --------
    processed : ndarray
        Array processado com mesmo shape do input
    """
    # Validar shape do input
    assert images.ndim == 3, f"Esperado 3D, recebido {images.ndim}D"
    assert images.shape[1:] == target_shape, \
        f"Esperado {target_shape}, recebido {images.shape[1:]}"

    processed = images.copy()  # (n_images, height, width)
```

Boas Práticas: Exemplo de Código Bem Documentado (cont.)

Python

```python
    if normalize:
        # Normalizar cada imagem individualmente
        min_vals = processed.min(axis=(1,2), keepdims=True)  # (n, 1, 1)
        max_vals = processed.max(axis=(1,2), keepdims=True)  # (n, 1, 1)
        processed = (processed - min_vals) / (max_vals - min_vals)

    return processed
```

Erros Comuns - Evite!

✗ Erro 1: Confundir axis

- arr.mean(axis=0) ≠ arr.mean(axis=1)
- Sempre visualize mentalmente qual dimensão está sendo agregada

✗ Erro 2: Broadcasting incompatível

- (10, 5, 12) + (5, 12) ✓ | (10, 5, 12) + (5,) ✗
- Use np.newaxis para adicionar dimensões

✗ Erro 3: Modificar view inadvertidamente

- slice = arr[2:5]; slice[0] = 999 modifica arr!
- Use .copy() quando precisar de independência

Erros Comuns - Evite!

✗ Erro 4: Esquecer dtype

- int_arr / 2 pode perder precisão se não converter
- Sempre considere o dtype do resultado

✗ Erro 5: Não validar shapes

- Assumir shape sem verificar leva a bugs sutis
- print(shape) é seu amigo!

Debugging de Shapes

Python

```python
# Tecnicas para debugar problemas de shape

def debug_array(arr, name="array"):
    """Imprimir informacoes detalhadas sobre array"""
    print(f"\n=== {name} ===")
    print(f"Shape: {arr.shape}")
    print(f"Ndim: {arr.ndim}")
    print(f"Dtype: {arr.dtype}")
    print(f"Size: {arr.size}")
    print(f"Min: {arr.min():.2f}, Max: {arr.max():.2f}")
    print(f"Mean: {arr.mean():.2f}, Std: {arr.std():.2f}")
    print(f"Memory: {arr.nbytes / 1024:.2f} KB")
```

Debugging de Shapes (cont.)

Python

```python
# Usar durante desenvolvimento
debug_array(X, "MNIST images")
debug_array(y, "MNIST labels")

# Para operacoes complexas, verificar shapes intermediarios
arr = X[:10]  # (10, 8, 8)
print(f"Apos selecao: {arr.shape}")

arr_mean = arr.mean(axis=(1,2))  # (10,)
print(f"Apos mean: {arr_mean.shape}")

arr_expanded = arr_mean[:, np.newaxis, np.newaxis]  # (10, 1, 1)
print(f"Apos expansao: {arr_expanded.shape}")
```

Resumo do Bloco 4

Pipeline Completo e Boas Práticas:

1. Pipeline End-to-End:

- 6 etapas completas: carregar $\rightarrow$ explorar $\rightarrow$ analisar $\rightarrow$ transformar $\rightarrow$ visualizar $\rightarrow$ exportar
- Função customizada reutilizável para processamento
- Integração de todos os conceitos da aula

2. Performance e Otimização:

- Vetorização > loops Python (10-100x mais rápido)
- Broadcasting eficiente para operações em batch
- Views vs copies (gerenciamento de memória)
- Dtype apropriado (economia substancial de memória)

Resumo do Bloco 4

3. Boas Práticas Profissionais:

- Sempre verificar shapes após operações
- Documentar dimensões e significado de cada axis
- Usar axis explicitamente para clareza
- Validar inputs no início de funções
- Debugging sistemático com função helper

4. Código Limpo:

- Evitar erros comuns (axis, broadcasting, views)
- Funções reutilizáveis e bem documentadas
- Validação rigorosa de shapes
- Comentários estratégicos

Recap Completo da Aula 12

O que aprendemos hoje:

Bloco 1 - Tensores e Dados 3D+:

- Conceito de tensores (rank 0-4+)
- Shape, indexação e slicing multi-dimensional
- Parâmetro axis em profundidade
- Broadcasting em 3D+

Bloco 2 - Imagens:

- Imagens como tensores (grayscale e RGB)
- MNIST: análise exploratória completa
- Filtros: binarização, blur, sharpen, bordas
- Transformações geométricas: flip, rotação, crop, pad
- Mosaicos e grids

Recap Completo da Aula 12 (cont.)

Bloco 3 - Outros Dados 3D+:

- Séries temporais multivariadas
- Dados de vendas 3D (produtos × regiões × tempo)
- Dados volumétricos (grids espaciais)
- Integração NumPy $\leftrightarrow$ Pandas

Bloco 4 - Pipeline e Boas Práticas:

- Pipeline completo end-to-end com MNIST (6 etapas)
- Função customizada reutilizável para processamento
- Performance: vetorização (10-100x), broadcasting, memória
- Boas práticas: documentação, validação, debugging sistemático

Recap Completo da Aula 12 (cont.)

Habilidades desenvolvidas:

- ✓ Manipular tensores de qualquer dimensionalidade
- ✓ Entender e usar axis corretamente
- ✓ Aplicar operações vetorizadas eficientemente
- ✓ Processar imagens com NumPy
- ✓ Analisar dados multi-dimensionais complexos
- ✓ Integrar NumPy e Pandas apropriadamente
- ✓ Escrever código eficiente e bem documentado

Conexões com Outras Aulas do Curso

Aula 12 integra conceitos de TODO o curso:

- **Aulas 1-2:** Estruturas de dados básicas $\rightarrow$ tensores complexos
- **Aulas 3-4:** NumPy básico $\rightarrow$ NumPy avançado (3D+)
- **Aulas 5-6:** Pandas $\rightarrow$ integração NumPy+Pandas
- **Aulas 7-8:** EDA em 2D $\rightarrow$ EDA em 3D+
- **Aulas 9-10:** Pré-processamento $\rightarrow$ transformações avançadas
- **Aula 11:** Séries temporais 1D $\rightarrow$ multivariadas 2D/3D

Aplicações práticas vistas:

- Processamento de imagens (filtros, transformações)
- Análise de vendas multi-dimensional
- Sensores e séries temporais multivariadas
- Dados científicos (volumétricos, espaciais)

Aplicações Práticas e Futuro

Onde você usará esses conceitos:

Visão Computacional:

- Pré-processamento de imagens
- Detecção de características
- Preparação para ML/Deep Learning

Análise de Negócios:

- Análise de vendas multi-dimensional
- Forecast com múltiplas variáveis
- Dashboards e visualizações complexas

Ciência de Dados:

- Feature engineering para ML
- Redução de dimensionalidade (PCA, SVD)
- Análise de dados científicos complexos

Recursos para Continuar Aprendendo

Documentação oficial:

- NumPy User Guide: `https://numpy.org/doc/stable/user/`
- Array manipulation: `https://numpy.org/doc/stable/reference/routines.array-manipulation.html`
- Broadcasting: `https://numpy.org/doc/stable/user/basics.broadcasting.html`

Tutoriais recomendados:

- NumPy: the absolute basics for beginners
- From Python to NumPy (livro online gratuito)
- Real Python - NumPy tutorials

Prática adicional:

- Kaggle datasets com imagens
- Exercícios: 100 NumPy exercises
- Projetos pessoais com dados reais

Recursos para Continuar Aprendendo

Comunidade:

- Stack Overflow (tag: numpy)
- Reddit: r/learnpython, r/datascience
- NumPy discussions no GitHub

Próximos passos (opcional):

- TensorFlow/PyTorch (tensores para Deep Learning)
- OpenCV (processamento de imagens profissional)
- Scikit-learn (redução de dimensionalidade avançada)

A Seguir: Aula Prática

Exercício Prático

Tempo: 60 minutos
Entrega: via Moodle (notebook)

Tarefas:

1. (atualizado durante a aula)

Notebook: Disponível no Moodle

Obrigado!